Statistical Modeling as a Search for Randomness Deficiencies

乱数欠陥の探索による統計的モデル化

Daniel Burfoot

Graduate School of Information Science and Technology
University of Tokyo

Supervisor: Yasuo Kuniyoshi

A thesis submitted for the degree of

Doctor of Philosophy

March 2010

This thesis is dedicated to my parents.

Acknowledgements

This thesis would not have been possible without the help of many people. Professor Yasuo Kuniyoshi has all the best qualities one could hope for in a supervisor. He is full of insights and advice, not only about research topics, but about the wider world as well. He has also been very generous in allowing me the freedom to pursue topics of special interest to me. Most importantly, he is a friendly and warm-hearted person.

Professor Tatsuya Harada is an inspiration as a hard-working and brilliant researcher, and is also very fun to talk to. I am very thankful to him for many comments on this thesis and on my other work.

I would also like to thank the students of the ISI laboratory, who are friendly, smart, diligent, and helpful. I especially thank Cota Nabeshima, who helped me get adjusted to life in the lab and in Japan. I would also like to thank the secretaries Sachiko Tamura and Mioko Tomaru, who were always willing to help me with various forms and paperwork, and always did so with a smile.

Dr. Max Lungarella played a key role in this thesis, by encouraging me to become interested in information theory. Max is a brilliant and friendly guy, and also a great gym partner. I am grateful to Dr. Tetsunari Inamura for many interesting conversations and for assistance with the research in Chapter 10. Martin Budi provided crucial assistance for the speech modeling research described in Chapter 9. Genevieve Patterson helped with the proof of monotonicity of $h(\beta)$ mentioned in Chapter 4.

My studies were supported by a scholarship from the Japanese Ministry of Education, Culture, Sports, Science and Technology.

Finally, I would like to thank my family for their unwavering love and support over many years.

Abstract

The basic goal of statistical modeling is to find a good approximation $Q(x)$ of an unknown real world distribution $P(x)$, by using an observed data set $\mathcal{X}$. If $\mathcal{X}$ is encoded as a bit string using the distribution $P(x)$ that generated it, then the resulting bit string will be random. Thus we can determine if a model $Q(x)$ is good by using it to encode the data set, and then searching the resulting bit string for randomness deficiencies.

This thesis develops an algorithm called PITA based on the above idea. Instead of bits, PITA uses the Probability Integral Transform to encode the data set as a sequence of values distributed on the $[0, 1)$ interval. A randomness deficiency is revealed by an uneven histogram of PIT values. If an uneven PIT histogram is found, an update based on the histogram can be applied to improve the model. The improvement in log-likelihood achieved by applying this update is a simple function of the histogram bin counts. The PITA algorithm uses this model update method, along with a set of feature functions, to produce a complex feature-based conditional model $Q(x|c)$, where c is a context. PITA proceeds iteratively, and in each round a smart feature scoring method is used to select the feature that will provide the best model improvement.

The PITA algorithm is quite general. This thesis demonstrates how the algorithm can be applied to the tasks of image compression, binary pattern classification, speech and word morphology modeling, and motion recognition. A key advantage of PITA is that it can be layered over some other initial model of the data. In the word morphology application, bigram models are used for initialization, while in the motion recognition application, Hidden Markov Models are used. In both cases the combined model provides improved performance.

Contents

List of Figures

Chapter 1

Introduction

1.1 Three Views of Statistical Modeling

This thesis advances a new perspective on the problem of statistical modeling. Statistical modeling is the attempt to obtain, on the basis of an empirical data set $\mathcal{X} = \{X^1, X^2 \ldots X^N\}$, an estimate for the probability distribution $P(x)$ that generated the data. This goal immediately raises a variety of philosophical and technical questions. One central question is: given that the real distribution is unknown, how can the quality of the estimate be assessed?

The following list gives three answers to that question, corresponding to three different perspectives of statistical modeling. The views are deeply related to one another, but contain important differences of emphasis. Each view has a principle, and a formulation, as follows:

- Traditional Statistics: Maximum Likelihood Principle,
  Formulation: Choose model class $\mathcal{M}$, search for optimal model parameters θ^*
- Information Theory: Maximum Compression Principle,
  Formulation: Choose model class $\mathcal{M}$, search for optimal model parameters θ^*
- Algorithmic Information Theory (AIT): Maximum Compression Principle,
  Formulation: Encode data as bit string, search for randomness deficiencies

As the list shows, the transition from the traditional statistics view to the AIT view involves two steps. The first step suggests a new goal for the modeling process. The

second step suggests a new way to achieve the goal. Considered separately, both steps are quite small. Together, they lead to a substantially different perspective on statistical modeling. This thesis is about the AIT view, but for the sake of conceptual development a brief discussion of the other two views is now given.

In traditional statistics one employs the Maximum Likelihood Principle. This principle suggests that given a model class $\mathcal{M}$ with associated parameters θ, we should choose the optimal parameters θ^* that maximize the probability of the data given the model:

$$\theta^* = \arg\max_{\theta} P(\mathcal{X}|\theta) \tag{1.1}$$

The Maximum Likelihood Principle is well established, and can be used to solve many problems. However, it has certain conceptual drawbacks. One drawback is that it can lead to overfitting. If the model class $\mathcal{M}$ is complex enough, then the fact that some choice of parameters assigns high probability to the data set tells us nothing. The standard example of overfitting happens when one attempts to fit a Gaussian mixture model with 100 components to a set of 100 data points. By choosing the means of the mixture components to be exactly equal to the data points, and setting the variances to nearly zero, a very high probability can be achieved. However, this "comb" model is obviously worthless: it has simply overfit the data.

A second drawback to the Maximum Likelihood idea is that it is difficult to connect the notion of "likelihood" to anything tangible in the real world. The statement that a model assigns a probability of $8.5 \cdot 10^{-12494}$ to a data set is not intuitively significant. Furthermore, if a complex model is used to describe a large data set, it can be difficult to verify the legitimacy of the model.

To understand this point, imagine a Maximum Likelihood Contest in which the goal is to find a model that assigns the highest probability to a complex data set $\mathcal{X}$. The current top ranked model achieves a likelihood of $3.4 \cdot 10^{-15112}$ on the data set. Then a challenger submits a new model, implemented in a software program. The referee tests the program by invoking it on the data set. After running for a while, the program prints out:

> Modeling complete: likelihood is $7.7 \cdot 10^{-10752}$.

If this claim is legitimate, then the new model is superior and should be declared the new leader of the contest. But it is very difficult for the judge to verify that the model follows the rules of probability theory and that the software does not contain any bugs.

A more satisfying view of statistical modeling comes from replacing the Maximum Likelihood Principle with the Maximum Compression Principle. Now the goal is to losslessly compress the data set $\mathcal{X}$ to the smallest possible size. Lossless data compression is basically a hard problem, because the number of short codes is intrinsically limited. For example, there are only $2^3 = 8$ codes of length 3 bits. Thus, to obtain the shortest net codelengths, one must assign the scarce short codes to the common outcomes, while using longer codes for the uncommon outcomes.

The theory of information, due to Shannon (1948), provides an exact answer to the question of how to choose optimal codelengths. Shannon showed that the optimal codelength for an outcome x is related to the probability of x by the equation $L(x) = -\log_2 P(x)$. Any other codelength function $L(x)$ will produce larger expected codelengths. Therefore, the problem of compression now becomes a problem of statistical modeling: to get the shortest possible codes, it is necessary to obtain an estimate $Q(x)$ of the distribution $P(x)$ that generated the data. The better the estimate, the shorter the resulting codes will be. Profoundly, then, the Maximum Compression Principle leads to the same formulation as the Maximum Likelihood Principle. Given a model class $\mathcal{M}$ with associated parameters θ, one chooses optimal parameters θ^* by minimizing $L(\mathcal{X})$:

$$\theta^* = \arg\min_{\theta} -\log_2 P(\mathcal{X}|\theta) \tag{1.2}$$

$$= \arg\max_{\theta} P(\mathcal{X}|\theta) \tag{1.3}$$

One advantage of the Information Theory view is that codelengths are a much more tangible quantity than likelihoods. When someone says he has compressed a database $\mathcal{X}$ to a length of $4.8 \cdot 10^6$ bits, it is quite clear what this means. If the previous best compression result was $7.3 \cdot 10^6$ bits, then it is obvious that the new model is superior to the previous one. Furthermore, adopting the codelength perspective solves the problem of verifying the legitimacy of complex models implemented in software. One simply takes the compressed data file, measures its length in bits, and then decompresses it. If

the decompressed version of the data does not exactly match the original version, then there is a bug in the model. For this reason, a Maximum Compression Contest is much easier to referee than a Maximum Likelihood Contest [1].

It is also worth noting that a variant of the Maximum Compression Principle can be used to avoid overfitting. This is called the Minimum Description Length Principle [Rissanen (1978)]. This principle instructs us to calculate the codelength required to encode the model itself, and then attempt to minimize the total codelength of the encoded data plus the model.

The focus of this thesis is on the third item in the above list, the AIT view of statistical modeling. This view starts from the Maximum Compression Principle and then asks the following general question: when is it possible to compress a bit string? The answer comes from the field of Algorithmic Information Theory (AIT), and in particular from the theory of randomness [Li & Vitanyi (1997); Martin-Löf (1966)]. The deep realization is that a bit string can be compressed if and only if it has a **randomness deficiency**. To detect a randomness deficiency in a bit string, one applies a statistical test to it. A randomness deficiency is revealed if the result of the test deviates substantially from what would be expected from a random string. For example, consider a statistical test that measures the frequency of 1's occurring after the prefix 0110. In a random string, this frequency should be nearly 50%. If the measured frequency in the encoded bit string is substantially different from 50%, this indicates a randomness deficiency.

If a randomness deficiency is present, it can be exploited to reencode the string; the resulting string will be shorter and more random. Thus, instead of working with probabilities as is generally done in statistical modeling, the basic strategy of the AIT view is to encode the dataset $\mathcal{X}$ and then search the encoded data for randomness deficiencies. We elaborate on this idea in the next section.

1.2 AIT View of Modeling

A basic description of the AIT view of modeling is as follows. First, the data set $\mathcal{X}$ is transformed into a bit string using some standard encoding scheme (such as ASCII text). Then a large battery of statistical tests is applied to the string. If one of the

[1] Indeed, several such contests already exist, for example the Hutter Prize: http://prize.hutter1.net/.

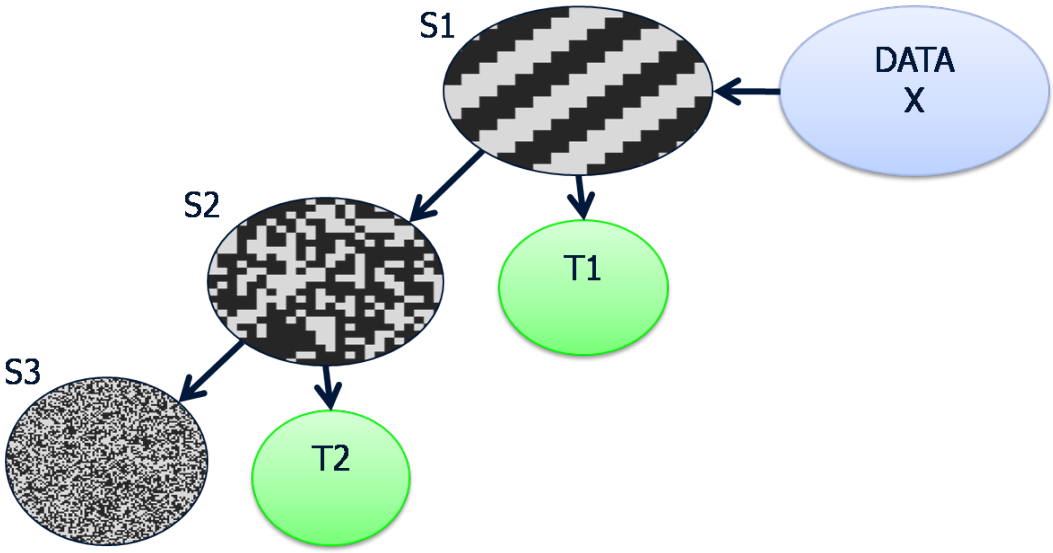


Figure 1.1: Key Figure of thesis: the AIT view of statistical modeling. The data is first encoded using some naïve encoding method such as ASCII text. Then the encoded data (S_i) is searched for randomness deficiencies. If one is found, a green transformation can be applied to reencode the string; the resulting string is shorter and more random. At each stage, we have $T_i(S_{i+1}) = S_i$, so that the original data can always be reconstructed.

tests reveals a randomness deficiency, it is used to compress the string. The resulting reencoded string is shorter and more random. The search and reencode process is repeated until all the statistical tests indicate that the string is random. An illustration of this idea is shown in the Key Figure 1.1. Note that the green transformations are obtained simply by combining a statistical test with an appropriate set of parameters.

The traditional view of statistics involves two basic objects of interest: the real data and the model. The AIT view intoduces a fundamentally new object: the encoded data. As discussed below, feature selection and feature combination are hard problems in the traditional view. But by introducing the concept of encoded data, these problems become almost trivial.

An important realization is that the set of transformations T_i in the AIT view plays the same role as a statistical model in the traditional view. To find the probability of a new data set $\mathcal{X}'$, the green transformations are applied to it, resulting in a new encoded bit string of length $L(\mathcal{X}')$. The corresponding probability is $P(\mathcal{X}') = 2^{-L(\mathcal{X}')}$.

Any data set can be thought of as containing two parts: a structure component and a purely random component. In this view, the goal of statistical modeling is to separate or "distill" the structure from the randomness. In the Key Figure 1.1, the green transformations represent the structure, while the encoded data remaining at the end of the process is the pure randomness.

Note also that the green transformations of the Key Figure 1.1 can be run both ways, corresponding to encoding or decoding. If the decoding transformations are applied to the bit string remaining at the end of the modeling process, the original data set is reconstructed. If, on the other hand, the decoding transformations are applied to a new random bit string, the result is a sample from the model. The resulting sampled dataset should "look like" the original data. This technique becomes very interesting when one considers more complex models for things like images or sentences. If a good model for the probability of a sentence can be learned, then sampling from the model will produce "typical" sentences. In practice it is generally impossible to obtain a perfect model for complex data types, so the sampled data will differ from the real data. This is also useful: by analyzing the differences between the real data and the sampled data, one can discover the limitations of the current model. Then the model can be modified to repair the limitations.

A key point regarding the idea of randomness deficiencies is that, for a string to be random and therefore incompressible, it must satisfy a very large number of statistical tests. Consider a binary string S. Then to be random, the frequency of 1s must be approximately 50%, the frequency of the substring 01 or any other 2-bit block must be about 25%, the frequency of the substring 011, or any other 3-bit block, must be about 12.5%, and so on for longer blocks. Furthermore, the conditional frequencies obtained when counting only blocks that satisfy some condition, should be exactly the same as the non-conditional frequencies. For example, the frequency of the block 101 observed after the prefix 010 should be about 12.5%. Similar statements can be made using any other prefix. One can imagine a great variety of exotic conditions. In a random string the frequency of 01 blocks in positions indexed by triangular numbers (1, 3, 6, 10 . . .) must be approximately 25%. If one indexes positions by the recursion relation $g_{i+1} = g_i + n_i$, where n_i is the ith digit in the decimal expansion of $\sqrt{2}$, then the frequency of 0011 at positions S_{g_i} must be approximately 6.25%.

An interesting connection exists between the AIT view of modeling and the much-studied problem of profiting from the stock market. Stock traders implement the same basic process of searching for and exploiting randomness deficiencies. For stocks the encoded data is the stream of price changes. If a randomness deficiency is found in this stream, it can be exploited to make money. For example, if one finds that Microsoft stock usually goes up on Friday, then one can exploit this fact by buying Microsoft stock on Thursday. Once someone starts doing this, the structure is removed: instead of usually going up on Friday, the stock will now go up (less) on Thursday. Again, there is an enormous number of possible types of randomness deficiencies. It is possible that Microsoft stock always goes up on days when Google stock goes down and the dollar-yen exchange rate changes by more than 5%. The idea that the path of stock prices should look very much like a random walk is known as the "efficient market hypothesis" in economics [Fama (1970)].

It should be noted that the AIT view of modeling is not necessarily superior to the other views. All of the same deep philosophical issues remain. The problem of overfitting occurs if a transformation is applied that is too complex relative to the performance improvement it achieves. The problem of choosing a prior is equivalent to the problem of choosing a set of statistical tests. Adopting the AIT view is not auto-

matically beneficial. However, as is often the case, new approaches can be found by looking at a problem in a new light.

1.2.1 Conceptual Barriers to AIT View

The basic concepts of Algorithmic Information Theory and the theory of randomness have been known for quite some time. In spite of this, the AIT view is not widely used in machine learning. Two conceptual barriers have impeded progress in this area. The first barrier is simply that bits are very hard to use. This is because there is no one-to-one mapping between bits and real data samples. Due to the Shannon relation $L(x) = -\log_2 P(x)$, outcomes that are assigned high probability will use a small number of bits, while low probability outcomes will require many bits.

The second conceptual barrier is a limited understanding of what a statistical test is, and to what types of data it can legitimately refer. In the "typical" understanding of data compression and the AIT view, the original data is forgotten immediately after it is transformed into a bit string. Statistical tests refer only to previous bits of the string, not to the original data set. Also, statistical tests are not allowed to refer to an external information source. Thus, the typical understanding allows only the first test in the following list:

1. **Test** (any bit string): Count the frequency of 1s in the encoded data after the prefix 0110.

2. **Test** (image compression): Count the frequency of 1s in the encoded data corresponding to pixels where the previous pixel has a value less than 50.

3. **Test** (pattern classification): Count the frequency of 1s in the encoded data corresponding to labels X^k where the associated pattern C^k satisfies $W(C^k) = 1$, where $W(\cdot)$ is some binary context function.

Allowing statistical tests to refer to the real data or to external data sources is quite important for practical reasons. It is much easier to define the statistical tests in terms of the original data (e.g.: the previous pixel is less than 50) than in terms of encoded data (e.g.: the previous bits were 0101). Furthermore, one of the core problems of machine learning is to predict a set of labels X from a set of contexts or patterns C.

In this case, tests that refer to previous bits in the encoded version of the X data are of very limited value. It is much better to use tests that refer to information contained in the contexts C.

1.2.2 Probability Integral Transform Value Analysis (PITA)

The research in this thesis is built on two insights that overcome the conceptual barriers mentioned above. These insights allow us to use the AIT view illustrated in the Key Figure 1.1 for practical purposes.

The first insight, elaborated in Section 2.1.1, is that the encoded data of the Key Figure 1.1 does not need to be represented as a bit string for the basic process to work. Instead, we use a technique called the Probability Integral Transform to transform data outcomes into values distributed on the $[0, 1)$ interval [Angus (1994)]. Technically these numbers are called "Probability Integral Transform Values", but we will use the shorthand "PIT values". The key advantage of using PIT values for the encoded data is that there is always a one-to-one relationship between PIT values and data outcomes.

The second idea is just to expand the notion of what kinds of statistical tests can be used in the modeling process. In the algorithm developed below, the statistical tests are allowed to refer to the original (non-encoded) data. Tests can also refer to external data sources, specifically the patterns used in the pattern classification problem.

By using these two insights, in Chapter 3 we develop an abstract algorithm called Probability Integral Transform value Analysis (PITA), which is a practical version of the process illustrated in the Key Figure 1.1. This algorithm uses a battery of statistical tests $\mathcal{W}$ to search for randomness deficiencies in a sequence of PIT values representing some set of data outcomes X. Each data outcome is associated with a context. The statistical tests (also called "context functions" or "features") are based on information contained in the contexts C. If a test reveals a randomness deficiency in the PIT value sequence, it is used to update the models and reencode the PIT values.

While the core of the thesis involves the PITA algorithm and its applications, it should be emphasized that the AIT view of modeling is quite abstract. The PITA algorithm is only one possible technique that can be developed on the basis of the AIT view. There are many different ways to detect randomness deficiencies, and many different ways to transform encoded data to exploit randomness deficiencies.

1.3 Motivating Examples

The basic purpose of the PITA algorithm is to combine predictive information drawn from different sources (context functions and features) into one aggregate probability distribution. This section presents two examples where naïve approaches run into difficulty and PITA is effective.

1.3.1 Example #1: Prediction of Election Results

Consider the problem of the predicting the results of an election. The outcome of an election depends on hundreds of factors, including the state of the economy, the political platforms advocated by the different parties, and the personal characteristics of the candidates. Assume a historical data set exists which records the outcome of the election (X), as well as various information about the political and economic conditions (C) at the time of the election. In particular, the following facts (or features) are recorded:

- $W_1(C)$: over the last year, economic growth has exceeded the historical average.
- $W_2(C)$: over the last year, unemployment has been lower than the historical average.
- $W_3(C)$: over the last year, the ruling party has been involved in a scandal.
- X: true if the ruling party remained in power after the election.

The goal of the analysis is to predict the variable X based on the information contained in the context functions $W_i(C)$. Since the features can be calculated before an election, a good model $Q(x|c)$ can be used to predict the outcome of an election.

1.3.2 Example #2: Pixel Modeling

As a second example, consider the following problem which relates to modeling the distribution of pixels in an image. The target image is traversed row-by-row, pixel-by-pixel. There is a set of context functions which operate on the pixel history. Combining the result of these context functions with the actual pixel outcomes produces a dataset

that has the same basic form as the one described above. The data set might contain samples that include the following information:

- $W_1(C)$: true if the current color channel is "red".
- $W_2(C)$: true if the value of the previous pixel was < 50.
- X: actual pixel value.

Note that in this case the entire database comes from a single image. An image of size $640 \cdot 480 \cdot 3$ is a dataset with 921,600 outcomes. The four histograms in Figure 1.2 show the distribution of pixels, conditional on the binary context functions W_1 and W_2.

Given this information, the goal is to build a model $Q(x|c)$ of the probability of a pixel value given the pixel history. This problem is essentially equivalent to the problem of losslessly compressing the image. The image can be compressed if a good model $Q(x|c) \approx P(x|c)$ can be found.

1.3.3 Feature Combination

In both of the example problems mentioned above, the goal is to obtain a good approximation $Q(x|c)$ of the "real" distribution $P(x|c)$, where x is either the election result or the pixel outcome, and c is the context.

The simplest, and often the most effective, methods in statistical modeling involve plain counting. If there were no context information C, the best model is obtained simply by counting the X outcomes of each type. If the historical data set records 145 elections in which the ruling party won, and 87 elections in which the opposition won, then we approximate $Q(X = T) = 145/(145 + 87) \approx 63\%$.

Counting can work even with context information, by splitting the data set into subsets based on the context. Say the context outcome takes only two values, $C = C_a$ and $C = C_b$. Then $Q(x|c)$ can be estimated by using the same counting trick on the subsets C_a and C_b. Unfortunately, simple counting tricks cannot be used when the number of context outcomes is large. In this case, the context-specific subsets will be too small for the counts to be reliable. Assume the context in the pixel modeling problem is limited to include only the previous 5 pixels of the same color channel. Even with this strict restriction, the number of possible context outcomes is $256^5 \approx 10^{12}$.

Unless the data set is truly vast, most of the subsets defined by these contexts will be empty.

The problem can be simplified by introducing a set of context functions $W_i(c)$ and building models of the form $Q(x|W_1(c), W_2(c)\ldots)$ If the context functions are binary, then using $W_i(c)$ collapses the large context space into a single bit. Because of this collapsing effect, when using a small number of context functions, the data subsets will be reasonably large, so the counts will be reliable. However, if even a moderate number of features are used, the same problem comes up again. For example, if only three features are used the number of combinations is $2^3 = 8$. But if 100 features are used, then the number of combinations is 2^{100}.

This combinatorial explosion can be avoided by conditioning on only one W_i feature at a time. Doing so will produce statistics for the data subsets corresponding to $W_i = 1$ and $W_i = 0$. These subsets should be large enough to make the counts reliable. But now there is a new problem, which is how to combine the individual W_i statistics into one aggregate distribution $Q(x|W_1(c), W_2(c)\ldots)$.

The most obvious answer to this question is a method is called naïve Bayes. Given a feature outcome $W_i = 1$, naïve Bayes updates a model probability $Q(x)$ as follows:

$$Q'(x) := \frac{1}{Z} N(x|W_i = 1)Q(x) \tag{1.4}$$

Where $N(x|W_i = 1)$ are empirical frequencies, and Z is a normalization factor. This update is just a modified form of Bayes' rule.

The problem with naïve Bayes is that it assumes the features W_i are *independent*. To see the problem with this assumption, imagine that in the election prediction problem it is found that $N(X = 1|W_1 = 1) = .9$ and $N(X = 1|W_2 = 1) = .9$ (recall the W_1 refers to economic growth, while W_2 refers to unemployment). Now say that the features for the new election are $W_1 = W_2 = 1$. Then the naïve Bayes scheme will predict a 98.78% chance of victory for the incumbent, or 81:1 odds. But that estimate is too high; because W_1 and W_2 are not independent. Knowing *either* $W_1 = 1$ or $W_2 = 1$ may raise the odds from 1:1 to 9:1 in favor of the ruling party, but knowing *both* should not increase the odds by another factor of 9. This is because the two features both relate to the economy, so they contain a lot of overlapping information.

A similar problem arises in the pixel modeling problem. Given the information in either context function W_1 or W_2, it is easy to obtain a conditional model $Q(x|W_1(c))$

or $Q(x|W_2(c))$. These distributions would be exactly equal to the histograms shown in Figure 1.2. But it is quite difficult to combine the information to obtain a model $Q(x|W_1(c), W_2(c))$.

The PITA algorithm solves this problem as follows. The algorithm proceeds iteratively, adding a new context function W_i in each round. The trick is to take counts relating to the PIT values (encoded data) instead of the pixels (original data). Note that the PIT values depend on the current state of the model, while the pixels do not. The PIT value statistics obtained using the model $Q(x|W_1(c))$ contain the exact information necessary to update the model.

The PIT value distributions for the pixel modeling problem are shown in Figure 1.3. Notice how the top histograms are identical to Figure 1.2, but the bottom histograms are different. This is because the PIT values are recomputed when the update corresponding to W_1 is applied. The bottom histograms show the W_2-conditional distribution of PIT values obtained using the updated model $Q(x|W_1(c))$.

1.3.4 Feature Selection

Feature selection is an important and related problem, which arises when there is a large number of features W_i (say, millions). A model that uses all of these features will not only waste huge amounts of computational power but also almost certainly overfit the data. It is much better to create a model using only the best features. To do this requires a method for finding the best features.

A basic strategy would be to score all the features in terms of predictive power, and then use only the top features determined by this ranking. The predictive power of a feature W_i could be measured by the correlation or the mutual information between W_i and the outcome variable X. This strategy does not work well, for the same basic reason as above: in the real world, features are not independent.

Consider the election prediction problem again, and assume that the predictive power (S) of the three features has been found to satisfy: $S(W_1) > S(W_2) > S(W_3)$. Based on this ranking the naïve feature selection algorithm would first add W_1, then W_2, and then W_3. But this is not correct, again because W_1 (economic growth) and W_2 (unemployment) are highly related. Once W_1 is added, the *additional* predictive

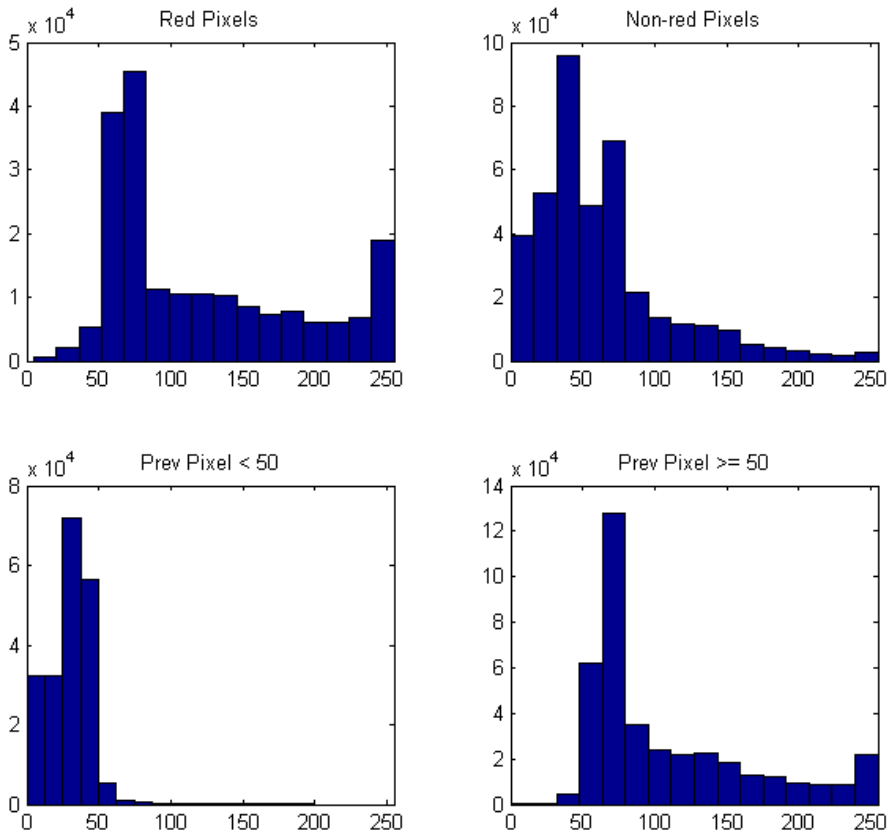


Figure 1.2: Basic image statistics. The top plots show the distribution of red and non-red pixel values. The bottom plots show the distribution of pixel values when the previous pixel is less than 50 or greater than 50.

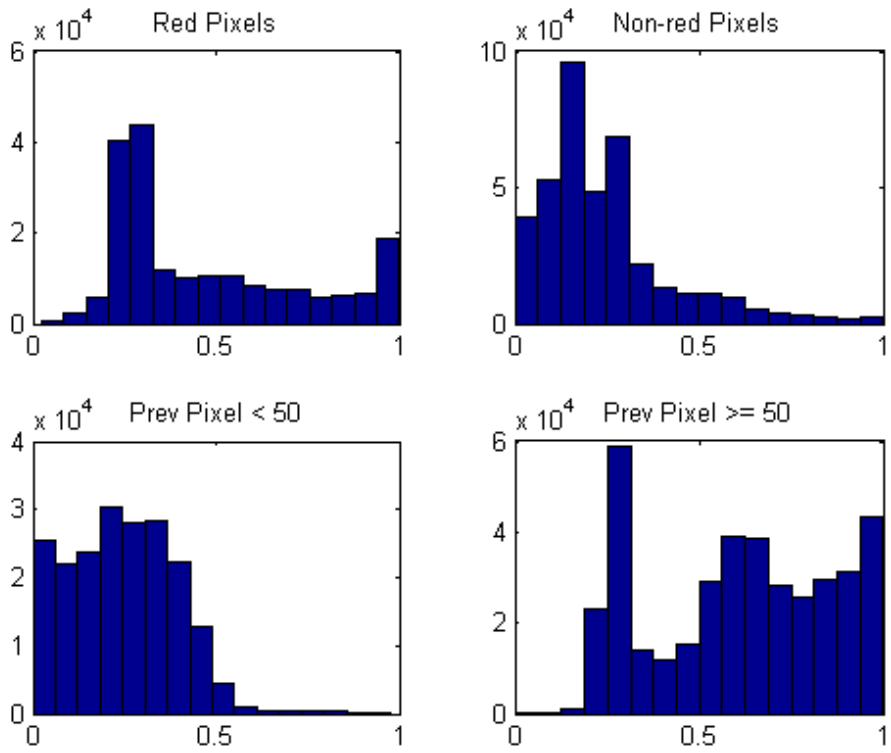


Figure 1.3: The top histograms show distributions of PIT values, generated using a uniform model $Q(x)$, corresponding to red vs. non-red pixels (W_1). The bottom histograms show distributions of PIT values generated using the model $Q(x|W_1(c))$, corresponding to positions where the previous pixel was less than or greater than 50. The top histograms are basically identical to those in Figure 1.2. The bottom histograms are different, because the PIT values come from the updated model.

power produced by W_2 is quite small. Therefore, after W_1 is added, it is better to add W_3, because it relates to to a different type of information (scandal).

Smart feature selection requires a way of measuring the improvement in predictive power achieved by adding a feature to the model. Obvious feature scoring methods, such as mutual information and correlation, refer only to the feature and the data: $S = S(W_i, \mathcal{X})$. Such scores do not take into account the current state of the model and are therefore insufficient. A smart score must be a function $S = S(W_i, Q_{@}, \mathcal{X})$, where $Q_{@}$ is the current state of the model.

In the PITA framework, the feature score is completely natural. It is simply the mutual information between a test W_i and the encoded data (PIT values) Φ. Since the encoded data is a function of both the real data and the model, $\Phi = \Phi(Q_{@}, \mathcal{X})$, the mutual information $I(W_i, \Phi)$ is of the form $S = S(W_i, Q_{@}, \mathcal{X})$ as required. The mutual information between a test W_i and the encoded data Φ can also be interpreted as a measure of the magnitude of the randomness deficiency in Φ revealed by W_i.

1.4 A New Mindset for New Applications of Statistics

A basic philosophical postulate of this thesis is the existence of data sets with "rich" or complex structure. The definition of a rich data set is one that cannot be described well using a simple model. An interesting two-dimensional taxonomy of data sets can be built using the notion of the richness of a data set, along with its size. Some examples of the different categories are given in the following list:

1. Small, plain structure: a dataset giving the heights of 100 people.
2. Large, plain structure: a dataset giving the heights of 10^8 people.
3. Small, rich structure: data from a clinical trial to test the efficacy of a new drug; data from a psychology experiment with $N = 257$ participants.
4. Large, rich structure: large text corpuses, image databases, databases of speech recordings, motion capture databases.

Historically, the main applications of statistics have been to fields such as psychology, genetics, biology, medicine, and economics. Two important characteristics of

these fields have influenced statistical thinking. First, it is often the case that real world data is actually distributed according to some standard, well-known distribution such as the Gaussian or the Laplacian. Second, the amount of data available is generally quite limited; in medical statistics each data sample comes from a person whose life is potentially at risk. Thus, the datasets generated in these fields are usually of case #1, case #2, or case #3 in the above list. For plain data, there is nothing to be gained by using a complex model. In case #2 it is possible to justify the use of a complex model, but there is no point in doing so, since the data can be described well using a simple Gaussian distribution. Much research in medicine, psychology, and related fields involves data sets of type #3. For example, the physiological response of the human body to a drug is certainly a very complex process. However, the data sets available in medical statistics are almost always very small. Because of this, simple models must be used to avoid overfitting. Classical statistics, which relies almost exclusively on the use of simple models, works fairly well on these types of problems.

In recent decades, researchers have attempted to apply statistical methods to fields such as computer vision [Huang & Mumford (1999); Nakayama *et al.* (2009)], speech understanding [Gazor & Zhang (2003b); Sohn *et al.* (1999)], motion understanding [Inamura *et al.* (2004)], and natural language processing [Rosenfeld (1996)]. These new applications deal with data sets that correspond to type #4 in the above list. Large quantities of data can be obtained, and there are good reasons to believe that complex models can provide better descriptions of the data than simple models can.

A central philosophical goal of this research is use of large data bases to construct complex models [Burfoot & Kuniyoshi (2009); Burfoot *et al.* (2008)]. This goal is based on the belief that some real-world phenomena cannot be described well using simple models. This is an abstract empirical statement about the nature of reality. The thesis does not attempt to prove this belief; doing so may be impossible. The statement is simply a hypothesis which motivates the research.

Note that the goal is not to construct complex models for their own sake. A simple model is better than a complex one if both achieve the same level of empirical performance. An analogy to the world of business may clarify the point. The cost of labor is a major expense for any business. If the number of employees can be reduced without harming the performance of the business, then from a purely capitalist perspective, this should be done in order to increase profits. However, it is obvious that some very large

companies are also very profitable. The reason is that for certain business tasks large numbers of employees are necessary.

The "complex world" hypothesis cannot be proved or disproved. However, the following comments by Vladimir Vapnik may lend it some plausibility:

> I believe that something drastic has happened in computer science and machine learning. Until recently, philosophy was based on the very simple idea that the world is simple. In machine learning, for the first time, we have examples where the world is not simple. For example, when we solve the "forest" problem (which is a low-dimensional problem) and use data of size 15,000 we get 85%-87% accuracy. However, when we use 500,000 training examples we achieve 98% of correct answers. This means that a good decision rule is not a simple one, it cannot be described by a very few parameters. This is actually a crucial point in approach to empirical inference [Vapnik & Gilad-Bachrach (2008)].

1.5 Organization and Contributions

The theoretical core of the thesis is the PITA algorithm, presented in Chapters 3. This algorithm is developed in two steps.

The first step, contained in Chapter 2, involves a special kind of model update based on a PIT value histogram. Section 2.1 introduces the Probability Integral Transform, and Section 2.1.1 explains the relationship to the method of inverse transform sampling. Sections 2.3 and 2.4 show how the information contained in a histogram of PIT values can be used to define a simple model update. An expression for the total codelength savings achieved by the update is obtained.

The development of the PITA algorithm is completed in Chapter 3. Section 3.2 points out the key property of the PIT histogram model update: it can be used to update an *ensemble* of models. Section 3.3 introduces the idea of context functions, which are used to partition the data into subsets. The ensemble of models corresponding to a given subset can then be updated using the appropriate PIT histogram transformation. Section 3.4 returns to the twin problems of feature selection and feature combination, and shows how the PIT histograms can be used to solve these problems. Section 3.5

discusses the computational costs and memory requirements of the algorithm. The PITA algorithm is especially well-suited for data sets that are too large to be held in memory. Section 3.6 demonstrates that the savings achieved by the histogram update is equal to a special kind of mutual information between the context function and the encoded data. Section 3.8 shows how the PITA model updates can be *layered* over some other type of model, producing a hybrid model with improved performance. The chapter also discusses several other technical ideas.

A relatively short Chapter 4 discusses alternative model ensemble updates based on statistics related to PIT values. The alternative updates are based on single-parameter transformation functions. The key is to find functional forms where the optimal parameter can be calculated from some easily computed PIT value statistic.

Chapter 5 discusses related work, in particular Maximum Entropy, boosting, goodness of fit tests, and other applications of the PIT idea in statistics and machine learning. Detailed comparisons are made between this previous research and PITA. An analysis of the situations in which PITA has a strong advantage over previous methods is also given. The central difference between PITA and other machine learning algorithms is that PITA uses the AIT view of modeling, in which the encoded data plays a central role.

The remainder of the thesis shows how the PITA algorithm can be applied to the following tasks:

- Chapter 6: Image compression.
- Chapter 7: Binary pattern classification.
- Chapter 8: Word morphology modeling.
- Chapter 9: Speech modeling.
- Chapter 10: Modeling and recognition of motion capture data.

Chapter 2

Model Ensemble Update based on PIT Histogram

This thesis proposes to adopt the AIT view of the modeling problem as illustrated in the Key Figure 1.1. However, instead of using bit strings, the encoded data will be represented as PIT values. These are obtained by applying the Probability Integral Transform to the data set $\mathcal{X}$ using the current model $Q_{@}$. This chapter introduces the Probability Integral Transform and proposes a method for updating a model ensemble using information contained in a histogram of PIT values. This method is then used as part of an algorithm for constructing complex context-conditional models $Q(x|c)$. This algorithm is called PITA and is discussed in Chapter 3.

2.1 Probability Integral Transform

The Probability Integral Transform is based on the following simple idea. Let $P(x)$ be the cumulative distribution function (CDF) for some continuous random variable X: $P(x) = Pr(X < x)$. Now define the random variable $\phi = P(X)$. Then ϕ is distributed uniformly on the $[0, 1]$ interval [Angus (1994)].

In statistical modeling, the real distribution $P(x)$ is not available. Instead there is an empirical data set $\mathcal{X} = \{X^1, X^2 \ldots X^N\}$. The goal is to obtain a model $Q(x) \approx P(x)$ from analyzing the data set $\mathcal{X}$.

An important question in statistics is how to measure the quality of the approximation $Q \approx P$. Techniques for doing this are called Goodness of Fit tests in the statistics

literature (see Chapter 5 for a discussion). The Probability Integral Transform provides another trick for measuring the quality of the model Q. The model Q is used to obtain PIT values $\phi^k = Q(X^k)$. If $Q \approx P$, then the PIT values should be nearly uniformly distributed on the unit interval. An uneven PIT value distribution implies that Q is a bad approximation of P.

Notice the similarity between the idea of a uniform distribution of PIT values and a random bit string. Chapter 1 mentioned that a bit string can be compressed if and only if it contains a randomness deficiency. Compressing the bit string is equivalent to improving the model. The same basic idea will hold with a set of PIT values. If a randomness deficiency is found in the PIT values, it can be used to improve the model. A key point is that the number of PIT values is always equal to N, the number of data samples. This means that a one-to-one mapping between PIT values and original data outcomes can always be maintained, which is not the case for bits. Instead of reducing the number of PIT values, the modeling process expands the region of the $[0, 1]$ interval that the PIT values are allowed to occupy.

2.1.1 Inverse Transform Sampling

Another way to understand the PIT concept is through the connection to the idea of sampling. To sample from a model means to produce a fictional data set that is "typical" of the model. This can be useful for various reasons. For example, the descriptive adequacy of a model can be qualitatively assessed by sampling from it and comparing the sampled data to the real data.

The following two step process, called inverse transform sampling, can be used to sample from a model [Devroye (1986)]. First a random number generator is invoked to produce $[0, 1]$ uniform random variables. Then the model's inverse cumulative distribution function (CDF) is used to map these numbers to the space of data outcomes.

Consider the following example of sampling from a Gaussian model with parameters $\mu = 50$ and $\sigma = 10$. Say that the first number produced by the random number generator is $\phi = .25$. This ϕ-value implies a sampled data value of about 43.26, because:

$$\int_{-\infty}^{43.26} N(50, 10)dx = .25 \tag{2.1}$$

The Probability Integral Transform can be thought of as the *inverse* of this sampling process. Given a real data set $\{X^k\}$ and a model Q, the PIT values $\{\phi^k\}$ are a set of $[0, 1]$-distributed numbers that, if emitted by the random number generator, would produce X^k when sampling from Q:

$$\phi^k = \{\phi : Q^{-1}(\phi) = X^k\} \tag{2.2}$$

Notice how the PIT value ϕ^k depends on both the model Q and the real data outcome X^k. An issue arises if the data is not continuous. In this case the inverse function Q^{-1} is not well-defined: many ϕ values map to the same outcome X. The simplest way to solve the problem is to select ϕ at random on the valid range. Often, the precise value of ϕ will not matter, as long it is within the proper range. In other cases, this becomes an issue called the Bin Overlap problem, discussed in Section 3.11.

2.1.2 Analyzing the PIT values

An analysis of the PIT value distribution can help determine if a model is good, and provide a way of improve the model if it is imperfect. Suppose a set of data $\mathcal{X}$ is generated by inverse transform sampling. The random number generator produces a set of uniform $[0, 1]$ values, and these values are mapped to the data space using a distribution $P(x)$. Since $\mathcal{X}$ is generated by sampling from $P(x)$, $P(x)$ is a perfect model for $\mathcal{X}$. When the Probability Integral Transform $\phi^k = P(X^k)$ is applied, the resulting PIT values will be exactly the original set of uniform $[0, 1]$ random numbers given to us by the random number generator. A histogram of ϕ^k values will be nearly flat.

Now imagine that $\mathcal{X}$ is generated by using some distribution $P(x)$, but the ϕ^k are calculated using some other (model) distribution $Q(x)$. Assuming the distributions are significantly different, the resulting PIT histogram will be uneven, because the model $Q(x)$ has assigned a large probability width to outcomes that occur rarely according to $P(x)$, and vice versa. The PIT values will be clumped together or skewed to one side of the unit interval. This skewed PIT histogram indicates a randomness deficiency, just like an overabundance of 1's in a bit string. For this reason, the PIT values can play the same role as the bit strings S_i in the Key Figure 1.1. Furthermore, as discussed below, the PIT histogram contains exactly the information necessary to update the model.

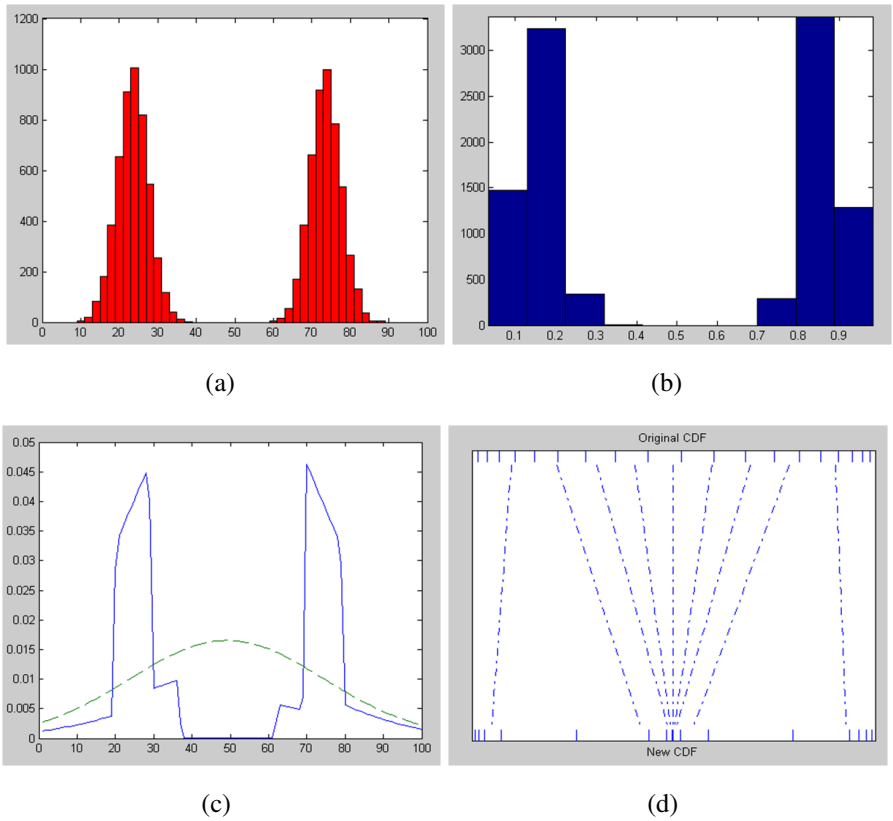


Figure 2.1: Illustration of PIT histogram based reencoding. The original data is shown in (a). This data is modeled using a Gaussian with the optimal mean and variance, shown as the green dotted line in (c). Applying the Probability Integral Transform $\phi^k = Q(X^k)$, where $Q(\cdot)$ is the Gaussian model, yields the PIT histogram (b). This histogram corresponds to the transformation shown in (d). Using the transformation to update the original model yields the blue curve in (c), clearly improving the fit.

This idea is illustrated in Figure 2.1. Here we start with a data set that comes from a Gaussian mixture model with two components (Figure 2.1(a)). Our first attempt at modeling the data was a single Gaussian, shown as the green dotted line in Figure 2.1(c). This model is obviously quite imperfect, and this imperfection is illustrated by the highly uneven PIT histogram (Figure 2.1(b)) obtained using the model. The PIT histogram CDF transformation, to be discussed below, is shown in Figure 2.1(d). The effect of this transformation is to expand CDF regions corresponding to histogram bins with many hits while contracting regions with a small number of hits.

2.2 Notation

Before proceeding further, it will be useful to define the notation used in this thesis.

Real world distributions are denoted P, while models are denoted Q. The goal of modeling is to find a good approximation such that $Q \approx P$. Of particular interest are complex conditional distributions $Q(x|c) \approx P(x|c)$, where c is a context variable that can take many possible values.

There is a subtle difference between the notation x and X. The lower case x denotes an argument to a function such as $P(x)$ or $Q(x)$, while X refers to the actual outcome of a random variable.

The data set $\mathcal{X}$ is a set of N data samples $\{X^1, \ldots X^k \ldots X^N\}$. A context value $\{C^1, \ldots C^k \ldots C^N\}$ is associated with each data sample. The concrete definition of the data samples X^k and contexts C^k depends on the application. Basically, the X^k are whatever the user is interested in predicting and the C^k are whatever is used to do the prediction. Some examples are given below. The index k always refers to data set indices.

The PITA algorithm developed in this thesis operates on an *ensemble* of models. Each data outcome has its own model attached to it. The model associated with the kth data sample is $u^k(x)$. This notation can be used to write the total codelength required to encode the data set $\mathcal{X}$ using the model ensemble is:

$$L = -\sum_{k=1}^{N} \log_2 u^k(X^k)$$

The individual models are related to the full complex conditional model by the expression $u^k(x) = Q(x|C^k)$. The full conditional distribution $Q(x|c)$ is never explicitly represented; doing so would be impossible because the space of contexts is assumed to be vast. Instead, the individual $\{u^k\}$ models for each data point X^k are built and updated. A set of models $\{u^k\}$ is called a model ensemble. Note that a single u^k is just an array of numbers.

In addition to models and data, it will also be important to refer to histograms, histogram bin widths, and histogram bin counts. The bin widths of an M-bin histogram are denoted $\{e_1, \ldots e_j \ldots e_M\}$. The normalized bin heights, which are the number of samples that fall into a given range divided by the total number of samples, are

denoted $\{e'_1, \ldots e'_j \ldots e'_M\}$. Note that the bin widths and the normalized bin heights are conceptually similar to probability distributions. The index j refers to histogram bins.

A central concept in this thesis is the relationship between probability distribution functions (PDFs) and cumulative distribution functions (CDFs). The notation μ^k refers to the CDF corresponding to u^k, while ϵ and ϵ' correspond to e and e'. By convention, a CDF for an n-outcome variable has $n + 1$ elements; the first element is 0 and the last element is 1. By that convention, $u(x_i) = u_i = \mu_i - \mu_{i-1}$, where $1 \leq i \leq n$. Also, it will sometimes be convenient to refer to the upper and lower CDF boundaries for a given data outcome. Thus $\mu_+(x_i) = \mu_i$ and $\mu_-(x_i) = \mu_{i-1}$. The region $[\mu_-(x_i), \mu_+(x_i)] \in [0, 1]$ is often called the "CDF outcome window" or "CDF window" corresponding to an outcome x_i. Using this notation the empirical codelength can be written as:

$$L = -\sum_{k=1}^{N} \log_2(\mu_+^k(X^k) - \mu_-^k(X^k))$$

As discussed below, the PIT values ϕ^k are chosen such that:

$$\mu_-^k(X^k) < \phi^k < \mu_+^k(X^k)$$

Note the dependence of the PIT value ϕ^k on both the real data outcome X^k and the kth model μ^k.

Another notational convention is to use calligraphic letters to denote sets. Thus $\mathcal{X}$ refers to the data set, $\mathcal{C}$ refers to the set of contexts, and $\mathcal{U}$ refers to the set of models. The battery of context functions used to search for randomness deficiencies is denoted $\mathcal{W}$.

2.2.1 Example X^k and C^k Definitions

To give a more concrete understanding of the meaning of the notation X^k and C^k, some examples of what these variables mean in various applications are now provided.

- **Image Compression**. The X^k is a single pixel value, and C^k is the history of previously sent pixels.

- **Pattern Recognition**. The X^k is the label or category of the pattern, and C^k is the pattern itself. For example in the face detection problem, X^k is a binary label that indicates "face" or "non-face", while C^k is the associated image.

- **Time Series Modeling**. The X^k is each new observation, and C^k is the history of previous observations. This formulation is used for the speech modeling (Chapter 9) and motion recognition (Chapter 10) applications discussed below.

In the work described in this thesis, the X^k data is assumed to be single dimensional. While this is not strictly necessary from a theoretical standpoint, for technical reasons related to the construction of CDFs, it is much easier to handle single dimensional data. This restriction is not particularly problematic, because the chain rule of probability can be used to rewrite joint distributions as conditional distributions. For example, a five dimensional joint distribution $p(x_1, x_2, x_3, x_4, x_5)$ can be constructed from a series of conditional distributions:

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)p(x_4|x_3, x_2, x_1)p(x_5|x_4, x_3, x_2, x_1)$$

The PITA algorithm can then be used to model the conditional distributions. In the time series modeling applications of Chapters 9 and 10, each new data observation arrives in the form of a D-dimensional vector. In those chapters, the solution is to build D separate models, one for each dimension.

It is worth noting that there are two ways of looking at the problem of modeling an image. Consider a single image with $D = 400,000$ pixels. One way of looking at the image is as a single data sample in a D-dimensional space. Alternatively, the image can be thought of as a data set with D samples $\{X, C\}$, where the context corresponding to each outcome is the history of previous pixels. This thesis uses the second view.

2.3 Histogram CDF Remapping Transformation

The transformation illustrated in Figure 2.1(d) can be formalized as follows. Let $\{\mu_0 = 0, \mu_1, \mu_2, \ldots \mu_n = 1\}$ be a CDF for a discrete random variable with n outcomes. Given a set of histogram bin widths $\{e_j\}$ corresponding to a CDF $\{\epsilon_0, \epsilon_1, \epsilon_2, \ldots \epsilon_M\}$ and

normalized bin heights $\{e'_j\}$ corresponding to a CDF $\{\epsilon'_0, \epsilon'_1, \epsilon'_2, \ldots \epsilon'_M\}$, the histogram based CDF remapping transformation is given by:

$$j(i) = \{j : \epsilon_{j-1} \leq \mu_i < \epsilon_j\}$$
$$\mu_i = \epsilon_{j(i)-1} + \beta(\epsilon_{j(i)} - \epsilon_{j(i)-1}) \tag{2.3}$$
$$\mu'_i = \epsilon'_{j(i)-1} + \beta(\epsilon'_{j(i)} - \epsilon'_{j(i)-1}) \tag{2.4}$$

Thus Equation 2.3 determines the value of $\beta \in [0, 1]$, and this value is then used to calculate μ'_i. This transformation is illustrated in Figure 2.2. Note that if the CDF window of a data outcome x is contained entirely within one of the histogram bins so that $\epsilon_{j-1} \leq \mu_-(x) < \mu_+(x) \leq \epsilon_j$, then the total change in the probability of x can be expressed simply:

$$u'(x) = \frac{e'_j}{e_j} u(x) \tag{2.5}$$

In other words the probability is simply scaled by the same factor as the histogram bin containing the CDF outcome window. For an outcome window that spans a histogram bin boundary, the update will be more complex, but the resulting probability can still be bounded. Consider a CDF outcome that spans a single histogram bin partition:

$$\epsilon_{j-1} < \mu_-(x) < \epsilon_j < \mu_+(x) < \epsilon_{j+1}$$

Assume for simplicity that $e'_j/e_j < e'_{j+1}/e_{j+1}$. Then the updated probability will be within an interval:

$$\frac{e'_j}{e_j} \leq \frac{u'(x)}{u(x)} \leq \frac{e'_{j+1}}{e_{j+1}}$$

2.4 PIT Histogram Update

Consider a dataset $\mathcal{X} = \{X^1, X^2, \ldots X^k, \ldots X^N\}$. Assume that every data point has a model $\mathcal{U} = \{u^1, u^2, \ldots u^k, \ldots u^N\}$ associated with it. For every data sample, we select a PIT value ϕ^k such that:

$$\mu_-^k(X^k) \leq \phi^k < \mu_+^k(X^k) \tag{2.6}$$

For the time being we are not concerned with the specific value of ϕ^k, as long as it is within the required range. A set of histogram bin widths $\{e_j\}$ is chosen (almost always uniform), and the number of PIT values falling in each bin is counted. This results in a set of histogram counts N_j. The normalized bin heights are $\{e_j\} = N'_j/N$. The CDF remapping transformation of the previous section is then used to update the model ensemble $\{u^k\}$, using the bin widths $\{e_j\}$ as the source distribution and the bin heights $\{e'_j\}$ as the target distribution. A crucial assumption allows the codelength savings achieved by this model ensemble update to be predicted exactly.

Theorem. Consider a data set $\{X^k\}$ with associated models $\{u^k\}$, and a set of histogram bin widths $\{e_j\}$ corresponding to $\{\epsilon_j\}$. Assume that for all k, there is some $j(k)$ such that:

$$\epsilon_{j(k)-1} \leq \mu_-^k(X^k) < \mu_+^k(X^k) < \epsilon_{j(k)} \tag{2.7}$$

Then if the normalized bin heights are given by $\{e'_j\}$ (corresponding to $\{\epsilon'_j\}$), the average per-sample codelength savings resulting from updating all of the models u^k using the histogram update indicated by e, e' is the Kullback-Liebler divergence $D(e'||e)$.

Proof. Because of the assumption expressed by Equation 2.7, the simple update rule of Equation 2.5 can be used. The updated probabilities are given by:

$$u'^k(X^k) = u^k(X^k)\frac{e'_{j(k)}}{e_{j(k)}}$$

The before (L) and after (L') code lengths are given by:

$$\begin{aligned} L &= -\sum_{k=1}^{N} \log u^k(X^k) \\ L' &= -\sum_{k=1}^{N} \log u^k(X^k) - \sum_{k=1}^{N} \log \frac{e'_{j(k)}}{e_{j(k)}} \end{aligned}$$

Thus the total savings is given by:

$$L - L' = \sum_{k=1}^{N} \log \frac{e'_{j(k)}}{e_{j(k)}} \tag{2.8}$$

$$= \sum_{j} \log \frac{e'_j}{e_j} \sum_{k':j(k')=j} 1 \tag{2.9}$$

$$= N \sum_{j} e'_j \log \frac{e'_j}{e_j} \tag{2.10}$$

This is N times the Kullback-Liebler divergence between $\{e_j\}$ and $\{e'_j\}$. □

The notationally difficult step in this derivation is the transition from Equation 2.8 - 2.9. Since all outcomes that are assigned to the same bin get the same factor, we can rewrite the sum as first going over all bins (j-index) and then over all outcomes assigned to a given bin (k'-index). The step 2.9 - 2.10 is justified by the fact that the number of outcomes assigned to a bucket is exactly N times the normalized bucket height e'_j.

The CDF remapping update changes the model probability $u^k(x)$ of all the possible values that the kth outcome could take. However, the codelength depends on only the outcome X^k that actually occurred, so the derivation refers to values $u^k(X^k)$. The bin containment assumption (Equation 2.7) is only necessary for these outcomes. Note that the bin containment assumption means that as long as ϕ^k falls within the appropriate range, the exact method used to choose it does not affect the histogram counts, and is thus unimportant.

It is worthwhile to interpret this theorem in terms of the idea of randomness deficiencies. Clearly, if the histogram is flat then the savings will be nearly zero; if it is uneven, substantial savings can be achieved by applying the update. An uneven PIT histogram corresponds to a randomness deficiency, and the larger the randomness deficiency, the more codelength savings can be achieved by applying the update.

A key fact about the above process is that it makes no assumptions regarding the type of data or how the models $\{u^k\}$ were obtained. It may be that all these models are the same, but this is not required. In fact, it is not even required that the samples represent the same type of data or have the same number of potential values. It is acceptable for one third of the data pool to represent a set of height and weight measurements for

Central Asian populations, while another third represents the stock price movements on the New York Stock Exchange, and another third contains measurements relating to the number of cosmic rays entering the Earth's atmosphere on a given day. It may be difficult to find a randomness deficiency in such an odd dataset, but if one *is* found, then it can be used to update the entire model ensemble, and doing so will achieve the average savings indicated by the KLD histogram score.

The above result may seem at first glance to be an obvious corollary of the definition of the Kullback-Leibler divergence, which is the code length penalty paid for using a model Q to encode a data source distributed according to some other distribution P. What makes it different is the fact that here we are updating an *ensemble* of models. There is no single source model or source distribution: each data sample X^k is encoded with its own model u^k. The savings achieved is a result of simultaneously updating N distinct models, improving the overall fit to a set of N data points.

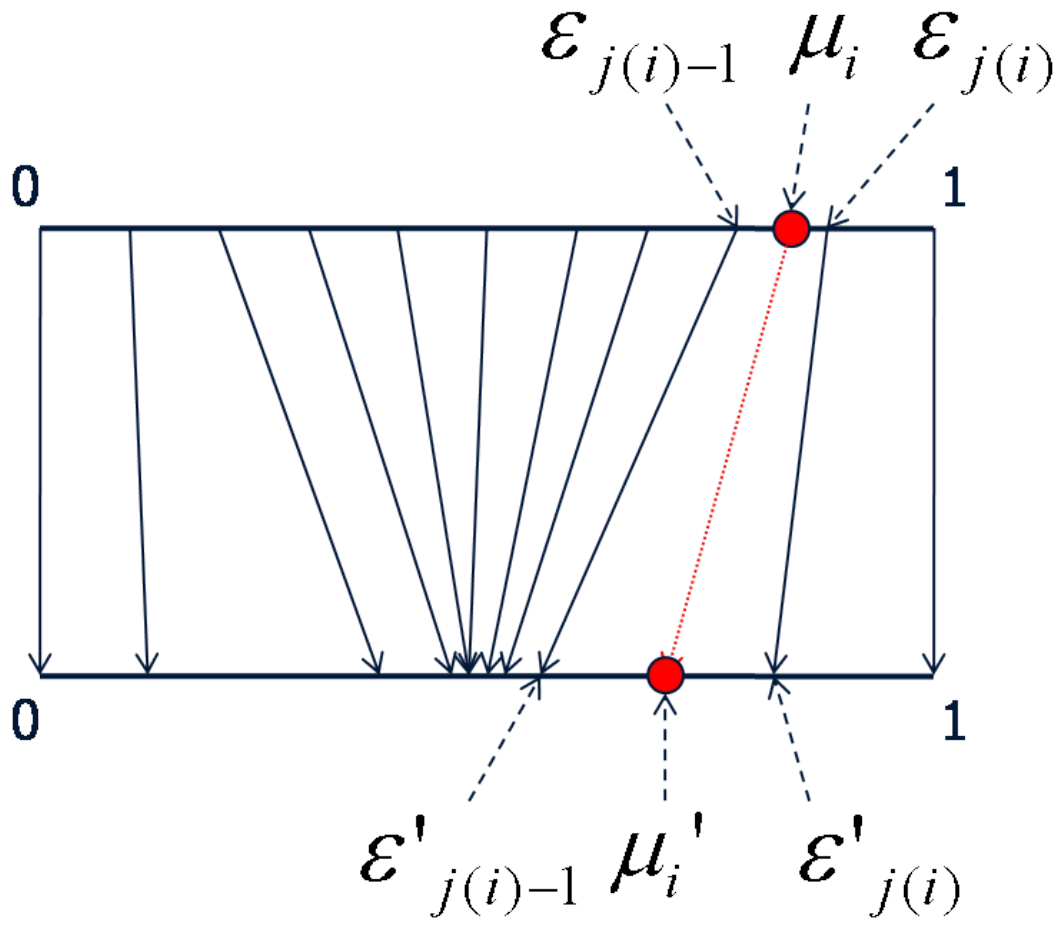


Figure 2.2: Illustration of the updating process applied to a single model CDF point μ_i. The ϵ and ϵ' points come from the information in the PIT histogram. Note that in general there can be many more μ points than ϵ points: 10,000 μ points can be remapped using 10 ϵ points.

Chapter 3

PIT Analysis Algorithm

This chapter presents the core idea of the thesis, the Probability Integral Transform value Analysis (PITA) algorithm. The algorithm uses the model update method of Chapter 2, in combination with a set of context functions (or features) to construct complex context-dependent models $Q(x|c)$. These kinds of models are quite general and can be used in many different applications. A pseudocode version of PITA is shown in Algorithm 1.

A rough explanation of the algorithm is as follows. The algorithm maintains an individual model distribution u^k for each data sample X^k. These distributions can be initialized as uniform, or they can be obtained from some other modeling process. At the beginning of the algorithm, PIT values $\{\phi^k\}$ are obtained from $\{u^k\}$and $\{X^k\}$ using the Probability Integral Transform. The algorithm then proceeds through a series of learning rounds. In each round, the battery of binary context functions $\mathcal{W}$ is applied to the stream of PIT values. Each context function splits the data into two subsets, from which two histograms H_0, H_1 are constructed. The context functions are then scored by taking the weighted sum of the two histogram KLD scores. The context function w^* with the highest score is used to update the models and PIT values, and then a new round begins. These points are explained in greater detail below.

3.1 Data Transmission Thought Experiment

One of the key points about the PITA algorithm, that distinguishes it from other techniques in machine learning, is that it deals with an *ensemble* of models $\{u^k\}$. The goal

Algorithm 1 Abstract PITA algorithm.

given data samples $\mathcal{X} = \{X^k\}$, associated contexts $\mathcal{C} = \{C^k\}$,
context functions $\mathcal{W}$.
initialize models $\mathcal{U} = \{u^k\}$
$\Phi = \{\phi^k\} = \mathcal{U} \to \text{ProbIntTrans}(\mathcal{X})$
for $t = 1 : T$ **do**
  for $w \in \mathcal{W}$ **do**
    $\Phi_0 = \{\phi^k : w(C^k) = 0\}$, $H_0 = \text{histogram}(\Phi_0)$
    $\Phi_1 = \{\phi^k : w(C^k) = 1\}$, $H_1 = \text{histogram}(\Phi_1)$
    score(w) = $|\Phi_0| \cdot \text{KLD}(H_0) + |\Phi_1| \cdot \text{KLD}(H_1)$
  end for
  let $\{w_t^*, H_{0,t}^*, H_{1,t}^*\}$ be the best context function, histogram
  for all k **do**
    if $w_t^*(C^k) = 1$ **then**
      $u^k := H_{1,t}^* \to$ cdf-remap(u^k) // Update shown in Figure 2.2
      $\phi^k := H_{1,t}^* \to$ cdf-remap(ϕ^k)
    else
      $u^k := H_{0,t}^* \to$ cdf-remap(u^k)
      $\phi^k := H_{0,t}^* \to$ cdf-remap(ϕ^k)
    end if
  end for
end for
output final distributions u^k, optimal tests w_t^*,
transformation parameters $H_{0,t}^*, H_{1,t}^*$

of PITA is to improve this model ensemble by applying a series of *simple* updates. It is worth reflecting on the meaning of the word "simple", which is of fundamental importance in statistics. The following thought experiment is an attempt to explain the PITA algorithm and the role of the word "simple" in it.

There is a data set $\{X^k\}$, which a sender wishes to transmit to a receiver. Both parties are assumed to start with an identical set of initial models $\{u^k\}$(in the simplest case, these are just uniform models). Because both the sender and the receiver have the same $\{u^k\}$, the sender can encode each sample X^k using the associated model u^k, and the re-

ceiver can decode it correctly. Initially, the cost of transmission is $\sum_k -\log_2 u^k(X^k)$.

Now, assume that the sender and receiver have agreed upon some protocol that allows them to update the models $\{u^k\}$. If the sender realizes that some update to the ensemble will produce a shorter codelength, then he encodes the update and sends it to the receiver. Then both sides perform identical model updates, ensuring that the models remain synchronized. The goal is to produce the shortest *net* codes, taking into account both the model update cost and the data encoding cost. Thus, the meaning of "simple" is that the update can be encoded with a short code. This idea is called the MDL Principle [Rissanen (1978)].

Consider what happens if we do not account for the model update costs in the data transmission problem. Then the sender can trivially reduce the codelength to zero, by updating individual models into delta functions. In other words, the sender simply sends a series of updates that turns each $u^k(x)$ into $\delta(x, X^k)$. These δ-function models then allow the data to be sent at zero cost.

This trivial scheme no longer works if the cost of the updates is accounted for, since explicit updates to specific models requires a large codelength. Instead, it is necessary to find more efficient updates. For example, the sender might notice that the outcomes X^k with even k indices tend to be very small. He can therefore apply a simple update to all the $\{u^k\}$with even k indices that shifts probability away from the large outcomes and towards the small outcomes. This update can be specified with a short code, but if the number of samples N is large, it can achieve a substantial net savings.

3.2 Model Ensemble Updates

A method of reasoning about model ensembles is necessary to implement the above data transmission scheme. This is not trivial, because it is not enough to make observations about the data $\{X^k\}$alone. For example, it is not useful to *merely* observe that the X^k corresponding to $k \in \{450, 574\}$ are all distributed within a narrow region, if the models u^k for $k \in \{450, 574\}$ already assign high probability to that region. Instead, it is necessary to find some way in which the models are systematically miscalibrated.

Furthermore, it is not immediately obvious how to implement the model ensemble update. One simple option would be to apply a multiplicative factor α to some range of values. For example, one could update $u'(x) = \alpha u(x)$ for $x \in \{0, 5\}$. The problem

with this is that the new distribution will have to be renormalized immediately after the update. The actual change in probability of an outcome $x \in \{0, 5\}$ will not be α, but some other value that depends on the probability assigned by the model to other values $x \notin \{0, 5\}$. This makes it very difficult to predict the actual change in net codelength.

Solving these problems is, of course, the whole reason why the PIT histogram transformation was introduced in Chapter 2. The PIT value ϕ^k is determined by X^k and μ^k as follows:

$$\mu^k_-(X^k) \leq \phi^k < \mu^k_+(X^k) \tag{3.1}$$

Because the PIT value depends on both the data point X^k and the model u^k, the PIT histogram allows us to visualize and quantify the extent to which the ensemble of models $\{u^k\}$ fits the data $\{X^k\}$. Furthermore, the histogram KLD score of Equation 2.10 can be used to predict the codelength savings that will be achieved by applying the update.

So, in the terms of the data transmission example, the first step is for the sender to calculate the PIT histogram for the entire data set. If this histogram is substantially uneven, then the sender sends the histogram to the receiver, and both parties update all the models u^k using the histogram CDF update.

Now, this updating process can be repeated a couple of times, but there is a limit to how much can be gained, for the following reason. When the PIT histogram transformation is used to update an ensemble of models, the corresponding PIT values must be updated as well. The new PIT values $\phi^{k'}$ must be chosen such that Equation 3.1 is satisfied, using the new models $\mu^{k'}$. In fact, the exact same method (Equation 2.4, Figure 2.2) for updating the model CDF points μ^k can also be used to update the PIT values ϕ^k.

Upon reflection, the following fact becomes obvious: the new PIT histograms corresponding to the updated $\phi^{k'}$ will be *more flat* than the previous PIT histograms. This is because the effect of the model update is to expand regions with many PIT outcomes and to contract regions with few outcomes. This flattening effect is exactly analogous to the randomization process in the AIT view of modeling (Key Figure 1.1).

Recall that the savings that can be achieved by the model update depends on the degree of unevenness of the PIT histogram. Thus, as the histograms become more

and more flat, the savings produced by the model ensemble update gets smaller and smaller. Context functions can be used to circumvent this problem.

3.3 Contexts and Context Functions

In Chapter 1 we noted that the frequency of 1s in a random bit string must be about 50%. This is not the only condition that a random bit string must satisfy. In fact, a truly random string must satisfy an enormous number of conditions:

- The frequency of 1s is about 50%.
- The frequency of 1s after the prefix "0010" (or any other prefix) is about 50%.
- The frequency of 1s in positions indexed by square numbers (1, 4, 9, 16, ...) is about 50%.
- ... and many other conditions.

A flat PIT histogram is analogous to a bit string in which the frequency of 1s is about 50%. Just as with a bit string, there are many other conditions that the PIT value sequence must satisfy in order to be truly random. For the sequence to be random, it is necessary for any *subset*[1] of the PIT outcomes to produce a flat histogram. Thus, using the pixel modeling example:

- The histogram of all PIT values must be nearly flat.
- The histogram of PIT values corresponding to pixels in the red color channel must be nearly flat.
- The histogram of PIT values corresponding to pixels where the previous pixel is < 50 must be nearly flat.
- ... and many other conditions.

[1] More precisely: any simply-definable subset.

The subsets can be defined using the notion of a context. The context C^k is a package of data that is "attached" to the data sample X^k. The concrete definition of the context depends on the application. In a time series modeling application, the context is just the history of previous observations. In a pattern recognition problem, the context is the pattern. Note that, in terms of the data transmission example, the receiver must already have the contexts C^k, or be able to construct the contexts from the previously sent X^k data (e.g., context C^k may just be the outcomes $X^{k-1}, X^{k-2}, \ldots$).

In general, the size of the context outcome space is far too large to be conditioned upon directly. Instead, a set of context functions or features $\mathcal{W}$ is used. Typically these will be simple binary functions $w(c) \in \{0, 1\}$, but it is easy to generalize to the case of multiple discrete outcomes. Another way of thinking of a binary context function is as a way of splitting the full data set into two subsets:

$$S_0 = \{k : w(C^k) = 0\}$$
$$S_1 = \{k : w(C^k) = 1\}$$

After applying the PIT histogram update to the full data set a few times, the *full* PIT distribution becomes flat. However, it may very well be that a *subset* of the PIT values will still have an uneven histogram. If such a subset can be found, then an additional codelength savings can be achieved by applying the CDF update transformation to it. This idea is illustrated in Figure 3.1.

Thus, the full PITA algorithm can be explained in terms of the data transmission example as follows. The models $\{u^k\}$are initialized according to some predefined protocol, and the PIT values $\{\phi^k\}$are calculated based on $\{u^k\}$and $\{X^k\}$. Then the sender searches the PIT value distribution for randomness deficiencies, by using a battery of context functions $\mathcal{W}$. In each round, the context function w_t^* that reveals the largest randomness deficiency, and thus can be used to achieve the largest codelength savings, is selected. The sender transmits the index of w_t^* in $\mathcal{W}$, as well as the histograms $H_{t,0}^*, H_{t,1}^*$ associated with the context function. The receiver now has all the necessary information to update the model ensemble, so both parties perform the identical update, ensuring that the models remain synchronized. Then another round begins.

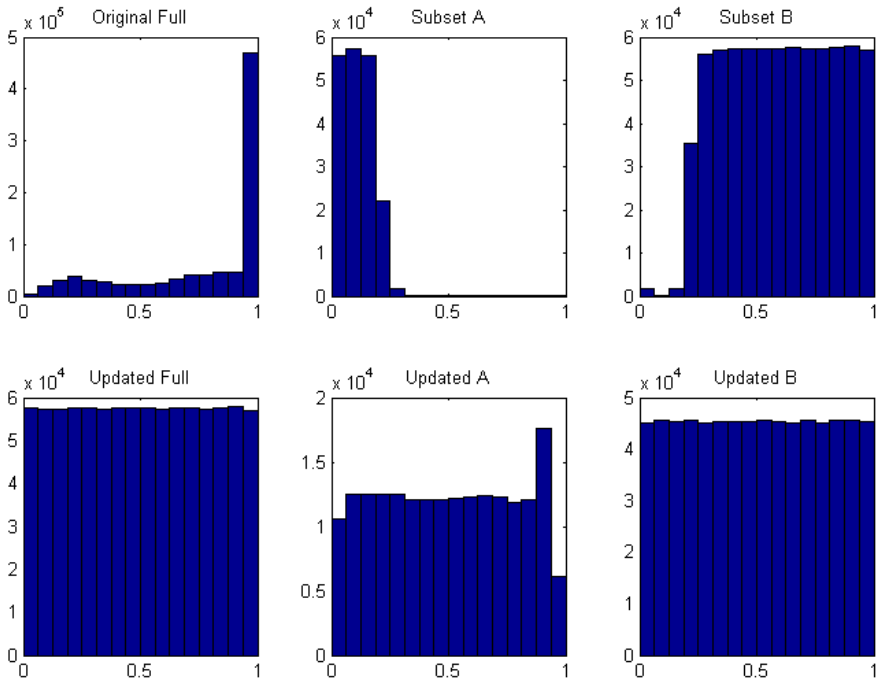


Figure 3.1: Illustration of repeated applications of the histogram update method. The histogram in the upper left shows the full subset of PIT values from a dataset corresponding to the pixels of an image. The lower left shows the new PIT distribution that is produced by updating the models using the histogram in the upper left. This distribution is then split into subsets A and B, correspoding to outcomes where the previous pixel is less than 50 (A) or greater than 50 (B). This split reveals a substantial randomness deficiency. Both subsets are then updated, producing new, more random PIT distributions as shown in the lower center and lower right.

3.4 Feature Selection and Combination

The discussions contained in Sections 1.3.3 and 1.3.4 emphasized that any feature-based modeling scheme must solve the related problems of feature combination and feature selection[1]. Given a set of statistics relating to feature W_1, and another set of statistics relating to feature W_2, how can these features be combined together to obtain an aggregate model $Q(x|W_1(c), W_2(c))$? And how can the best features be selected from a large package (say, $|\mathcal{W}| = 10^6$)?

The ability of the PIT histogram update to be used on an ensemble of models means that it can be used to solve the feature combination problem. In the first round, the first feature W_1 is used to split the data into subsets. Then the subsets are updated using the PIT histogram method. The model is now of the form $Q(x|W_1(c))$. In the next step, a new feature W_2 is used to split the data into two subsets, and those subsets are again updated. The model is now of the form $Q(x|W_1(c), W_2(c))$. Notice how the W_2 step does not "overwrite" the information already built into the model using W_1. This can be seen by observing that the model update is guaranteed to produce codelength savings, assuming the key assumption is met. This would not be true if the W_2 step "overwrote" the information from W_1.

To solve the feature selection problem, it is necessary to find a way to score features. The histogram KLD score provides a natural way of doing this. The score of a feature is simply the weighted sum of the histogram KLD scores for the subsets defined by the feature:

$$|S_0| \cdot KLD(S_0) + |S_1| \cdot KLD(S_1) \tag{3.2}$$

As noted in Chapter 1, a "smart" feature selection score must be a function $F = F(W, Q_@, \mathcal{X})$ where $Q_@$ is the current model. A score of the form $F = F(W, \mathcal{X})$ is insufficient, because this form cannot express how much improvement is achieved by using a feature W. The above scoring expression is of the form $F = F(W, \Phi)$. Since the PIT values depend on the model and the data, $\Phi = \Phi(Q_@, \mathcal{X})$, this score is of the correct form.

[1] Note: in this thesis the word "feature" means the same as "context function".

Chapter 1 mentioned an example problem relating to the prediction of an election. This problem involved three features W_1, W_2, W_3 relating to economic growth, unemployment, and political scandals. In this example W_1 and W_2 had stronger predictive power than W_3, but W_1 and W_2 were also very similar to one another (since economic growth and unemployment are strongly linked). Therefore, a smart feature selection method would choose W_1 in the first round, but choose W_3 in the second round, since W_2 adds very little that is not already contained in W_1.

In PITA, this is achieved as follows. In the first round, W_1 reveals the largest randomness deficiency, so it is selected. Then the models are updated using W_1, and new PIT values are calculated. Now, the new histograms corresponding to W_1 will be nearly flat. Since W_2 is very similar to W_1, the W_2 histograms will also be nearly flat, so W_2 will get a low score in the second round. On the other hand, W_3 will still have an uneven histogram, because it is not related to W_1. This means it will get a good score in the second round and be chosen.

The PIT histogram idea is very simple, and yet it allows us to solve the hard problems of feature combination and selection. The reason is that the statistics used are related to the *encoded* data, not the original data. Thus, two problems that are difficult in the traditional view of statistics become easy in the AIT view.

3.5 Computation and Memory Costs

The main computational cost of the PITA algorithm comes from the logging for-loop of Algorithm 1. This loop requires each context fucntion $w \in \mathcal{W}$ to be applied to each data sample, resulting in a computational complexity of $\mathcal{O}(N|\mathcal{W}|)$ per round. For T rounds, the complexity is $\mathcal{O}(TN|\mathcal{W}|)$. Note that while this can be large, it is still linear in N. Compared to the logging step, the model update step is very fast (it is only $\mathcal{O}(N)$). Therefore, if the computational cost of the algorithm is a problem, most attention should be paid to optimizing the logging step. Methods for doing this tend to be technical and application-dependent, so the issue is not pursued further here.

An important potential application of PITA is to very large datasets. Of particular interest are datasets so large that they cannot fit in memory. To process such datasets, batches of data must be read in one by one. After reading a data batch and performing some computations on it, memory must be cleared to make room for the next batch.

This mode of operation is quite constraining. Many algorithms need to refer to each data point repeatedly during the learning process. For example, both the backpropagation algorithm associated with the Multi-Layer Perceptron [Rumelhart *et al.* (1986)] and the Generalized Iterative Scaling algorithm associated with Maximum Entropy models [Darroch & Ratcliff (1972)] require a sum over the entire data set to make small, gradient-descent style parameter updates $\lambda_{t+1} = \lambda_t + \Delta\lambda$. Since the parameter updates tend to be small, these algorithms require many database traversals, incurring a disastrous computational cost if the data set is very large.

The PITA algorithm is well-suited for the large data set mode. In each PITA round, the PIT value associated with a data outcome is logged in the histograms attached to each context function. After this is done, the data sample is not used again until the next round, and its memory can be cleared. This technique requires a slight refinement of the logging step shown in the schematic outline of Algorithm 1. This modification, shown in Algorithm 2, is used to generate the histograms H_0 and H_1 associated with each context function. By using this logging technique, we ensure that each PITA round requires only one traversal of the data set.

Algorithm 2 Refined version of logging for loop.

```
for k = 1 : N do
  for w ∈ 𝒲 do
    if w(C^k) = 1 then
      H_1^w → log-hist-count(φ^k)
    else
      H_0^w → log-hist-count(φ^k)
    end if
  end for
end for
```

Since this modified logging scheme use a separate set of statistics for each w, and in principle $|\mathcal{W}|$ is large, it may appear at first glance to require a lot of memory. Actually the memory requirements are not that large. An important property of the PIT histogram model updates and the associated feature scoring scheme is that they depend a very small number of statistics: the histogram bin counts $[H_0^w, H_1^w]$. All the information necessary to score the features and perform the model ensemble update

is contained in the $[H_0^w, H_1^w]$ statistics. Because of this the memory requirement for logging is only $\mathcal{O}(B|\mathcal{W}|)$, where B is the number of histogram bins used (a typical value for B is 10). Note that the memory requirement does not depend on N.

In Chapter 4 we discuss a set of alternative model updates and associated scoring methods. These updates depend on only one logged statistic per feature, giving memory requirements of $\mathcal{O}(|\mathcal{W}|)$.

3.6 Mutual Information Analysis

It is possible to interpret the histogram KLD score in terms of the mutual information between two special kinds of random variables. Let there be a dataset $\{X^k\}$with models $\{u^k\}$and contexts $\{C^k\}$, and let $\{\phi^k\}$ be the PIT values. There is a context function $w(c) \in \{1, 2, \ldots d, \ldots\}$ which partitions the data into subsets $\{S_1, S_2 \ldots S_d \ldots\}$. There is also a set of histogram bin boundaries $\{\epsilon_0 = 0, \epsilon_1, \ldots \epsilon_J = 1\}$.

Let $\kappa \in [1, \ldots N]$ be a data sample index selected uniformly at random. Let D and A be random variables whose outcomes are related to the choice of κ as follows:

$$\begin{aligned} D &= w(C^\kappa) \\ V &= \{v : \epsilon_{v-1} \leq \phi^\kappa < \epsilon_v\} \end{aligned}$$

Using these definitions the following theorem holds.

Theorem. Assume the ϕ^k variables are uniformly distributed in aggregate, so that $p(A = a) = e_a$. Then the average per-sample savings achievable by applying the PIT histogram transformation to each of the data subsets S_d is equal to the mutual information $I(A, D)$.

Proof. The entropy of A is:

$$\begin{aligned} H(A) &= -\sum_a p(a) \log p(a) \\ &= -\sum_d p(d) \sum_a p(a|d) \log p(a) \end{aligned}$$

While the conditional entropy of A given D is:

$$\begin{aligned} H(A|D) &= -\sum_a \sum_d p(a,d) \log p(a|d) \\ &= -\sum_d p(d) \sum_a p(a|d) \log p(a|d) \end{aligned}$$

The mutual information $I(A, D)$ is related to the entropy $H(A)$ and the conditional entropy $H(A|D)$ as follows:

$$\begin{aligned} I(A,D) &= H(A) - H(A|D) \\ &= -\sum_d p(d) \sum_a p(a|d) \big[\log p(a) - \log p(a|d) \big] \\ &= \sum_d p(d) \sum_a p(a|d) \log \frac{p(a|d)}{p(a)} \end{aligned}$$

The result can be achieved simply by identifying the histogram related quantities with the quantities in the above formula. The probability $p(a|d)$ is the empirical probability that a sample falls into the ath histogram bin partition, given that the context function assigned it to subset S_d. So it is just the normalized histogram bin height $e'_{a,d}$ for one of the data partitions S_d. The term $p(a)$ is the aggregate probability that a sample (in any partition) will fall into a given histogram bin; by assumption this is equal to e_a. Since we use the same histogram bin widths e_a for all partitions S_d, $e_a = e_{a,d}$. Finally the term $p(d) = N_d/N$ is just the size of a given partition divided by the total number of samples. Putting these together, we obtain:

$$I(A,D) = \frac{1}{N} \sum_d N_d \sum_a e'_{a,d} \log \frac{e'_{a,d}}{e_{a,d}}$$

This expression is just the weighted sum of the histogram KLD scores Equation 2.10 for each subset S_d. □

3.7 PIT Analysis for Sequential Data Modeling

Sequential data modeling is the problem of predicting a symbol from the symbols that have preceded it. This can be formalized as finding a probability distribution $p(x|H)$

where x is the next symbol to arrive, and H is the history of symbols. The formulation is quite general, since any multivariate distribution can be rewritten as a product of conditional distributions using the chain rule of probability:

$$\begin{aligned} p(X_1, X_2, X_3 \ldots X_N) &= p(X_N|X_1, X_2 \ldots)p(X_1, X_2 \ldots) \\ &= \prod_{i=1}^{N} p(X_i|X_1 \ldots X_{i-1}) \\ &= \prod_{i=1}^{N} p(X_i|H_i) \end{aligned}$$

By identifying the histories H_i with the contexts C^k, the PITA algorithm can be used to attack this problem. The above formulation is common to a wide variety of applications, including data compression, time series analysis, and natural language processing.

3.8 Layering Models

One of the main strengths of the PIT analysis idea is that it makes no assumptions about the nature of the model ensemble $\{u^k\}$. The method used to construct these models is irrelevant. In particular, the initial models can be obtained as the output of some other modeling method, such as a neural network or a Hidden Markov Model. Any randomness deficiencies detected by the algorithm will allow improved models to be obtained.

One simple example of how initial models can be used arises in statistical natural language processing (SNLP). In SNLP research one typically has a corpus of text available to use for training. A standard task is to predict the next word in a sentence based on the previous words. An obvious first step is to use the corpus to construct N-gram models, the simplest example of which is the bigram. Bigram conditional probability distributions are obtained by simple counting:

$$p(X_{t+1} = V_i|X_t = V_j) = \frac{N_{i \to j}}{N_i}$$

Where V_i, V_j are words, N_i is the number of times V_i is observed in the data set, and $N_{i\to j}$ is the number times the word V_j followed V_i in the data set. These bigram distributions can then be used as the initial models in the PITA algorithm. This requires no modifications to PITA. The idea of layering is illustrated conceptually in Figure 3.2.

If the context functions $\mathcal{W}$ detect substantial randomness deficiencies in the PIT stream $\{\phi^k\}$ generated using the data and the initial models, then the updated models produced by PITA will be better than the previous models in terms of codelength. Of course, there is no guarantee that the $\mathcal{W}$ will reveal any randomness deficiencies. The success or failure depends on the choice of $\mathcal{W}$, and on whether the initial model is a good description of the data. If the initial model is perfect or nearly so, it will be impossible to detect any randomness deficiencies.

Applying PITA as a follow up to any other initial modeling process is therefore something of a "win-win" proposition. If no randomness deficiencies are found, then the initial model is confirmed to be very good. On the other hand, if randomness deficiencies are detected, an updated model ensemble can be obtained that gives better performance than the intial model.

One potential pitfall here is overfitting. If the PITA model updates do not provide significant performance improvement relative to their complexity, overfitting may result. However, a simple modification to the PITA algorithm can be used to avoid overfitting: terminate the algorithm when the codelength improvement per round falls below a certain threshold. The threshold can be determined using the MDL principle [Rissanen (1978)] by counting the number of bits required to encode a context function and its parameters. PITA's ability to adaptively select the model complexity is a strong advantage compared to many other methods in machine learning such as Artificial Neural Networks or Hidden Markov Models. In these latter methods the model complexity depends on various parameters that must be selected by the user: the number of nodes in the neural network or the number of hidden states in the HMM. The standard training algorithms associated with these models do not provide easy mechanisms to adaptively select the model complexity.

A more involved example of PITA's layering ability is given in Appendix C. The "black box" model used in this example is a Markov Random Field.

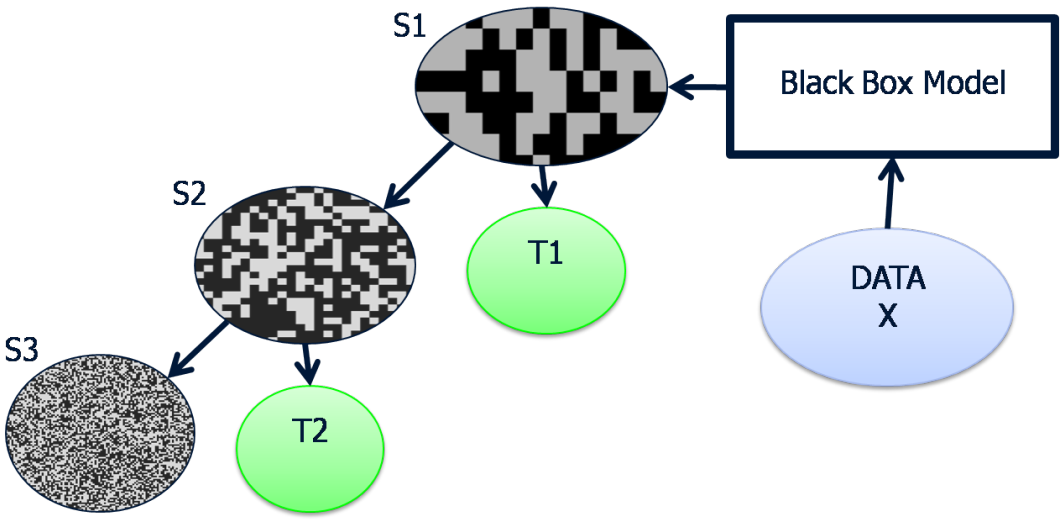


Figure 3.2: Modified version of the AIT view of modeling, in which the initial models are obtained by some "black box" modeling algorithm. Once the black box models are used to obtain the initial set of $\{u^k\}$, the PITA algorithm proceeds exactly as before. If substantial randomness deficiencies are found in the encoded data then updated models will be a significant improvement over the original black box models.

3.9 CDF Reordering

The PITA algorithm is based completely on the use of model CDFs. The cumulative distribution function $\{\mu_i\}$ associated with a discrete probability distribution $\{u_i\}$ is defined as:

$$\mu(x_n) = \sum_{i=1}^{n} u(x_i)$$

One could also define the CDF in terms of the recursion relation:

$$\begin{aligned} \mu_0 &= 0 \\ \mu_i &= u_i + \mu_{i-1} \end{aligned}$$

Either way, it is clear that the CDF implicitly requires an ordering of the outcomes x_i, and that many different CDFs $\{\mu_i\}$ can be obtained from the same PDF $\{u_i\}$ by changing the order of the x_i. In cases where the variable is a discretized version of a numeric outcome, then there is a natural ordering (e.g., $x_i < x_{i+1}$). If there is no such natural ordering, however, then the success or failure of the PIT analysis will be highly dependent on the choice of ordering used.

Consider the following example, which arises in the word morphology application described in Chapter 8. There is a context function $w(c)$ that fires when the previous two letters observed are "gr". The outcomes that occur after this function fires will almost always be vowels. However, a PIT histogram built using a typical alphabetic CDF ordering of the letters (such that x_1 = "a", x_2 = "b" ... x_{26} = "z") will probably fail to reveal a randomness deficiency, or show only a weak one. This is because in the standard ordering, the vowels are scattered throughout the sequence. If a different ordering was used in which all the vowels were grouped together at the beginning of the sequence, a much larger randomness deficiency would be revealed. This idea is illustrated in Figure 3.3.

In fact, there is no reason why we cannot use different variable orderings to construct the CDFs, as long as we are consistent. To do this, we construct "extended" context functions $w' = \{w, O\}$ where $w = w(c)$ is a standard context function, and

$O = \{o_1, o_2 \ldots o_n\}$ is an outcome ordering. The CDF corresponding to O is calculated as:

$$\mu_i = \sum_{i=1}^{n} u(x_{o_i})$$

When logging PIT values for w', new PIT values are obtained based on the new CDF ordered using O. These new values are then used to construct the histogram associated with the extended context function w'.

To update the model ensemble using a context function with a variable ordering O, a three-stage process is applied. First the CDF corresponding to the given ordering is obtained from the model PDF $\{u_i\}$, then the PIT histogram remapping is applied in the CDF space, and then we transform back to obtain a new PDF $\{u'_i\}$.

3.10 Variable Width Histograms for Model Updates

In general, the number of bins J and the the histogram bin widths e_j to use in the logging process are design parameters of the PITA algorithm. Typically the histogram bin widths are chosen to be uniform so that $e_j = J^{-1}$ for all j. However in some cases it is worthwhile to consider using non-uniform bin widths.

To see why, note that it is possible for a PIT histogram to be completely flat, even if the PIT values have a very strong randomness deficiency. For example, consider a PIT distribution where 49% of the values lie in the $[0, 1/4]$ range, 49% lie in the $[3/4, 1]$ range, and 1% lie in both the $[1/4, 1/2]$ and the $[1/2, 3/4]$ ranges. If this distribution is viewed with an evenly spaced 4-bin histogram, a savings of .86 bits per outcome can be achieved. However, if viewed using a standard 2-bin histogram, then the distribution will appear to be completely flat.

The naïve response to this problem is to use a histogram with a very large number of bins. There are several reasons why this is not an optimal solution. First, it increases the amount of memory required in the logging phase. Second, using CDF updates based on large histograms increases the complexity of the model, making it less likely to generalize well. Finally, if the histogram has many bins, then the bins must be very thin, which means that the Bin Overlap problem discussed in Section 3.11 will become more severe.

A general principle of the PITA algorithm is to log many different statistics, but to only use one set of statistics to actually do model updates. For example, in Algorithm 1, histogram counts $H^w_{0,1}$ are logged for many different context functions, but only the counts corresponding to the best context function $H^*_{0,1}$ are used to update the models. This "log many, use one" strategy allows us to achieve the maximum performance benefit for the minimum increase in model complexity.

This strategy can be extended to the choice of histogram bin widths as follows. During the logging step, PIT histograms are constructed with many bins (say, 100). However, these large histograms are not used to perform model updates. Instead, they are collapsed into a smaller histogram with variable width bins. For example, a 100 bin histogram with uniform bin widths could be collapsed into a 2-bin histogram with bin widths $\{e_1 = .35, e_2 = .65\}$. When collapsing a histogram to 2-bins, the optimal bin width can be found simply by recalculating the histogram KLD score for each candidate width. Note that when collapsing a 100-bin histogram into a 2-bin histogram, there are only 99 possible choices for the bin width; the single bin edge in the collapsed 2-bin histogram must correspond to one of the bin boundaries in the original 100-bin histogram.

Using a smaller, variable width histogram for the model update reduces the complexity of the resulting model. A variable-width 2-bin histogram model update is more complex than a fixed-width 2-bin histogram update, but less complex than a 100-bin histogram update. A fixed width 2-bin histogram update requires only one parameter: the expansion factor of one bin (the expansion of the other bin is determined by the normalization requirement). A variable width 2-bin histogram requires two parameters: a width and an expansion factor. A 100-bin histogram update requires 99 parameters. Thus, the variable-width 2-bin histogram is simple, but has an enhanced ability to detect randomness deficiencies.

3.11 Bin Overlap Problem

The main problem with the model update method of Chapter 2 is the crucial "bin containment" assumption that each CDF window would fit cleanly into a histogram bin:

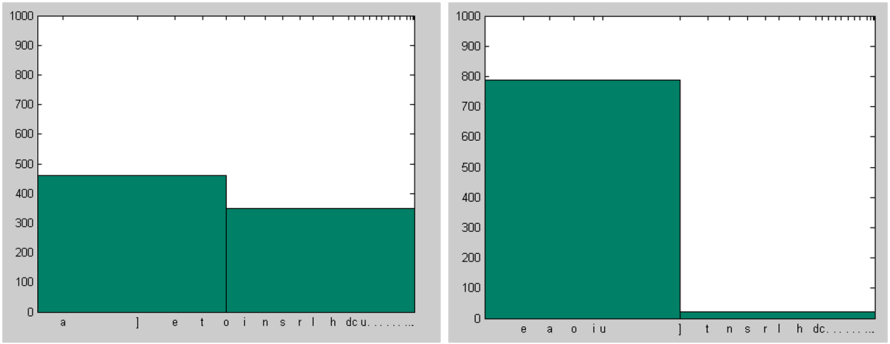


Figure 3.3: Illustration of CDF reordering idea. The data in both histograms are PIT values that follow the prefix "gr". In (a), the default CDF ordering is used. In (b), a new ordering is used, in which the vowels ("aeiou") are placed ahead of the other letters. This reveals a much stronger randomness deficiency than is possible with the default ordering.

$$\epsilon_{j(k)-1} \leq \mu_{-}^{k}(X^k) < \mu_{+}^{k}(X^k) < \epsilon_{j(k)}$$

In other words, the CDF window corresponding to the outcome X^k is completely contained by the jth histogram bin for some choice of j. Because of this assumption, we were able to find the *exact* codelength savings that would result from a CDF remapping update specified by a PIT histogram. The assumption does not hold in general, and so the codelength savings formula (Equation 2.10) will not be precise. The problem is that the CDF remapping transformation will differentially expand or compress a CDF outcome window that spans a histogram bin boundary. This differential updating makes it impossible to calculate the codelength savings directly from the histogram. We call this the "Bin Overlap" problem.

A partial solution to the Bin Overlap problem is presented in Section 3.11.1 below. Before doing so, it is worth analyzing the conditions under which the problem will occur, and why it is a problem.

The Bin Overlap problem occurs primarily when the models assign large probabilities to data outcomes X^k, corresponding to wide CDF windows $[\mu_{-}, \mu_{+}]$. If the CDF windows are large, then it is quite likely that $[\mu_{-}, \mu_{+}]$ will span some bin boundary. Conversely, if the CDF windows are small, then most windows will probably

fall within a single histogram bin. In applications where the data outcomes can take on many values, most of the CDF windows will be quite small, so the Bin Overlap problem is less of an issue.

Note that the accuracy of the histogram KLD score as a prediction of the actual savings degrades smoothly as the Bin Overlap problem becomes more and more severe. That is to say, if the bin containment assumption is true for all of the data outcomes, then the prediction will be perfect. If it is only true for 99% of the outcomes, it will still be highly accurate.

The point of the histogram KLD score is that it allows us to choose the best possible context function. If the score becomes less accurate, then it may interfere with our ability to choose the best context function. However, the context function selected by the KLD score, though it may not be optimal, will still probably be good. Thus, the failure of the bin containment assumption does not break the algorithm, but merely causes some degree of performance reduction.

In Section 3.11.1 below, a method is presented to deal with the Bin Overlap problem. This method is used for all the applications described in this thesis, with the exception of the pattern classification application of Chapter 7. In Chapter 7 we discuss an alternative strategy for dealing with the Bin Overlap problem. This is to use the KLD score merely as a heuristic to generate a list of candidate features. Then, as a post-processing step, a "brute-force" search is used to find the best context function from the list of candidates. This strategy works because of the relatively small size of the datasets typically considered in the pattern classification problem. However, it results in a less pure version of the PITA algorithm.

3.11.1 Fine Graining Technique

This section presents a solution to the Bin Overlap problem based on the idea of *fine graining*. The idea of the fine graining approach is to imagine that there are two data sets we wish to model, the original data set $\mathcal{X}= \{X^k\}$ and an auxiliary data set $\mathcal{Y}= \{Y^k\}$. The goal is to encode both $\mathcal{X}$ and $\mathcal{Y}$ and achieve the lowest possible net codelength for the combined database. Now, we assume that $\mathcal{Y}$ has already been well-modeled and encoded as a bit stream. Thus it does not matter what the original

contents of $\mathcal{Y}$ were; the $\{Y^k\}$ outcomes can be assumed to be just a sequence of bytes. It is also assumed that there are at least as many of these Y^k outcomes as X^k outcomes.

The trick is to encode the X^k and Y^k as joint outcomes $Z^k = \{X^k, Y^k\}$ instead of encoding them separately. This can be done simply by slicing the outcome window assigned to each X^k by its model u^k into 256 subwindows, each corresponding to a Y^k outcome. The joint encoding scheme does not help or hurt us in terms of codelength, as the number of bits required for a joint outcome is just:

$$\begin{aligned} L(Z^k) &= -\log_2 \frac{u^k(X^k)}{256} \\ &= -\log_2 u^k(X^k) + 8 \\ &= L(X^k) + L(Y^k) \end{aligned}$$

Thus, encoding a Z^k outcome costs exactly as much as encoding the corresponding X^k and Y^k outcomes separately. So what does this subdivision of the CDF window achieve? The point is that by making the CDF outcome windows smaller, we make it much less likely for a window to span a histogram boundary. This means that the codelength savings predicted by the histogram KLD score will become much more accurate. Consider the following lemma.

Lemma. Consider a dataset $\mathcal{X}= \{X^k\}$ with associated models $\{u^k\}$, and a two-bin histogram with width distribution $E = [0, 1/2, 1]$, and height distribution $E' = [0, R/2, 1]$, where we assume $1 < R < 2$ for simplicity. Let $N_1, N_{1/2}, N_2$ be the number of left-bin outcomes, boundary-spanning outcomes, and right-bin outcomes respectively:

$$N_1 = |\{k : \mu_+^k(X^k) < 1/2\}| \tag{3.3}$$

$$N_{1/2} = |\{k : \mu_-^k(X^k) < 1/2 < \mu_+^k(X^k)\}| \tag{3.4}$$

$$N_2 = |\{k : \mu_-^k(X^k) > 1/2\}| \tag{3.5}$$

We will call the outcomes corresponding to N_1 and N_2 the "clean" outcomes, since they fit cleanly into a histogram bin. The outcomes corresponding to $N_{1/2}$ are called "dirty" outcomes. The change in net codelength caused by the histogram transformation is bounded by:

$$\Delta L \geq N_1 \log_2(R) + (N_{1/2} + N_2) \log_2(2 - R) \tag{3.6}$$

$$\Delta L \leq (N_1 + N_{1/2}) \log_2(R) + N_2 \log_2(2 - R) \tag{3.7}$$

It follows that the difference between the maximum and minimum possible change in codelength is given by:

$$\Delta L_{max} - \Delta L_{min} = N_{1/2}\big(\log_2(R) - \log_2(2 - R)\big) \tag{3.8}$$

Proof. For the clean outcomes assigned to the left bin, the bin containment assumption (Equation 2.7) holds. The probabilities of all these outcomes are increased by a factor R and so the codelengths are reduced by $\log(R)$. Similarly, for the clean outcomes assigned to the right bin, the codelengths change by an amount $\log(2 - R)$. The codelength changes for the dirty outcomes are clearly bounded by these values. Thus the maximum codelength savings corresponds to the case where the $N_{1/2}$ dirty outcomes are all updated by $\log(R)$, and the minimum corresponds to the case where they are all updated by $\log(2 - R)$. □

This lemma indicates that the critical factor preventing us from using the simple formula for the histogram savings is the number of dirty outcomes $N_{1/2}$. The fine-graining scheme allows us to reduce the number of dirty outcomes, and thus come arbitrarily close to the exact formula for the histogram savings.

To see this, given the dataset $\{X^k\}$, the models $\{u^k\}$, and the auxiliary data outcomes $\{Y^k\}$, define the "fine-grain" CDF points as follows:

$$\mu_-^k(Z^k) = \mu_-^k(X^k) + \frac{u^k(X^k) * (Y^k - 1)}{\tau} \tag{3.9}$$

$$\mu_+^k(Z^k) = \mu_-^k(X^k) + \frac{u^k(X^k) * (Y^k)}{\tau} \tag{3.10}$$

Where we assume $\tau = 256$ since the Y^k outcomes are just bytes. This is nothing more than a way of slicing the CDF window for a particular outcome into τ equally sized bins. Now we can state another lemma.

Lemma. Given a dataset $\{X^k\}$, models $\{u^k\}$, and histogram partition $E = [0, 1/2, 1]$, let $N_1, N_{1/2}, N_2$ be defined as in Lemma 1 above. Now let the auxiliary outcomes $\{Y^k\}$

be drawn uniformly at random on a range $[1, 2 \dots \tau]$, and let the fine grain models be defined as in Equation 3.9- 3.10 above. Finally let $N'_1, N'_{1/2}, N'_2$ be the fine-grain analogues of $N_1, N_{1/2}, N_2$ using the same histogram partition. Then:

$$E[N'_{1/2}] = \frac{N_{1/2}}{\tau}$$

Proof. All outcomes that were clean in the original space will remain clean in the fine-grain space, since:

$$\mu^k_-(X^k) \leq \mu^k_-(Z^k) < \mu^k_+(Z^k) \leq \mu^k_+(X^k)$$

Thus if any outcome is dirty in the fine-grain space, it must also have been dirty in the original space. Now consider a CDF outcome window that was dirty in the original space. Since the fine-grain transition partitions the outcome window into disjoint regions, there is exactly one value of Y^k for which the corresponding fine-grain outcome spans the histogram boundary. By assumption, the probability of this Y^k outcome is τ^{-1}. Therefore, the expected number of dirty outcomes in the fine-grain space is simply $\tau^{-1} N_{1/2}$. □

The Lemma given above refers only to the case of two-bin histograms, but it should be obvious that the same basic idea applies when dealing with multi-bin histograms.

The first Lemma shows that the variation in the savings achieved by the histogram update depends on the number of dirty CDF outcomes. The second Lemma shows that when transitioning to the "fine-grain" outcome space, the number of dirty outcomes is reduced by a factor of τ. Thus, in the fine-grain outcome space, the KLD score of the PIT histograms (Equation 2.10) is an accurate measure of the codelength savings achieved on the *joint* encoding task.

The fine-graining method raises the following question: how do we know the savings we achieve on the combined data $\mathcal{X}+\mathcal{Y}$ corresponds to savings in the real data $\mathcal{X}$, and not the auxiliary data $\mathcal{Y}$? The point is that $\mathcal{Y}$ is random by assumption, so it can never be substantially compressed. Thus the codelength $L(\mathcal{Y})$ is basically constant. So the majority of the savings achieved on $L(\mathcal{Z})$ should correspond to reductions in $L(\mathcal{X})$. Note that we are always able to calculate the codelengths of the original data $L(\mathcal{X})$ and the joint data $L(\mathcal{Z})$, so we can find $L(\mathcal{Y})$ as well, since $L(\mathcal{X})+L(\mathcal{Y})= L(\mathcal{Z})$. There are

some subtleties here, of course. However, the results of Chapter 8 and Chapter 9 provide empirical evidence that the fine-graining scheme works: the histogram KLD score generated by using fine-grained outcomes provides an accurate prediction of savings on the real $\mathcal{X}$ data.

Chapter 4

Alternative CDF Transformation Techniques

4.1 Introduction

The PITA algorithm developed in Chapter 3 used the histogram-based CDF transformation of Chapter 2 to update an ensemble of models. This model ensemble update has several key properties:

- The transformation is determined by a small number of summary statistics related to the PIT values ϕ^k (the histogram bin counts).
- The transformation is defined using a small number of parameters (the normalized histogram bin counts).
- Using only the summary statistics, it is possible to make accurate predictions about the codelength savings that will be achieved by using the transformation to update the model ensemble.
- In order for the predictions to be accurate, the CDF outcome windows had to be small. If this assumption is not met, then the fine-graining trick of Section 3.11.1 can be used to artificially thin the windows.

In this chapter, we will describe a new set of CDF transformation methods that have similar properties. To understand the similarities and differences between the new

Algorithm 3 One round of test selection and scoring.

given data samples $X = \{x^k\}$, associated contexts $C = \{c^k\}$,
PIT values $\{\phi^k\}$, statistical test battery $\mathcal{W}$.
for $k = 1 : N$ **do**
  for $w \in \mathcal{W}$ **do**
    if $w(c^k)$ **then**
      STATS(w) $\rightarrow$ log-stat (ϕ^k)
    end if
  end for
end for
for $w \in \mathcal{W}$ **do**
  PARAM(w) = calc-param(STATS(w))
  SCORE(w) = scoring-function(STATS(w))
end for
$w^* = \arg\max_w$ SCORE(w)
for all k **do**
  if $w^*(c^k)$ **then**
    u^k := cdf-update-function(u^k, param(w^*))
    ϕ^k := cdf-update-function(ϕ^k, param(w^*))
  end if
end for

transformations and the previous one, consider Algorithm 3. This algorithm shows one round of the PITA algorithm, with abstract versions of various operations and quantities. The concrete versions of the abtract concepts corresponding to the PIT histogram transformation are as follows:

- STATS(w) - histogram bin counts.
- PARAM(w) - normalized histogram bin counts.
- scoring-function - histogram KLD score.
- cdf-update-function - histogram based CDF update.

In this chapter we discuss a new set of CDF transformations, with associated methods of computing STATS(w), PARAM(w), and SCORE(w). There is a nice relationship between the form of the transformation function and the appropriate ϕ^k-related statistics that must be logged to optimize the transformation. Note that for the case of the histogram transformation, it is trivial to compute PARAM(w) from STATS(w) - they are essentially the same thing. For the new transformations, the optimal parameter will be a function of the logged statistics, but not always a simple function. Table 4.1 provides a summary of the new transformations and their properties.

Table 4.1: Summary of CDF transformations. The function $\mathbb{1}_j$ is an indicator function returning 1 if its argument is in the jth histogram bin. See text for definition of functions $H(\beta)$, $h(\beta)$, and $d(\phi, f)$.

TEST	STATS	PARAM	SCORE
Histogram	$\gamma_j = \frac{1}{N}\sum_k \mathbb{1}_j(\phi^k)$	$\beta_j^* = \gamma_j$	$\sum_j \gamma_j \log(\gamma_j)$
Power Law	$\gamma = \frac{1}{N}\sum_k \log \phi^k$	$\beta^* = -\gamma^{-1} - 1$	$\log(\beta+1) + \beta\gamma$
Exponential	$\gamma = \frac{1}{N}\sum_k \phi^k$	$h(\beta^*) = \gamma$	$H(\beta^*) - \beta^*\gamma$
Laplacian	$\gamma = \frac{2}{N}\sum_k \lvert\phi^k - 1/2\rvert$	$h(\beta^*) = \gamma$	$H(\beta^*) - \beta^*\gamma$
Shifted Laplacian	$\gamma = \frac{2}{N}\sum_k d(\phi^k, f^k)$	$h(\beta^*) = \gamma$	$H(\beta^*) - \beta^*\gamma$

Note that for practical reasons it is essential for the data contained in the STATS object to be quite small. This is because in principle, the size of the context function battery $|\mathcal{W}|$ is very large. If a large number of summary statistics are required for each context function, then the total memory requirements will be enormous. To use the histogram-based CDF transformation, a set of histogram bin counts must be logged for each context function. If the number of histogram bins is B, then the total memory requirement is $\mathcal{O}(B|\mathcal{W}|)$. The new methods of this chapter will require essentially only one statistic per test, giving memory requirements of $\mathcal{O}(|\mathcal{W}|)$.

The most important transformation developed in this chapter is unfortunately also the most complex. This is the Shifted Prediction Laplacian transformation described in Section 4.6. This transformation makes it possible to to use *predictor* context functions $f = f(c) \in [0, 1]$ instead of the simple binary context functions previously considered. If the predictions $f(C^k)$ accurately predict the ϕ^k, such that the differences $f(C^k) - \phi^k$

cluster around 0, then this transformation will yield substantial savings. This new technique greatly expands the scope of the PIT analysis method.

4.2 CDF Transformation Functions

To begin the development, consider the properties that a CDF transformation function T must have. The function T maps CDF points to new CDF points: $\mu' = T(\mu)$. In order for the updated points $\{\mu'_0, \mu'_1, \ldots \mu'_n\}$ to constitute a valid CDF, the function T must have certain properties:

- It must be a mapping of the form: $T : [0, 1] \rightarrow [0, 1]$.
- The transformation must have $T(0) = 0$ and $T(1) = 1$.
- The transformation must be monotonically increasing: $\frac{dT}{dy} > 0, y \in [0, 1]$.

From these definitions it becomes clear that a CDF transformation is itself just a CDF for a numeric variable distributed on the $[0, 1]$ interval. Furthermore, the derivative $t(y) = \frac{dT}{dy}$ of the transformation can be thought of as a PDF. This makes sense upon reflection: what PIT analysis is really doing is modeling the distribution of the encoded data ϕ.

The derivation given in Chapter 2 produced a formula that predicted the codelength savings achieved by applying the histogram update. This derivation relies on the crucial assumption that each CDF outcome window $[\mu_-, \mu_+]$ was completely contained within a histogram bin. The logic of this chapter relies on a similar assumption, which is simply that the CDF outcome windows are small: $u = \mu_+ - \mu_- \ll 1$.

If this assumption holds, then the change in probability caused by applying some CDF transformation $T(y)$ can be found by Taylor expanding $T(y)$ around μ_-:

$$\begin{aligned}
\mu'_+ - \mu'_- &= T(\mu_+) - T(\mu_-) \\
&= T(\mu_- + u) - T(\mu_-) \\
&= \big(T(\mu_-) + u\frac{dT}{dy}\big|_{y=\mu_-} + u^2\frac{1}{2}\frac{d^2T}{dy}\big|_{y=\mu_-} + \ldots\big) - T(\mu_-) \\
&\approx u \cdot t(\mu_-) \\
&\approx u \cdot t(\phi)
\end{aligned}$$

The approximation $\phi \approx \mu_-$ is valid because $\mu_- < \phi < \mu_+$ and all three values are close together by assumption. This expression shows that the function $t(y)$ plays the role of an expansion factor. CDF outcome windows around ϕ will be expanded or contracted depending on the value $t(\phi)$. The normalization requirement $\int_0^1 t(y)dy = 1$ indicates that in order to expand some regions of the unit interval, other regions must be contracted.

Now, given the database $\{X^k\}$ and corresponding models $\{u^k\}$ as discussed in Chapter 3, the overall change in codelength will be given by:

$$\begin{aligned} L - L' &= -\sum_k \log u^k(X^k) + \sum_k \log u^{k'}(X^k) \\ &= -\sum_k \log u^k(X^k) + \sum_k \log \left(u^k(X^k) t(\phi^k)\right) \\ &\approx \sum_k \log t(\phi^k) \end{aligned} \tag{4.1}$$

This idea provides the foundation for the new types of model ensemble updates to be discussed below. Given a CDF transformation function $T(y)$ with an associated PDF $t(y)$, the codelength change achieved by applying T to all of the $\{u^k\}$ is a simple function of the above sum.

In principle, Equation 4.1 alone is sufficient. To predict the savings resulting from a model ensemble update using function $t(y)$, it is sufficient to traverse the database and compute the sum $\sum_k \log t(\phi^k)$. The drawback is that this requires the specific function $t(y)$ to be known in advance of the traversal. This allows no flexibility in adapting the function $t(y)$ to the statistics of the data. A better strategy is to select a specific $t(y)$ from a family of parameterized transformations $t(y, \beta)$ by selecting an optimal parameter β^*, which is calculated from ϕ^k-related statistics. The following sections show how to do this for certain parameterized functions $t(y, \beta)$.

For the sake of notational simplicity, in this chapter codelength savings are measured in nats. One nat is equal to $1/\log(2) \approx 1.443$ bits.

4.3 Power Law Transformation

Consider transformation PDFs of the form $t(y) = \lambda y^\beta$. To ensure normalization, λ must be chosen such that:

$$\begin{aligned} 1 &= \lambda \int_0^1 y^\beta dy \\ &= \lambda \left[\frac{y^{\beta+1}}{\beta+1} \right]_0^1 \\ &= \frac{\lambda}{\beta+1} \\ \rightarrow \lambda &= \beta+1 \end{aligned}$$

Then, plugging this choice for the function $t(y)$ into Equation 4.1 we obtain the following expression for the codelength savings:

$$\begin{aligned} \frac{\Delta L}{N} &= \frac{1}{N} \sum_k \log t(\phi^k) \\ &= \frac{1}{N} \sum_k \log \lambda + \log(\phi^k)^\beta \\ &= \log(\beta+1) + \frac{\beta}{N} \sum_k \log(\phi^k) \\ &= \log(\beta+1) + \beta\gamma \end{aligned} \tag{4.2}$$

Where the quantity $\gamma = \frac{1}{N} \sum_k \log \phi^k$ has been identified as the relevant statistic for this transformation. Choosing a value β that maximizes the codelength savings ΔL is thus a simple matter of taking derivatives with respect to β. The result is:

$$\beta^* = -\frac{1}{\gamma} - 1 \tag{4.3}$$

Thus, to use the power law CDF transformation, it is necessary to traverse the dataset and compute the sum $\sum_k \log \phi^k$. Knowing this value (and nothing else), allows the optimal power β^* and the resulting savings ΔL^* to be calculated.

4.4 Exponential Transformation

Consider transformation PDFs of the form $t(y) = \lambda \exp(-\beta y)$. Normalization is ensured by choosing λ such that:

$$\begin{aligned} 1 &= \lambda \int_0^1 \exp(-\beta y) dy \\ &= \lambda \left[-\frac{\exp(-\beta y)}{\beta} \right]_0^1 \\ &= \frac{\lambda}{\beta}(1 - \exp(-\beta)) \\ \rightarrow \lambda &= \frac{\beta}{1 - \exp(-\beta)} \end{aligned} \tag{4.4}$$

Plugging this choice for the function $t(y)$ into Equation 4.1 results in the following expression for the codelength savings:

$$\frac{\Delta L}{N} = \log(\beta) - \log(1 - \exp(-\beta)) - \frac{\beta}{N} \sum_k \phi^k \tag{4.5}$$

$$= H(\beta) - \beta\gamma \tag{4.6}$$

Using the shorthand $H(\beta) = \log(\beta) - \log(1 - \exp(-\beta))$, and $\gamma = N^{-1} \sum_k \phi^k$ is the key statistic corresponding to this transformation. The optimum value for β is determined by the condition:

$$h(\beta^*) = \gamma \tag{4.7}$$

Where $h(\beta) = \frac{dH}{d\beta}$, and is given by:

$$h(\beta) = \frac{1}{\beta} - \frac{\exp(-\beta)}{1 - \exp(-\beta)} = \frac{1}{\beta} + \frac{1}{1 - \exp\beta} \tag{4.8}$$

Unfortunately, it is impossible to find an analytic solution to this equation that would give the optimal parameter β^* as a function of γ. At this point it is useful to state a lemma concerning the properties of the function $h(\beta)$.

Lemma. The function $h(\beta)$ of Equation 4.8 has the following properties:

- $\lim_{\beta \to \infty} h(\beta) = 0$
- $\lim_{\beta \to 0^+} h(\beta) = 1/2$
- $\frac{dh}{d\beta} < 0$ for $\beta > 0$

Proof. The first property is evident upon inspection of $h(\beta)$.

The second property can be proved by rewriting $h(\beta)$ and Taylor expanding the $\exp(\beta)$ terms:

$$\begin{aligned} h(\beta) &= \frac{1+\beta-\exp(\beta)}{\beta(1-\exp(\beta))} \\ &= \frac{1+\beta-(1+\beta+\frac{\beta^2}{2}+O(\beta^3)}{\beta(1-(1+\beta+\frac{\beta^2}{2}+O(\beta^3)))} \\ &= \frac{-\frac{\beta^2}{2}+O(\beta^3)}{-\beta^2+O(\beta^3)} \end{aligned}$$

As $\beta \to 0^+$, the above expression converges to $1/2$. The third property can be shown by computing the derivative $\frac{dh}{d\beta}$:

$$\begin{aligned} \frac{dh}{d\beta} &= -\frac{1}{\beta^2}+\frac{\exp\beta}{(1-\exp\beta)^2} \\ &= \frac{\beta^2\exp(\beta)-(1-\exp(\beta))^2}{\beta^2(1-\exp(\beta))^2} \\ &= \frac{(\beta^2+2)\exp(\beta)-(\exp(2\beta)+1)}{\beta^2(1-\exp(\beta))^2} \end{aligned} \tag{4.9}$$

Let us rewrite the numerator of Equation 4.9 as $v-w$, where $v=(\beta^2+2)\exp(\beta)$ and $w=(\exp(2\beta)+1)$. The derivatives of w and v are:

$$\begin{aligned} w' &= 2\exp(2\beta) \\ v' &= 2\exp(\beta)(1+\beta+\frac{\beta^2}{2}) \\ w'-v' &= 2\exp(\beta)(\exp(\beta)-1+\beta+\frac{\beta^2}{2}) \end{aligned}$$

The identity $\exp(\beta)=1+\beta+\beta^2/2+\beta^3/3+\ldots$ implies $w' > v'$. This inequality and the fact that $w(0)=v(0)=2$ by inspection imply that $w > v$ for $\beta > 0$. So

the numerator $v - w$ of Equation 4.9 is negative for $\beta > 0$. The denominator of Equation 4.9 is clearly positive for $\beta > 0$, which implies the third property. □

The result of the Lemma implies that if $\gamma \in [0, 1/2]$, then a solution of the equation $h(\beta^*) = \gamma$ can be found using a fast binary search in β. Note also that since $\phi^k \in [0, 1]$, $\gamma = N^{-1} \sum_k \phi^k \in [0, 1]$. The case where $\gamma \in [1/2, 1]$ is discussed in Section 4.7 below.

4.5 Laplacian Transformation

In this section we will consider transformations of the form $t(y) = \lambda \exp(-2\beta|y - 1/2|)$. This development is basically similar to the one given above for the exponential transformations. The normalization factor can be derived as follows:

$$\begin{aligned}
1 &= \lambda \int_0^1 \exp(-2\beta|y - 1/2|)dy \\
&= 2\lambda \int_{1/2}^1 \exp(-2\beta(y - 1/2))dy \\
&= 2\lambda(\exp\beta) \int_{1/2}^1 \exp(-2\beta y)dy \\
&= 2\lambda(\exp\beta)\left[\frac{\exp(-2\beta y)}{-2\beta}\right]_{1/2}^1 \\
&= \frac{\lambda}{\beta}[1 - \exp(-\beta)] \\
\rightarrow \lambda &= \frac{\beta}{1 - \exp(-\beta)}
\end{aligned}$$

Plugging this choice for the function $t(y, \beta)$ into Equation 4.1, we obtain the following expression for the codelength savings:

$$\begin{aligned}
\frac{1}{N}\Delta L &= \log(\beta) - \log(1 - \exp(-\beta)) - 2\frac{\beta}{N}\sum_k |\phi^k - 1/2| \\
&= H(\beta) - \beta\gamma
\end{aligned}$$

Where $H(\beta)$ is the same as in the previous section, and $\gamma = 2N^{-1}\sum_k |\phi^k - 1/2|$ is the key statistic for this transformation. Using this modified form for γ, we obtain the same optimization criterion $h(\beta^*) = \gamma$. Thus the same fast binary search method can be used to find β^*. In this case the requirement $\gamma \in [0, 1/2]$ implies that the ϕ^k have clustered near the center of the unit interval. The case where γ is outside of this range is discussed in Section 4.7 below.

4.6 Shifted Prediction Laplacian Transformation

The previous section considered a Laplacian transformation centered at $y = 1/2$. This CDF transformation has the effect of expanding the central regions of the $[0, 1]$ interval, while contracting the marginal regions. This transformation is useful if the ϕ^k tend to cluster around $1/2$.

However, sometimes the ϕ^k cluster, but in some other region of the unit interval. Furthermore, sometimes the ϕ^k do not show an aggregate clustering tendency, but it *is* possible to make a context-dependent prediction $f^k = f(C^k)$ about where the PIT value will fall. If the prediction is good, then the differences $\phi^k - f^k$ will cluster around 0. The goal of this section is to find a transformation that can exploit this ability to predict ϕ^k.

The obvious function to use would be $t(y, \beta, f) = \lambda \exp(-2\beta|y - f|)$. The problem with this transformation is that the normalization factor λ becomes a function not only of β, but also of f. Because $f^k = f(C^k)$ is different for each outcome, the code-length savings expression becomes intractable. Specifically, the k-dependent terms do not nicely collapse into a single sum as in Equations 4.2 and 4.6.

This problem can be solved by defining a modular distance function $d(y, f)$ as follows:

$$d(y, f) = \begin{cases} |y - f| & : |y - f| < 1/2 \\ 1 - |y - f| & : |y - f| > 1/2 \end{cases} \tag{4.10}$$

Our expansion function will then be $t(y, \beta, f) = \lambda \exp(-2\beta d(y, f))$. The reason for using this modular distance function is to simplify the expression for the normalization factor λ. As illustrated in Figure 4.1, by using the modular distance scheme,

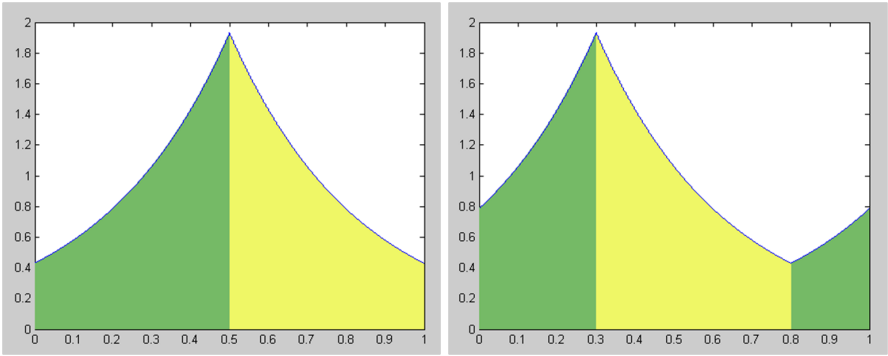


Figure 4.1: The image on the left shows a simple double exponential with $\lambda = 1.5$. The plot on the right shows a modular double exponential, also with $\lambda = 1.5$ and using a prediction of $f = .3$. The green and yellow regions have the same area in both plots. Because of this modular shift of the unit interval, the normalization factor for the PDF depends only on λ and not on the prediction f.

the integral $\int_0^1 t(y, \beta, f)dy$ becomes independent of the prediction f. Thus the normalization factor λ takes on the exact same form as in Equation 4.4. Furthermore, the savings achieved by the transform, and the condition for the optimal value β^*, will be exactly the same as in Equations 4.6 and 4.7, except with a different k-sum:

$$\begin{aligned} \frac{1}{N}\Delta L &= \log(\beta) - \log(1 - \exp(-\beta)) - \frac{2\beta}{N}\sum_k d(\phi^k, f^k) \\ &= H(\beta) - \beta\gamma \end{aligned}$$

Where $H(\beta)$ is the same as above, and $\gamma = 2N^{-1}\sum_k d(\phi^k, f^k)$ is identified as the key statistic for this transformation. Using the new definition for γ, the same optimization criterion $h(\beta^*) = \gamma$ results from attempting to maximize the codelength savings, and the same fast binary search method can be used to find β^*. Here the requirement $\gamma \in [0, 1/2]$ implies that the average modular distance $d(\phi^k, f^k)$ is less than $\frac{1}{4}$.

4.7 Reversing the Exponentials

For the CDF transformations based on the exponential distribution, involving the optimization criterion $h(\beta^*) = \gamma$, we saw that if the condition $\gamma \in [0, 1/2]$ held, then the optimal parameter β^* was positive and could be found with a fast binary search. Also, for all the γ statistics discussed above, it is the case that $\gamma \in [0, 1]$. Thus, to complete the development we must consider the case where $\gamma \in [1/2, 1]$.

This condition corresponds to a situation where the ϕ^k cluster on the opposite location from where the given transformation "expects" them to. In other words, the basic exponential transformation tends to expand the left side (near 0) of the unit interval, while contracting the right side (near 1). If the ϕ^k cluster near 0, then $\gamma < 1/2$ and savings can be achieved by the transformation with $\beta > 0$. If the ϕ^k cluster near 1, however, then $\gamma > 1/2$ and the β-calculation technique breaks down. For the Laplacian transformation, $\gamma < 1/2$ and savings are achieved if the ϕ^k cluster near $1/2$. If the ϕ^k cluster at the edges of the unit interval, the method will break down. A similar idea holds for the Shifted Prediction Laplacian.

Fortunately a simple trick allows us to solve this problem. The trick is just to use a reversed transformation. These reversed transformations are shown in Table 4.2.

Table 4.2: Reversed Exponential Transformations.

Transformation	Reversed Form	STATS
Exponential	$\lambda \exp(-\beta(1-y))$	$\gamma_r = \frac{1}{N}\sum_k (1-\phi^k)$
Laplacian	$\lambda \exp(-2\beta(1/2 - \lvert y - 1/2\rvert))$	$\gamma_r = \frac{2}{N}\sum_k (1/2 - \lvert\phi^k - 1/2\rvert)$
Shifted Prediction	$\lambda \exp(-2\beta(1/2 - d(y, f)))$	$\gamma_r = \frac{2}{N}\sum_k (1/2 - d(\phi^k, f^k))$

As can be seen from the table, the relevant statistics γ_r for each reversed transformation are related to the original statistics: $\gamma_r + \gamma = 1$. Therefore, if for the original transformation we find $\gamma > 1/2$, then the reversed transformation will have $\gamma_r < 1/2$. Thus, we can find an optimal β value for the reversed transformation using the condition $h(\beta^*) = \gamma_r$.

4.8 Summary

In this chapter we discussed several alternate types of CDF transformations that can be used in different situations. The characteristics of these transformations are summarized in Table 4.1.

For the power law, exponential, and basic Laplacian transformations, the main benefit in comparison to the histogram transformation of Chapter 3 is that the new transformations require only a single parameter β^*. In contrast the histogram transformation requires $B - 1$ parameters, where B is the number of histogram bins. Empirically, the distribution of the ϕ^k data is often relatively well described by an exponential or Laplacian distribution. Thus, from the perspective of model complexity, it is better to use a one-parameter transformation if it describes the data reasonably well. A better codelength savings can always be obtained by using enough histogram bins, but this comes at the price of increasing model complexity.

The main important result of this chapter, however, is the shifted prediction Laplacian transformation. This tool significantly expands the power of PIT analysis. Now, the statistical test battery $\mathcal{W}$ can be enhanced to include predictor functions $f(C^k) \in [0, 1]$ as well as simple binary context functions $w(C^k) = \{0, 1\}$. If these predictor functions are good, that is if the modular distances $d(f^k, \phi^k)$ cluster around 0, then substantial savings will be achieved. As is the case with the binary context functions, we can use a "shotgun" approach and define many predictor functions, letting the PITA algorithm select the best ones.

It should be clear that there exist many more valid CDF transformations than those listed in Table 4.1. Of particular interest would be a Gaussian-style transformation of the form $t(y) = \lambda \exp(-\beta y^2)$ or something similar. The problem with this function is that the normalization factor λ takes on a complex form (essentially the error function). The exploration of other transformations remains as future work.

Chapter 5

Related Work

In this chapter, we discuss some ideas and algorithms that are related to the PITA algorithm. The most similar ideas are the Maximum Entropy framework and boosting algorithms (e.g. AdaBoost). Sections 5.1 and 5.2 discuss the similarities and differences between these ideas and PITA. Section 5.3 discusses the relationship between the PIT histogram and various Goodness of Fit tests discussed in the statistical literature. Finally, Section 5.4 discusses alternative uses of the PIT in fields such as forecasting and econometrics.

5.1 Maximum Entropy

The Maximum Entropy (MaxEnt) principle, introduced by Jaynes (1957), suggests choosing a statistical model based on two considerations:

- The model should satisfy a set of empirical constraints specified by the data.
- The model should have the maximum entropy consistent with these constraints.

Here the word "entropy" refers to the information (Shannon) entropy (see Appendix A). The empirical constraints depend on a set of context functions W_i. Given these functions, the constraints are expressed by the requirement: $E_{data}[W_i] = E_Q[W_i]$. In words, the empirical expectations of the context functions must match the model expectations. These expectations are calculated as follows:

$$E_{data}[W_i] = \sum_{k=1}^{N} W_i(X^k) \tag{5.1}$$

$$E_Q[W_i] = \sum_{x} Q(x)W_i(x) \tag{5.2}$$

It can be shown that the two requirements of matching expectations and maximizing entropy imply models of the form:

$$Q(x) = \frac{1}{Z} \exp\left(\sum_i \lambda_i W_i(x)\right) \tag{5.3}$$

Where Z is a "partition function" that ensures normalization, and the λ_i are a set of parameters that must be found using an optimization process. The algorithm most commonly used for this purpose is called Generalized Iterative Scaling [Darroch & Ratcliff (1972)], though other options are available [Malouf (2002)]. The optimization is convex, which ensures that the optimal choice of λ_i is unique, but the optimization algorithm may be quite computationally expensive.

The above, non-conditional form is not very useful, because it does not allow us to include any context information. A modified form is as follows:

$$Q(x|c) = \frac{1}{Z_c} \exp\left(\sum_i \lambda_i W_i(x, c)\right) \tag{5.4}$$

Where the normalization factor Z_c is specific to the context c. The constraints are given by:

$$E_{data}[W_i] = \sum_{k=1}^{N} W_i(X^k, C^k) \tag{5.5}$$

$$E_Q[W_i] = \sum_{x,c} Q(x, c)W_i(x, c) \tag{5.6}$$

The literature records several impressive applications of the MaxEnt framework, especially in the field of statistical natural language processing [Berger *et al.* (1996); Ratnaparkhi (1999); Rosenfeld (1996)]. The main power of the MaxEnt idea is that it allows information from many diverse sources (context functions) to be combined

together to produce one aggregate probability distribution. For example, a well known problem in language modeling is the existence of long range dependencies. Consider the following sentence:

> If the value of the dollar continues to fall, then the Japanese government will be forced to intervene by weakening the yen, in order to protect exporters.

In the above sentence, the word "then" clearly depends strongly on the presence of the word "if" at the beginning. It is very difficult for language models such as N-grams or Hidden Markov Models to express the dependence between the "if/then" pair, because the two words are relatively far apart. In MaxEnt, the relationship can be expressed simply by writing a special context function that refers specifically to the words "if" and "then":

$$W(x,c) = \begin{cases} 1 & \text{“if”} \in c \quad \& \quad x = \text{“then”} \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

While MaxEnt is a powerful and widely used algorithm, it has some drawbacks. One problem is the requirement of finding the λ_i parameters of Equation 5.3. The optimization process requires the repeated computation of the model expectations in Equation 5.6. Note how the sum in this equation runs not only over all outcomes x, but over all contexts c as well. While the space of outcomes (X) is assumed to be relatively small, the space of contexts C is enormous, so this sum is generally intractable. Thus, the following approximation is used instead:

$$E_Q[W_i] \approx \sum_{x,c} Q(x|c)\tilde{Q}(c)W_i(x,c) \tag{5.8}$$

$$= \sum_{k=1}^{N} \sum_{x} Q(x|C^k)W_i(x,C^k) \tag{5.9}$$

Where $\tilde{Q}(c) = \sum_k \delta(c, C^k)$ is the empirical frequency of context C^k. Now, instead of summing over all possible contexts c, we now need to compute sums over the entire data set. Thus, even with the approximation, these sums can be quite costly, especially if the data set is too large to fit in memory. Furthermore, the optimization algorithm

proceeds by making a series of iterative updates to the λ_i parameters until they converge. Each time an update is made, the model $Q(x|c)$ changes, and so the sums must be recomputed.

Another drawback to MaxEnt relates to the feature selection problem. While MaxEnt gives us a principled way of constructing a model from a set of feature expectations, it does not instruct us how to select those features in the first place. Of course, there are several well known methods for feature selection [McCallum (2003); Pietra *et al.* (1997)]. The basic principle is: "freeze" the model by holding the current λ_i values constant. Then add a new feature W_{new} to the model with parameter α. Find the value of α that optimizes the change in log-likelihood, and use this optimal improvement value as a score for W_{new}. Finally, select the new feature that has the best score, add it to the set of W_i, and rerun the λ_i optimization process.

Clearly, this method has several problems. It is not clear how valid the "freezing" approximation is: will the predicted improvement really be close to the actual improvement? Also, the optimization of α requires several traversals of the entire dataset. Finally, the λ-optimization must be re-run after adding each new feature. Of course, the previous λ_i values can be used as a starting point for the subsequent optimization, and this will speed up convergence, but overall it is clear that the MaxEnt feature selection methods are computation-intensive.

5.1.1 Comparison

In several ways, MaxEnt is quite similar to the PITA algorithm. Both MaxEnt and PITA solve the problem of integrating information from multiple context functions. Also, the tendency of the PIT values to become increasingly random is conceptually similar to the idea of maximizing entropy.

It is actually possible to describe PITA as a layered version of MaxEnt in which the empirical constraints come from the *encoded* data, not the original data. In this view, in each PITA round, we build a single feature MaxEnt model of the distribution of PIT values. A PIT histogram bin count plays the same role as an empirical constraint in MaxEnt. Since we are using only a single feature, it is trivial to find the required λ parameter. After building a single-feature MaxEnt layer, we update the encoded data and repeat the process to pruduce the next layer.

In spite of these similarities, PITA appears to have several advantages compared to MaxEnt. As noted above, MaxEnt requires an optimization process to find the λ parameters, which involves sums over the entire data set. If the data set is large, especially if it is too large to fit in memory, computing these sums can be enormously costly. In contrast, the PITA algorithm requires only a single traversal of the data set for each round. All information necessary to choose the best test, and to find the optimal parameters for the test, are contained in the histogram bin counts (or other statistics as discussed in Chapter 4).

As noted in Chapter 1, any feature-based modeling method must solve the twin problems of feature combination and feature selection. The MaxEnt solution to feature combination is not very elegant: it simply relies on large amounts of computation, and on the fact that the optimization is convex. The feature selection method of MaxEnt is also somewhat crude. In contrast, the PITA algorithm contains a "built-in" feature selection method, which has a nice interpretation in terms of the randomness deficiency revealed by a context function.

Both feature scoring schemes depend on an approximation, but the conditions under which the PITA approximation is valid are easily understood: the CDF outcome windows must be small. The conditions under which the MaxEnt approximation is valid are not as well understood.

Perhaps the main difference between PITA and MaxEnt is the shift to the AIT view of modeling. This makes many things more clear. For example, it turns out that it is possible to use MaxEnt for layering, just like PITA. However, this fact is not obvious, and this technique seems to be rarely used. In the AIT view, it becomes clear that *any* kind of randomness deficiency in the encoded data can be used to improve *any* kind of model. For example, if we have a Hidden Markov Model of a time series data set, we can search the encoded data for randomness deficiencies. If one is found, we can immediately obtain an improved model.

5.2 Boosting

When attempting to predict some event, it is often the case that one cannot define any single rule that makes good predictions with high probability, but one can obtain a

large number of rough heuristics with predictive power that is slightly better than random. The goal of boosting is to combine such "weak" hypotheses together to form one "strong" predictor. To achieve boosting, it is necessary to solve the feature combination and feature selection problems mentioned in Chapter 1. Most boosting algorithms proceed greedily: in each round, the weak hypothesis that provides the best improvement is added to the model.

Perhaps the most well-known boosting algorithm is called AdaBoost [Freund & Schapire (1996)]. The key idea of AdaBoost is to maintain a set of weights d_t^k for each data sample X^k. In the t-th boosting round, AdaBoost searches for a weak hypothesis that provides a good prediction on the *weighted* distribution:

$$W_t^* = \arg\min_{W \in \mathcal{W}} \quad \sum_k d_t^k (W(C^k) - X^k) \tag{5.10}$$

Where $\mathcal{W}$ is the set of weak hypotheses (context functions in our terminology), and C^k are the patterns or contexts. Note that in the basic form of AdaBoost, only binary outcomes $X \in \{0, 1\}$ are considered. After a new context function W_t^* is selected in round t, the weights are updated as follows:

$$d_{t+1}^k = \begin{cases} d_t^k & (W^*(C^k) = X^k) \\ \gamma_t \cdot d_t^k & (W^*(C^k) \neq X^k) \end{cases} \tag{5.11}$$

Where $\gamma_t > 1$ is a factor which depends on the weighted error rate achieved by W_t^*, and after the update is applied the distribution is normalized so that $\sum_k d^k = 1$. The effect of this update is that the samples which are incorrectly classified by W_t^* get larger weights, while the weights of correctly classified samples tend to decrease. This weighted distribution mechanism is what allows AdaBoost to solve the feature selection problem. The weights are an approximation of the ability of the current model, which is just the set of W_t^* hypotheses that have been chosen so far, to describe the data. Thus the score is a function of the form $S = S(W, X, Q_{@})$ as required for "smart" feature selection.

Various modifications exist that allow AdaBoost to be used for multiclass problems and regression. However, the original design of AdaBoost is intended to solve the binary pattern classification problem. It is not clear how well-suited the variant

versions are to the other types of problems. For example, to solve the multiclass problem with M_c classes, AdaBoost simply makes M_c copies of the dataset, and attacks the multiclass problem as a larger set of binary problems [Friedman *et al.* (2000)].

There have been several subsequent studies analyzing AdaBoost and describing variants. In [Friedman *et al.* (2000)], AdaBoost was viewed as a form of additive logistic regression, in which each model update attempts to minimize an exponential loss criterion. It was shown by [Kivinen & Warmuth (1999)] that the AdaBoost update to the weight distribution is equivalent to finding the distribution d_{t+1} such that the Kullback-Liebler divergence $D\{d_{t+1}||d_t\}$ with the previous distribution d_t is minimized, while also ensuring that the weighted error of the new weak hypothesis is $1/2$. A variation on this idea is used by InfoBoost [Aslam (2000)], which updates the weight distribution to ensure that there is zero mutual information between it and the prediction of the new weak hypothesis.

5.2.1 Comparison

There are several important similarities between boosting methods and the PITA algorithm described in this thesis. In both cases, the goal is to integrate predictions drawn from many diverse data sources into one combined prediction. Like the boosting algorithms, PITA proceeds through a number of rounds, selecting a new context function in each round. Also, this selection is greedy: the feature that provides the best immediate performance gain is selected.

Furthermore, the PITA algorithm involves an effect (histogram flattening) which is conceptually similar to the decorrelation effects of AdaBoost and InfoBoost. In AdaBoost, the updated weight distribution d_{t+1} is uncorrelated with W_t^*. In PITA, the updated PIT histograms associated with W_t^* should be flat.

Finally, both PITA and AdaBoost include "smart" feature scoring methods. AdaBoost uses the notion of a weighted sample distribution, while PITA uses the idea of the encoded data. These mechanisms take into account the current state of the model, allowing a "smart" score of the form $S = S(W, X, Q_@)$.

However, there are several important differences between boosting methods and PITA. The first difference is that PITA learns conditional density models $Q(x|c)$ while

AdaBoost learns point prediction functions $\hat{X} = F(C)$. While it is difficult to obtain a conditional density from a point prediction, it is easy to go the other way:

$$\hat{x} = \arg\max_x Q(x|C) \tag{5.12}$$

Image compression is an example of an application where this asymmetry is important. Due to lighting effects, it is very common for images to have many extreme pixels (i.e. 0 or 255 for 8-bit pixels). Thus, for many pixels, a good model should include substantial probability at both extreme values of the $[0, 255]$ range, with relatively small probability in the center. Now, AdaBoost can be used in regression mode to make a point prediction of a pixel outcome. But in the case of a pixel with large probability at the extreme values, then the point prediction will be near the center of the range. Any reasonable method for turning a point prediction into a density will assign large probability to the center of the range, if that's where the point prediction falls. For this reason, it is very hard to use regression based techniques to solve the image compression problem.

Another key advantage of PITA compared to AdaBoost is the option of layering. As explained in Section 3.8, given an arbitrary "black box" model for some data set, it is possible to layer a set of PITA transformations over the original model, achieving improved performance. It does not appear to be possible to use AdaBoost in this way.

5.3 Goodness of Fit Tests

The PIT histogram idea discussed in this thesis bears a strong similarity to various kinds of goodness of fit tests used in the statistics literature. In this section, we will describe two important goodness of fit tests and illustrate the differences between these tests and the methods of the thesis.

The main use of goodness of fit (GOF) tests is to determine whether or not a given model is a good fit for a particular empirical data set. This is called the model selection problem. If the GOF test indicates that the data set is very unlikely given the model, one rejects the model.

5.3.1 Kolmogorov-Smirnov Test

Consider a data set $\mathcal{X} = \{X^1, X^2 \dots X^N\}$. The empirical cumulative distribution function of the data set is given by:

$$P_N(x) = \frac{1}{N} \sum_{k=1}^{N} I(X^k, x) \tag{5.13}$$

Where $I(X^k, x) = 1$ if $X^k < x$ and 0 otherwise. Let $P(x)$ be the real cumulative distribution generating the data. The Kolmogorov-Smirnov statistic is defined as:

$$D_N = \sup_x |P_N(x) - P(x)| \in [0, 1] \tag{5.14}$$

One of the most important results in theoretical statistics, the Glivenko-Cantelli theorem, tells us that in the limit of large N, D_N converges to 0 almost surely [Lamperti (1996)]. However, if we look instead at the value $\sqrt{N}D_N$, it turns out that the tendency of D_N to approach zero is balanced by the $\sqrt{N}$ term. Thus $\sqrt{N}D_N$ does not converge to zero, but rather takes on some non-zero random value. Kolmogorov found that the distribution of this random value was related to a type of random walk process called the Brownian bridge $B(t)$:

$$P\Big(\lim_{N\to\infty} \sqrt{N}D_N < x\Big) = P\Big(\sup_{t\in[0,1]} |B(t)| < x\Big) \tag{5.15}$$

The distribution on the right hand side of the above limit does not depend on the data $\mathcal{X}$ or the distribution P and its form (probability distribution function) is known. Given a data set and a model, by using the model $Q(x)$ in place of $P(x)$ in Equation 5.14 we can compute the implied value of $\sqrt{N}D_N$ and look up its probability by consulting the distribution of $\sup B(t)$. If this distribution indicates that the obtained $\sqrt{N}D_N$ is very unlikely, we reject the model Q.

Another test related to the Kolmogorov-Smirnov test is the Cramer-von Mises criterion. The difference is that instead of using the supremum value of $|P_N(x) - P(x)|$, one integrates this function over the range of x. This results in a statistic called ω^2. As in the case of the Kolmogorov-Smirnov test, if the model distribution $Q(x)$ is the same as the real distribution $P(x)$, then the distribution of ω^2 converges to a known function.

5.3.2 Pearson's χ^2 Test

Pearson's χ^2 test is another well-known goodness of fit test [Chernoff & Lehmann (1954)]. This test is better suited to discrete (non-continuous) probability distributions than the Kolmogorov-Smirnov test. To use the test, one first partitions the full data outcome space into a set of bins or regions R_j. If the data is discrete, then choosing the discrete outcomes as the region partition is a natural choice, but outcomes can also be combined to form larger regions (this may be useful if some outcomes are very rare). If the data are numeric, then a set of boundaries must be chosen to determine the region partition.

Given the regions R_j and the model density $q(x)$, one calculates the expected number of observations for each bin:

$$E_j = \sum_{x \in R_j} q(x) \tag{5.16}$$

Where the sum should be replaced with an integral if the distribution is continuous. Then one calculates the observed number of values that fall into a given bin:

$$O_j = \frac{1}{N} \sum_{k=1}^{N} I(X^k, R_j) \tag{5.17}$$

Where $I(X^k, R_j) = 1$ if $X^k \in R_j$ and 0 otherwise. Based on these observed and expected values, one computes the statistic:

$$X^2 = \sum_{j=1}^{J} \frac{(O_j - E_j)^2}{E_j} \tag{5.18}$$

As with the statistics $\sqrt{N} D_N$ and ω^2 discussed above, the X^2 statistic converges to a known distribution, the χ^2 distribution, as the number of observations becomes large. One can then look up the probability of observing a given X^2 statistic, assuming the data is generated by the model distribution $q(x)$, by consulting the χ^2 distribution.

One issue with the χ^2 test is that the χ^2 distribution depends on a parameter called the degrees of freedom. To find the correct large N behavior for X^2, we need to find the correct degrees of freedom parameter. The parameter used in conjunction with this test is $J - K$, where J is the total number of regions R_j and K is the number of parameters in the model $Q(x)$.

5.3.3 Comparison

The PIT histogram method of Section 2.4 is similar in some sense to the GOF tests described above. Like the GOF tests, the PIT histogram contains a set of statistics that depend on both the data and the model. By analyzing these statistics, we can decide if the model provides a good fit to the underlying data.

The PIT histogram is especially similar to Pearson's χ^2 test. In both cases, we are dividing an outcome interval into bins, counting the number of samples that fall into each bin, and comparing the observed number to the expected number. The closer the observed values are to the expected values, the better we judge the model to be. A flat PIT histogram implies that the observed distribution of PIT values is equal to the expected distribution, so the model is good.

There are several important differences between the PIT histogram and the GOF test statistics mentioned above.

- The PIT histogram does not assume we are dealing with a single model. In general we will have many different data points, each with its own model. The PIT histogram provides a summary of the extent to which the *ensemble* of models describes the data set.

- The PIT histogram provides a way of updating the model(s), through the CDF update discussed in Section 2.4.

- The meaning of the score associated with a PIT histogram is different from the score associated with a GOF test. The PIT histogram score indicates the code-length savings that will be achieved by applying the update. The GOF scores indicate the probability of observing a data set assuming the data was generated by a given model.

Thus, although the PIT histogram is conceptually similar to the GOF tests described in this section, we see that it serves a very different purpose.

The following example contrasts the philosophy of this thesis to the mindset of traditional statistics. Assume there is a data set $\mathcal{X}$, and some default model class $\mathcal{M}$ used to describe $\mathcal{X}$. We use the word "default" to mean that there is no real reason to believe that $\mathcal{M}$ describes the data well, but $\mathcal{M}$ has worked in other cases and there is

no reason to believe it does *not* describe the data well. Assume that $\mathcal{M}$ is the family of Gaussians.

A rough description of the approach used in traditional statistics would be as follows. Using the data $\mathcal{X}$ and the model class $\mathcal{M}$, apply some procedure to obtain a set of optimal parameters θ^*, resulting in a specific model $M = \mathcal{M}(\theta^*)$. From this model M and the data, calculate a GOF statistic such as $\sqrt{N}D_N$ or ω^2 as discussed above. Based on the results of the GOF test, determine whether the model M is a good model. If not, we conclude that the Gaussian is not a good choice for the model class $\mathcal{M}$, and go in search of a new model class such as the Laplacian or the Poisson. Note the background assumption here that "real" data is generally described by some relatively simple model class (Gaussian, Laplacian, etc).

The mindset of this thesis is different. We follow the same initial parameter-finding step, obtaining the same initial model M. Then we use the data set and the model to construct a PIT histogram. If the histogram indicates that M is a bad model, we simply *update* M using the histogram, to produce a new model M'. This process may be repeated several times, using different subsets of the data set, with each update improving the model. The resuling complex model bears little relationship to the original model class $\mathcal{M}$.

5.4 Alternate Uses of Probability Integral Transform

The methods of this thesis are based on the Probability Integral Transform, an idea which has been known in statistics for some time. The basic property of the Probability Integral Transform is as follows. Let $P(x)$ be the cumulative distribution function for some continuous random variable X: $P(x) = Pr(X < x)$. If we define the random variable $Y = P(X)$, then the distribution of Y is uniform on $[0, 1]$ [Angus (1994)].

The most basic use of the PIT is to do model selection. The logic is as follows. In real world statistics, we rarely know the real distribution $P(x)$ generating the data. Rather, we have a model distribution $Q(x)$, and we want to know if it is good. Due to the special property, if $Q \approx P$ than the resulting PIT histogram will be nearly flat ($U(0, 1)$). On the other hand, if the PIT histogram is substantially uneven, then we should reject the model.

The main power of the PIT, emphasized in this thesis, is that it can be used when there are many different models associated with many different outcomes. If the models (or predictions) are created by a human expert, then the PIT distribution can be viewed as a way of measuring the calibration of the expert. In particular, we would like to know if his predictions are overconfident or underconfident. Overconfidence means that the expert's predictions are too sharp relative to his actual predictive ability. Thus he might say: "Next year, GDP growth will be between $14 billion and $15 billion". If the actual GDP growth turns out to be only $10 billion, then we conclude that the expert's prediction was overconfident (for this trial). Conversely, the expert might try to avoid being proved wrong by saying: "Next year, GDP growth will be between $0 and $100 billion". We call this prediction underconfident. Underconfident predictions are more likely to be correct, but because they are so vague, they are not very useful.

If the expert only makes predictions relating to one type of quantity, then there are a variety of ways to measure his accuracy. However, consider the case where the expert is asked to make various predictions relating to quantities such as GDP growth, interest rates, currency exchange rates, stock prices, and so on. In this case it is hard to gauge his calibration, since the predictions deal with outcomes from different spaces. Exchange rate fluctuations are measured in units like dollar/yen, while GDP growth might be measured in billions of dollars.

The PIT idea provides a way to measure the calibration of an *ensemble* of models. Instead of analyzing the distribution of the actual data outcomes, one transforms them into PIT values using the associated models. Say the expert predicts a Gaussian distribution for GDP growth that has a mean of $15 billion and a standard deviation of $1 billion. Then if the real GDP growth is $10 billion, the resulting PIT value will be quite small. By aggregating all of the PIT values from many prediction-outcome trials, and plotting the results in a histogram, we can measure the expert's calibration. A histogram in which most of the values are clustered at the extremes indicates the expert is overconfident. If the PIT values are clustered near the center, the expert is underconfident, while a flat histogram indicates good calibration.

The Probability Integral Transform has been discussed in some research in fields such as economics and finance, mostly for the kinds of purposes mentioned above.

For example, [Diebold *et al.* (1998)] uses the PIT to evaluate density forecasts. Similarly, [Gneiting *et al.* (2007)] also used the PIT to evaluate density forecasts, but added a another term that measured the predictive sharpness of the forecast. There appear to be few examples of research using the PIT idea in the machine learning literature. One such example is [Learned-Miller & John (2003)], where a PIT-related scoring function is used to define an Independent Components Analysis (ICA) algorithm. Other than that reference, the PIT idea is not widely used in machine learning.

The PIT is also related to the copula, a mathematical construction that has generated a huge amount of interest in the financial statistics literature [Embrechts *et al.* (2002); Frees & Valdez (1998); Genest & Rivest (1993)]. The idea behind the copula is to map arbitrary multivariate distributions from some original D-dimensional data space to the $[0,1]^D$ unit hypercube. This is done in such a way that the marginal distributions of each variable are $U(0,1)$. A famous theorem by Sklar (1959) shows that this can be done for any continuous distribution. Once the distributions have been mapped to the $U(0,1)$ space, the dependencies between the variables can be modeled using various standard methods. It is much easier to find a good multivariate model for the $[0,1]^D$ space than for a space in which each dimension has a very different range (GDP growth, interest rates, etc.)

5.4.1 Comparison

As noted, the PIT idea does not seem to be used much in the machine learning literature. Some interest has been shown towards the PIT in the fields of finance, economics, and weather forecasting. In these fields it is useful to be able to evaluate the predictive quality of an ensemble of models. The PIT is also sometimes used to do model selection, by exploiting the idea that a good model should produce a flat PIT histogram.

The use of the PIT idea in this thesis goes much further than these previous applications, in several ways. The following concepts seem to be novel:

- The use of the PIT histogram to define a model update that can be applied to an ensemble of models simultaneously.
- The use of the histogram KLD score to measure the codelength savings achieved by applying the update.

- The identification of the PIT values with the encoded data in the AIT view of learning, and the resulting approach to statistical modeling based on the search for randomness deficiencies.

Chapter 6

PIT Analysis for Image Compression

6.1 Introduction

As noted in the previous chapters, many different problems can be formulated as an attempt to obtain good approximations $Q(x|c)$ of a real complex conditional distribution $P(x|c)$. One such problem is lossless image compression. Here the x data is a pixel value, and the c is the history of previous pixels.

Lossless image compression is a good starting point because it provides a rigorous test of the core idea. To see why, consider the testing process it involves. One first invokes the encoder and obtains a compressed file. One then measures the length of the compressed file (shorter codelengths are better). Finally, one invokes the decoder on the compressed file, and checks that the output is *exactly* the same as the original. Image compression is a rigorous test: if the algorithm contains a conceptual or technical flaw, either it will fail to achieve good compression, or the decoded image will be corrupt. Thus, the fact that the PITA algorithm can be used for image compression proves that the ideas are basically sound.

The image compression application discussed in this chapter is roughly similar to the raw data (e.g. speech) modeling applications discussed in the subsequent chapters. In each case, a large quantity of raw data is available, and the goal is to find the best possible model for the data. However, an important difference in the image compression application is that here the entire dataset comes from a *single* image. A 640x480x3 image contains about $9.2 \cdot 10^6$ pixels, so this is quite a large number of outcomes.

6.2 Image Compression Background

There are two basic kinds of data compression. In *lossless* compression, the data that comes out of the decoder is exactly the same as the data that goes into the encoder. In *lossy* compression methods, the data that comes out of the decoder is not required to be exactly the same as the original data.

A typical image contains a lot of precise information that the human eye is not capable of perceiving. By dropping components of images that the eye cannot perceive, improved compression rates can be achieved. Thus a major component of research in lossy image compression involves human perception. The question is: in what ways can an image be corrupted by data loss, and still remain "good enough" perceptually? Obviously, this kind of research tends to be somewhat subjective.

In contrast, lossless compression is entirely objective. Given two compression algorithms, one can easily decide which algorithm is better by testing them both on a large database. The algorithm that achieves a shorter compressed file size is better.

Data compression involves two conceptually distinct problems: *modeling* and *encoding*. Modeling is the problem of finding good distributions for pixel outcomes as discussed above. The encoding problem is: given a probability distribution $Q(x)$ for a pixel and the actual pixel outcome X, obtain a sequence of bits s_x that represents the outcome. As mentioned in Appendix A, a technique known as arithmetic encoding [Witten *et al.* (1987)] provides a very good solution to the encoding problem. So the research in this chapter focuses exclusively on the modeling problem. The full image compression algorithm discussed in this chapter combines the PITA algorithm with arithmetic encoding.

This mindset contrasts with the approach taken by "traditional" compression algorithms, which do not make an explicit conceptual separation between modeling and encoding. Consider for example the LZW algorithm [Ziv & Lempel (1978)]. This algorithm operates by running a sliding window over the history of outcomes. Instead of encoding an outcome explicitly, it instead sends a pointer to the location of the previous occurrence of the outcome in the history. If an outcome occurs frequently, it is likely its previous occurrence was quite recent. Pointers to recent positions are encoded using a small number of bits, and so frequent outcomes tend to get short codes.

The recent literature on image compression also tends to exploit the conceptual separation between modeling and encoding made possible by the advent of arithmetic encoding. However, the general trend is not necessarily toward obtaining the best possible models (and thus compression rates). This is because highly optimized systems used to obtain the best possible compression rates tend to be too slow for practical use. Thus, substantial research is dedicated to finding ways to compute good codes rapidly. One innovation in this area is the use of Golomb codes, which are optimal for a certain type of distribution [Weinberger *et al.* (2000)]. Empirically, it is found that the distribution of prediction errors generated using standard pixel prediction techniques roughly matches the distribution for which Golomb codes are optimal. Encoding a pixel error value using a precomputed Golomb code takes much less time than encoding the value using arithmetic encoding.

Another line of research involves the question of how to rapidly combine context information to produce a pixel prediction or probability distribution. In the Glicbawls compression algorithm, the pixel is predicted using a least squares method based on the already processed regions above and to the left of the current pixel [Meyer & Tischer (2001)]. CALIC, another successful image compression algorithm, predicts pixels using the local image gradient [Wu & Memon (2000)]. The prediction is also weighted using context adaptive error feedback.

6.3 Adapting PITA for Image Compression

Lossless image compression requires a model of the probability of an image $Q(I)$. However, the space of image outcomes is far too large to model directly. Instead, the problem is broken up into a series of conditional probability distributions, in which the probability of each pixel is conditioned on the previous pixels:

$$\begin{aligned} Q(I) &= Q(X^1) \cdot Q(X^2|X^1) \cdot Q(X^3|X^2, X^1) \ldots \\ &= \prod_k Q(X^k|C^k) \end{aligned} \tag{6.1}$$

This results in a codelength of:

$$L(I) = -\sum_k \log Q(X^k|C^k)$$

Where X^k is the kth pixel and C^k is the history of previous pixels up to k. To achieve the best possible codelengths it is therefore necessary find a good model such that $Q(x|c) \approx P(x|c)$, where $P(x|c)$ is the "real" conditional distribution. Thus, the PITA algorithm can be adapted for the image modeling problem simply by identifying the data outcomes $\{X^k\}$ as the pixels, and the contexts $\{C^k\}$ as the pixel histories. The initial models are uniform over the range of values a pixel can take on, which in this work is set to be $[0, 255]$. The context functions $\mathcal{W}$ are discussed in Section 6.3.1 below.

The algorithm has two phases, a learning phase and an encoding phase. In the learning phase, the PITA algorithm is run. The output of this phase is a set of final models $\{u^k\}$ along with tests w_t^* and transformation parameters $[H_{0,t}^*, H_{1,t}^*]$. Both of these outputs are required for the "encoding" phase. The indices of the selected tests w_t^* are encoded at the beginning of the compressed file, followed by the transformation parameters $[H_{0,t}^*, H_{1,t}^*]$. No special compression schemes are used to encode this model information, which is small compared to the pixel information. Following the context function information comes a bit string which represents the pixels $\{X^k\}$ encoded using the associated models $\{u^k\}$.

The decoding process is basically simple. The receiver first reads the context function w_t^* and parameter information $[H_{0,t}^*, H_{1,t}^*]$. Then, for each pixel, the decoder starts with a uniform model, and updates the model using the context functions and the histogram transformations. The resulting model is used to decode the pixel. The decoded pixel is then placed into the history (context) C for use in decoding the next pixel. Thus, for every pixel, the decoder has the exact same set of context functions, parameters and history data used by the encoder.

The following implementation note is worth mentioning. The PITA algorithm calls for the encoder to maintain separate models $u^k(x)$ for each pixel. If there are 10^7 pixels (a medium size image) then storing all of the models will require storage of $256 \cdot 10^7$ values. This is a large memory requirement. Furthermore, if an attempt were made to encode an image with a larger range of pixels (e.g. 16-bit or $[0, 65335]$), then this problem would become very severe. Fortunately, however, it is not necessary to

maintain the full CDF for each pixel in the PITA algorithm. Only three numbers per pixel are actually required: $\mu_-^k(X^k)$, ϕ^k, and $\mu_+^k(X^k)$. These represent the lower CDF bound, the PIT value, and the upper CDF bound respectively.

On the decoding side, it is not necessary for the decoder to keep the model u^k after the pixel X^k has been decoded. Thus, memory is not a problem for the decoder, but the decoder faces a different computational problem. Unlike the encoder who only needs the CDF bounds $\mu_-^k(X^k)$ and $\mu_+^k(X^k)$, in a naïve application the decoder must compute the entire CDF for each model u^k, from the w_t^* and $[H_{0,t}^*, H_{1,t}^*]$ information.

However, another optimization can be used that allows the decoder to avoid the requirement of computing the full CDF. Instead, the decoder performs a kind of binary search, evaluating the CDF only at a small number of points. Without getting into the details of arithmetic encoding, it can be understood as transmitting a value $\gamma \in [0, 1)$ from the encoder to the decoder. The decoder compares γ to the values of the model CDF to find the actual outcome. For example, consider the case where the actual outcome is $X = 145$. Then the decoder evaluates μ at eight different values as follows:

$$\gamma \geq \mu(128), \gamma < \mu(192), \gamma < \mu(160), \gamma \geq \mu(144)$$
$$\gamma < \mu(152), \gamma < \mu(148), \gamma < \mu(146), \gamma \geq \mu(145)$$

Then because $\mu(145) \leq \gamma < \mu(146)$, the decoder concludes that the outcome is $X = 145$. Thus, the decoder only has to evaluate μ at 8 different points, instead of the 256 points that would be required in a naïve application. This would be even more important if 16-bit images were used instead of 8-bit images.

6.3.1 Context Functions

This section describes the context functions $\mathcal{W}$ used for the application. As is generally true, using a larger number of tests (casting a wider net) leads to greater savings. However, it also leads to larger computational cost. Also, as more and more transformations are applied, it becomes harder and harder to find further randomness deficiencies. We have experimented with the following context function definitions:

- General Context Function (GCF): fires for all pixels.

- Median Edge Predictor (MEP): predict the pixel value using the Median Edge Predictor [Memon *et al.* (1997)].
- Color Context Function (CCF): fire for one and only one color channel.
- Red Difference Predictor (RDP): attempt to predict the value of the blue and green pixels based on the difference between the value of the current red pixel and the previous red pixel.
- Gradient Predictor (GP): predict the value of the target pixel using the simple formula $x = 2a - b$, where a and b are the values of the pixel one element back and two elements back, respectively.
- Abnormality Context Function (ACF): fires if one of the adjacent pixels was assigned a low probability.
- Rectangular Region Context Function (RRCF): fires for all pixels in a simple rectangular region.

For the items in the above list named "Context Functions" (GCF, CCF, etc) the standard histogram-based CDF transform is used with 64-bin histograms. For items named "Predictors", the Shifted Prediction Laplacian Transform of Section 4.6 is used. The above list is a tiny fraction of the full set of context functions that can be defined based on the pixel history. Further investigation and development of more advanced context functions and predictors should lead to further improvements in code length.

6.4 Experimental Results

In this section we evaluate the compression rates achieved by the PITA algorithm. The compression results are compared to the well-known lossless image compression format PNG[1]. The PNG results are intended to provide a rough guide to how compressible the images are.

We used two image databases to test the compression results. The first set is derived from the standard Corel 5K stock photo database. We took the first 5 images from each

[1] http://www.w3.org/TR/PNG/

Table 6.1: Compression results achieved by the PITA algorithm and by PNG on two image datasets. Totals are given in millions (10^6) of bits.

Dataset	Flat Total	PITA Total	PITA Ratio	PNG Total	PNG Ratio
Corel	9437	4880	0.517	4778	0.506
Hongo	7373	3646	0.494	3856	0.523

subfolder in the "Lib1" section of the database, for a total of 1000 images. The second set of 1000 images was obtained from a trial run of an autonomous robot exploring the local university campus ("Hongo"). Both datasets were originally encoded in PPM format, and transformed into PNG using Matlab's "imwrite" command.

Table 6.1 gives a brief summary of the compression results. The column "Flat Total" shows the total codelength that would be required to encode the dataset using a naïve 8-bit per pixel code. The total codelengths used by PITA and PNG are also shown. Note the large scale of the values (10^6 bits).

We also examined the bit savings achieved by each context function and predictor. Results for ten test images are shown in Table 6.2. Obviously, the majority of the savings is produced by the Median Edge Predictor (MEP). This is due not only to the predictive power of this function, but also to the fact that it is applied first. The reported values are the actual savings, not the predicted savings from the histogram KLD score.

6.5 Summary

The main purpose of this chapter was to provide a proof of concept of the PITA algorithm. Since lossless image compression is a hard test, by showing that PITA can be used for this purpose, we prove that the algorithm has no deep conceptual flaws. The compression rates are encouraging, but not revolutionary. The PITA algorithm's performance depends strongly on the choice of context functions $\mathcal{W}$ used. More careful research into the choice of context functions may yield compression rates that are closer to the state of the art.

It is worth noting the large scale of the codelength savings achieved in this experiment. As shown in Table 6.1, the total codelengths involved are on the order of 10^9 bits. This large scale implies that there is much room to justify complex models when

Table 6.2: Results for different bit savings achieved by various context functions and predictors are here. Savings are given in 1000s of bits.

Image	MEP	CCF(r,g,b)	RDP	ACF	Total PITA	Total PNG
corel0	4231.8	21.1	0	139.8	4343.5	4611.0
corel1	4887.5	15.4	0	118.7	4971.3	5409.0
corel2	4885.6	11.8	61.0	194.1	5100.2	5412.1
corel3	3035.4	88.9	11.9	230.1	3322.5	3029.0
corel4	3344.9	31.6	2.9	205.8	3537.1	3633.4
hongo0	2431.0	80.8	16.9	54.4	2544.5	2438.3
hongo1	2882.3	142.6	40.9	49.8	3076.3	2927.8
hongo2	3514.9	66.2	2.6	90.4	3631.7	3377.6
hongo3	3272.2	41.5	4.8	89.7	3365.5	3171.6
hongo4	3109.4	182.3	47.0	57.0	3355.3	3195.5

dealing with such large databases. For example, consider a complex model that can be used to reduce the total codelength by 1%. This corresponds to a savings of about 90 million bits. Thus, even if the model costs 10 million bits, (far larger than the models typically used in machine learning), the overall savings will still be very large [Burfoot & Kuniyoshi (2009)]. This net savings justifies the model in the MDL sense [Rissanen (1978)]. This theme appears again in Chapter 9.

Chapter 7

PIT Analysis for Pattern Classification

7.1 Introduction

In this chapter we apply the PITA algorithm to the problem of binary pattern classification. The result is immediately recognizable as the training phase of a boosting algorithm. The idea of boosting is to combine information from multiple "weak" predictors together to produce a single "strong" classifier [Schapire (1990)]. The weak predictors used in boosting play the same role as the battery of context functions $\mathcal{W}$ in the PITA algorithm. There is a very strong structural similarity between FundaBoost and the well-known AdaBoost algorithm [Freund & Schapire (1997)], which allows us to make very clean comparisons between the two. AdaBoost, variants of AdaBoost, other boosting algorithms, and the relationship between these algorithms and PITA are discussed at length in Chapter 5.

There is an important practical difference between the pattern recognition application discussed in this chapter, and the other uses of the PITA algorithm described in this thesis. In the other applications, we are interested in building models of raw data (images, words, speech, *etc.*) In pattern classification, the goal is to model the conditional probability of a set of label data given a corresponding set of data objects. Since the modeling target (X^k data) is the set of labels, the information content of the data being modeled is quite small: on the order of one bit per data sample. The median size of the domains used in the experimental results section is 3160, which corresponds to an information content of just under 400 bytes. In this extremely low information

content regime, it is very difficult to justify the use of *any* model by the MDL principle [Rissanen (1978)]. In contrast, the raw-data modeling applications discussed in the other chapters involve much larger data sets (the speech database of Chapter 9 contains more than $4 \cdot 10^8$ bits).

Because of the small size of the data set considered in the pattern classification problem, a "pure" application of the PITA algorithm does not yield competitive performance. The Bin Overlap problem described in Section 3.11 becomes particularly serious when dealing with binary outcomes, since the CDF window for an outcome will often be quite wide, and thus very likely to overlap a histogram bin boundary. Below we describe several modifications to the base PITA algorithm, which improve its performance for limited data problems.

7.2 FundaBoost

In this section we describe FundaBoost, which is the result of using the PITA algorithm, with some modifications, to do pattern classification. In this domain, we have a set of binary labels, corresponding to the data outcomes X^k in the PITA algorithm. The goal is to predict the labels from a set of data objects, which correspond to the PITA contexts C^k. The statistical tests $\mathcal{W}$ are defined based on the raw data objects; these are equivalent to the weak predictors used in boosting. The PITA algorithm is the *training* phase of the boosting process; the outputs are the weak predictors w_t^* and parameters H_t^*.

Because there are only two outcomes, we use two-bin PIT histograms. A two-bin histogram has only one free parameter, so in each boosting round we choose two parameters $\lambda_{0,t}^*$ and $\lambda_{1,t}^*$ corresponding to $w_t^* = 1$ and $w_t^* = 0$ The w_t^* and λ_t^* obtained from the training process can be used to classify new unseen data objects, as shown in Algorithm 5.

The goal of FundaBoost is to obtain good models $Q(x|c)$ of the probability of a label given a data object. The $u^k = Q(x|C^k)$ distributions maintained by PITA as simply the current model probabilities of a positive or negative outcome for the label. Progress is measured by the codelength (negative log likelihood) that would be required to encode the X^k data using the current models u^k.

If necessary, model probabilities can easily be transformed into predictions. Given a model CDF $\mu = [0, a, 1]$, the corresponding prediction is $\hat{x} = 1$ if $a > 1/2$ and $\hat{x} = 0$ otherwise. Using this relationship it is easy to calculate the current predictions $\hat{x}^k$ for each of the current models u^k. The number of samples incorrectly classified by the models u^k is equal to:

$$E_{train} = |\{k : u^k(X^k) < 1/2\}| \tag{7.1}$$

This implies that the number of errors on the training data is bounded by the current codelength:

$$\begin{aligned} |\{k : u^k(X^k) < 1/2\}| &= \sum_{k:u^k(X^k)<1/2} (1) \\ &\leq \sum_{k:u^k(X^k)<1/2} -\log_2 u^k(X^k) \\ &\leq \sum_k -\log_2 u^k(X^k) \\ &= L(\mathcal{X}) \end{aligned}$$

As noted above, because of the small size of the data being modeled we must be very careful to conserve as many bits as possible. In particular this means that the inaccuracies introduced by the Bin Overlap problem (Section 3.11) will be too significant to overlook.

While the histogram KLD score is no longer a highly accurate predictor of the savings achievable by a context function, it still contains substantial predictive power. It is likely that the context function which actually produces the optimal savings will also reveal a substantial randomness deficiency in the encoded data. Thus, the histogram KLD score can be used as a heuristic to help choose the best context function from a large list. The modified version of PITA used in this chapter uses the histogram KLD score to produce a heuristic ranking of the top M tests. Then, this reduced set of candidate context functions are evaluated using a slower, brute-force method described below. Because we have reduced the number of context function candidates, the brute-force method still runs in a reasonable time. This modified version of the PITA algorithm, called FundaBoost, is shown in Algorithm 4.

Algorithm 4 FundaBoost: modified PITA algorithm for pattern classification.

given data samples $\mathcal{X} = \{X^k\}$, associated contexts $\mathcal{C} = \{C^k\}$,
context functions $\mathcal{W}$.
initialize models $\mathcal{U} = \{u^k\}$
$\Phi = \{\phi^k\}$ = $\mathcal{U} \rightarrow \text{ProbIntTrans}(\mathcal{X})$
for $t = 1 : T$ **do**
  for $w \in \mathcal{W}$ **do**
    $\Phi_0 = \{\phi^k : w(C^k) = 0\}$, $H_0 = \text{histogram}(\Phi_0)$
    $\Phi_1 = \{\phi^k : w(C^k) = 1\}$, $H_1 = \text{histogram}(\Phi_1)$
    heuristic-score(w) = $|\Phi_0| \cdot \text{KLD}(H_0) + |\Phi_1| \cdot \text{KLD}(H_1)$
  end for
  $\{a\}$ = lower-bin-prob(X, U)
  $\{b\}$ = upper-bin-prob(X, U)
  let $\mathcal{V}$ be the M top tests by heuristic-score
  for $w \in \mathcal{V}$ **do**
    $\lambda_{0,w}$ = exact-lambda-search(w, $\{a\}$, $\{b\}$) // binary search
    $\lambda_{1,w}$ = exact-lambda-search($\neg w$, $\{a\}$, $\{b\}$)
    S_0 = exact-savings($\lambda_{0,w}$, w, $\{a\}$, $\{b\}$) // Equation 7.2
    S_1 = exact-savings($\lambda_{1,w}$, $\neg w$, $\{a\}$, $\{b\}$)
    exact-score(w) = $S_0 + S_1$
  end for
  let $\{w_t^*, \lambda_{0,t}^*, \lambda_{1,t}^*\}$ be the best test, parameters
  for all k **do**
    if $w_t^*(C^k)$ **then**
      $u^k := \lambda_{1,t}^* \rightarrow$ cdf-remap(u^k)
      $\phi^k := \lambda_{1,t}^* \rightarrow$ cdf-remap(ϕ^k)
    else
      $u^k := \lambda_{0,t}^* \rightarrow$ cdf-remap(u^k)
      $\phi^k := \lambda_{0,t}^* \rightarrow$ cdf-remap(ϕ^k)
    end if
  end for
end for
output final distributions u^k, optimal tests w_t^*,
transformation parameters $\lambda_{0,t}^*, \lambda_{1,t}^*$

Algorithm 5 The classification algorithm.

Given: learned parameters $w_t^*, \lambda_{0,t}^*, \lambda_{1,t}^*$
new sample c to classify
initialize $\mu = \{0, .5, 1\}$ // model CDF
for $t = 1 : T$ **do**
  if $w_t^*(c) = 1$ **then**
    $\mu := \lambda_{1,t}^* \rightarrow$ cdf-remap(μ)
  else
    $\mu := \lambda_{0,t}^* \rightarrow$ cdf-remap(μ)
  end if
end for
if $\mu_1 > .5$, output TRUE, else output FALSE

7.2.1 Exact Method for Finding Histogram Parameters

The post-processing method for selecting the optimal CDF remapping parameters λ^* can be understood by considering the update indicated by a two-bin histogram transformation (Equation 2.4). Let the pre-update CDF be given by $\mu^k = \{\mu_0^k = 0, \mu_1^k, \mu_2^k = 1\}$. Let $\mu_1^k = A^k + B^k$, where A^k is the region of the probability window that falls on the left side of the histogram bin partition, and B^k is the region that falls on the right side. The CDF remapping rule applies a factor λ to the left bin, and a factor $(2 - \lambda)$ to the right bin. So the post-update CDF will be given by: $\mu_1^{k'} = \lambda A^k + (2 - \lambda)B^k$.

To find the optimal choice for λ using this update scheme, we must take the label data into account. Let variables (a^k, b^k) be equal to (A^k, B^k) if the actual outcome if X^k is true, and $(.5 - A^k, .5 - B^k)$ otherwise. Then the pre-update probability assigned to the correct outcome is $u^k = a^k + b^k$, and the post-update probability is $u^{k'} = \lambda a^k + (2-\lambda)b^k$. All of this is more difficult notationally than conceptually, see Table 7.1 for examples of how the various terms relate to one another. Now to find the optimal λ we want the value that maximizes the total codelength. The codelength (and its derivatives) of the data subset corresponding to a test $w(C^k) = 1$ is given by:

$$L' = -\sum_{k:w(C^k)=1} \log(\lambda a^k + (2-\lambda)b^k) \tag{7.2}$$

$$\frac{dL'}{d\lambda} = -\sum_{k:w(C^k)=1} \frac{a^k - b^k}{2b^k + \lambda(a^k - b^k)} \tag{7.3}$$

$$\frac{d^2L'}{d\lambda^2} = \sum_{k:w(C^k)=1} \frac{(a^k - b^k)^2}{(2b^k + \lambda(a^k - b^k))^2} \tag{7.4}$$

The second derivative is clearly always positive and so we can find a root of the first derivative (which is the codelength minimum) using a binary search in λ.

Table 7.1: Examples of relation between CDF and variables A, B, a, b, X

CDF	A^k, B^k	a^k, b^k : X=true	a^k, b^k : X=false
μ^k = 0,.3,1	.3, 0	.3, 0	.2, .5
μ^k = 0,.9,1	.5, .4	.5, .4	0, .1
μ^k = 0,.6,1	.5, .1	.5, .1	0, .4
μ^k = 0,.5,1	.5, 0	.5, 0	0, .5

7.3 Experimental Results

We tested the performance of the proposed algorithm in comparison to AdaBoost.M1 on 25 of the datasets from the UCI machine learning repository [Asuncion & Newman (2007)]. We chose the datasets as follows. We started from the set of 33 datasets discussed by [Dietterich (2000)]. Of these, we dropped the datasets containing less than 500 samples. This gave 14 data sets. We also added the following newer datasets from the repository: “Abalone”, “Adult”, “Contraceptive”, “Cover”, “Magic”, “Mushroom”, “Spam”, “Yeast”, “Connect4”, “Car”, and “Flare”.

As the proposed algorithm currently only works for binary classification problems, the datasets with multiple class labels were modified to work as binary classification problems. For several datasets there was one class much larger than the others; we defined the binary problem to be predicting this class. When there was no such class,

Table 7.2: Binarization modifications to non-binary UCI datasets. Domains not listed were already formulated as binary prediction problems.

Domain	Binarization
Abalone	predict ring value < 9
Annealing	predict class #3
Splice	predict {"IE","EI"}
Segmentation	predict {"brickface","cement", "path"}
Satimage	predict "grey soil" classes
Krk	predict white win in < 13 moves
Vehicle	predict {"bus","van"}
Shuttle	predict "Rad Flow"
Waveform	predict class 1
Contraceptive	predict "No-use"
Cover	predict "Lodgepole Pine"
Yeast	predict "CYT"
Connect4	predict "win"
Car	predict "unacc"
Flare	predict > 1 class C flare.

we attempted to package similar classes together. These "binarization" modifications are described in Table 7.2.

For the datasets with prespecified partition of training and test data ("Annealing", "Adult", "Shuttle", "Satimage") we used the given partition. For the other datasets without such a partition, we used 10-fold cross validation. The only exception is the huge dataset "Cover", which we split into training and test sets of 100,000 samples each.

An important part of boosting is the selection of the set of weak predictors. The following function bank was used in these experiments. For numeric attributes, a set of 200 threshold points were chosen equally spaced between the minimum and maximum values of the attribute on the training data. An associated weak hypothesis was defined that returns true if the attribute is below the threshold or missing. For nominal attributes, a single function was defined for each possible value that returns true if the

attribute is not missing and equal to the specified value. The logical inverse of the above functions were also used. This choice for the weak classifier library is similar to "FindAttRule" described in [Freund & Schapire (1996)].

In addition to defining the library of weak predictors, the critical parameter that must be chosen for boosting algorithms is the number of rounds T. If too small a choice is made for T, one might not achieve a sufficiently powerful classifier; if too large a choice is made then overfitting will occur. AdaBoost seems to be more resistant than other algorithms to the problem of overfitting due to its ability to increase the margin between training examples [Schapire *et al.* (1998)], but overfitting is still a problem.

The T parameter was chosen dynamically using a "held-out" data scheme as follows. First, 10% of the training data was separated for use as a held out set. Then 1000 rounds of boosting were run for both algorithms, using the remaining 90% of the data. The held out prediction error $E_H(T)$ as a function of training round for both algorithms over the course of the learning. Then, the T^* value was calculated as follows:

$$T^* = \arg\min_T E_H(T) \tag{7.5}$$

In the case of a tie, the lower T value was selected. The trained classifier was rolled back from $T = 1000$ to $T = T^*$, and used to predict the test data. Table 7.3 shows the distribution of T^* choices made for the set of 205 total trials (10 fold cross validation was used for 20 domains; the remaining 5 domains used a single training/test run, $205 = 10 * 20 + 5$).

Table 7.4 shows the results of the comparison. We see that for many domains the algorithms show similar performance, but there are several domains where strong differences appear. Perhaps the most striking outcome is one the dataset "Krkp", where FundaBoost achieves 68 errors compared to AdaBoost's 126, a reduction of nearly 50%. FundaBoost achieves an even better relative performance improvement on the dataset "Shuttle", though in this domain both error rates are very small in absolute terms. FundaBoost also wins convincingly on the "Car" dataset. On the other side, AdaBoost wins strongly on the "Vehicle" and the "Segment" datasets. Another interesting point is that FundaBoost wins on all of the top six largest datasets.

Table 7.3: Distribution of T^* selections over 205 trials.

	FUNDABOOST	ADABOOST
$T^* \in [0, 200)$	174	167
$T^* \in [200, 400)$	20	21
$T^* \in [400, 600)$	7	7
$T^* \in [600, 800)$	4	3
$T^* \in [800, 1000)$	0	7

7.4 Summary

This chapter discussed a modification of the PITA algorithm that could be used to attack the binary pattern classification problem. When the abstract PITA algorithm was instantiated to solve this problem, the result is recognizable as a boosting algorithm. Here, the context function battery $\mathcal{W}$ plays the same role as the weak predictors of boosting.

The main problem that occurs in the case of binary pattern classification is that the Bin Overlap problem becomes very severe. This is because there are only two outcomes, so each CDF outcome window is very wide and thus very likely to overlap a bin boundary. For this reason, the "pure" version of PITA, even when using the fine-graining scheme, does not provide competitive performance. The FundaBoost algorithm used in this chapter is a modified version of PITA, in which the histogram KLD score is used as a heuristic rule to select a set of candidate context functions. After selecting the best candidates using the histogram KLD score, a brute-force method is used to compute the optimal parameters and exact codelength savings score.

The FundaBoost algorithm achieves performance that is competitive with the well-known AdaBoost algorithm, on the set of problems investigated in this research. On some domains FundaBoost does substantially better than AdaBoost, on other domains it is substantially worse. One noteworthy trend was that FundaBoost won consistently on the larger domains in the benchmark. Of course, these results need to be supported by more extensive empirical studies on other data sets, and using different kinds of context functions.

Table 7.4: Performance comparison of AdaBoost and FundaBoost, using the T^* selection scheme. The table shows the number of errors made on the held-out training data (E_H) and on the test data (E_T) after round T^*.

		FUNDABOOST		ADABOOST	
Domain	# test	E_H	E_T	E_H	E_T
Annealing	100	3	8	4	7
CreditA	690	72	99	67	92
BreastW	690	26	40	16	32
Vehicle	840	50	88	26	53
CreditG	1000	184	260	182	254
Flare	1060	159	185	154	174
Cont	1470	311	431	314	424
Yeast	1480	300	390	306	421
Car	1720	51	63	60	97
Satimage	2000	15	58	12	59
Segment	2310	37	56	13	31
Sick	2800	36	66	36	73
Hypo	3160	44	42	39	44
Splice	3190	169	181	158	196
Krkp	3190	55	68	92	126
Abalone	4170	722	895	710	888
Spam	4600	213	287	181	281
Waveform	5000	581	599	523	579
Mushroom	8120	0	2	0	2
Shuttle	14500	3	8	3	18
Adult	15060	421	2122	427	2137
Magic	19020	2423	2761	2511	2840
Krk	28050	4346	4903	4466	5055
Connect4	67550	12634	13979	13017	14383
Cover	100000	2178	21621	2254	22073

Chapter 8

PIT Analysis for Word Modeling

8.1 Introduction

This chapter describes how the PITA algorithm can be applied to the problem of modeling word morphology. Word morphology is the study of how words are constructed from letters. In English, some letter sequences are more likely than others. For example, the sequence "thr" is common, whereas "qix" is rare. Also, whenever the prefix "thr" is seen, the next letter is almost always a vowel. By writing down a set of context functions $\mathcal{W}$ that refer to letter sequences, it is possible to construct a model of the probability of a word using the PITA algorithm. This model can then be used for various purposes, such as text compression and recognizing whether a rare word (such as "mesonoxian"), not seen in the original corpus, is a real English word.

In addition to a statistical model $Q(x|c)$, the PITA algorithm also chooses a set of optimal context functions w_t^*. These features can be useful for other purposes. For example, in English it is very common for a word to end with the letters "-ing". Since this feature is very informative about the letter sequence, it will probably be chosen by PITA (or another good feature selection algorithm). However, the "-ing" feature is also very informative about the part of speech of a word. Specifically, if this feature is observed, it is very likely that the word is a gerund. Similar observations can be made about other suffixes such as "-ly" and "-ed". Thus, the features selected by PITA for the purpose of modeling letter sequences, can be reused for the purpose of part-of-speech tagging, an important task in natural language processing [Manning & Schutze (2002)].

Using raw words as a data source is attractive for two reasons. First, labels are unnecessary. Second, it is easy to obtain large quantities of data. Given a corpus of 10^6 words, with an average length of five letters, the "baseline" information content is $10^6 * 5 * \log_2(26) \approx 2.4 * 10^8$ bits. This large quantity of data is important from an MDL perspective [Rissanen (1978)], because it justifies the use of a complex model. In the application described below, each PITA model ensemble update requires about 42 bits to encode. When the total size of the data is on the order of 10^8, it is quite easy to find context functions that save more than 42 bits.

The application discussed in this chapter is different from the others described in the thesis because the outcomes are non-numeric. This means that a single PDF will correspond to many CDFs, depending on the outcome ordering. This is problematic because PITA depends totally on the use of CDFs. This obstacle is overcome using the outcome reordering scheme of Section 3.9, as discussed below.

8.2 Adaptation of PITA algorithm

The goal of the research of this chapter is to obtain a model for the probability of a word. An n-letter word is treated as a $n+1$ length letter sequence $\{X_1, X_2 \ldots X_{n+1}\}$. The extra outcome is due to the fact that a special letter '] ' is added to indicate the end of a word. Thus the word "thought" is represented as a series of outcomes $\{X_1 = \text{t}, X_2 = \text{h}, X_3 = \text{o}, \ldots X_7 = \text{t}, X_8 =]\}$. The probability of a word is then calculated as a product of a series of conditional probabilities for each letter:

$$
\begin{aligned}
P(X_1, X_2, X_3 \ldots X_{n+1}) &= P(X_{n+1}|X_1, X_2 \ldots X_n) P(X_1, X_2 \ldots X_n) && (8.1)\\
&= \prod_{i=1}^{n+1} P(X_i|X_1 \ldots X_{i-1}) && (8.2)\\
&= \prod_{i=1}^{n+1} P(X_i|C_i) && (8.3)
\end{aligned}
$$

Thus, the PITA algorithm will be used to find approximations $Q(x|c) \approx P(x|c)$ where x is a letter outcome, and c is the history of previous letters. As usual, an important problem will be to find a good set of context functions $\mathcal{W}$ that reveal the

relationship between each letter and the preceding letters. Section 8.2.2 presents the choice of context functions $\mathcal{W}$ used in this chapter.

This chapter demonstrates the use of several refinements to the basic PITA algorithm. As noted above, the outcomes in this application are non-numeric. The outcome reordering scheme of Section 3.9 is used to deal with this problem. The variable width histogram technique discussed in Section 3.10, as well as the fine-graining scheme of Section 3.11.1, are also used in this chapter. The technical details related to these methods are discussed in the following sections, and the experimental results indicate that they work.

8.2.1 Outcome Reorderings

PITA depends completely on the use of CDFs. The mapping from a PDF to a CDF depends on the outcome ordering. If there are many equally valid outcome orderings, there are many valid CDFs corresponding to the same PDF. This is not a problem when the outcomes are numeric, since in that case the ordering is natural. But if the outcomes are nominal (non-numeric), then the choice of an ordering can substantially impact the ability of PITA to detect randomness deficiencies.

Consider the following example. There is a large census database which records information about people such as their age, education level, profession, marital status, and income. In the first case, the goal is to predict the person's income from the other attributes. Obviously, income is a numeric outcome. Consider a context function w that fires if the person has completed a college degree. Since having a college degree is associated with a higher income, we expect that the PIT histogram corresponding to $w = 1$ will be skewed to the right of the $[0, 1]$ interval (high incomes), while the $w = 0$ histogram will be skewed to the left (low incomes).

Now consider what happens when we try to predict the "profession" outcome instead of the "income" outcome. Profession is a nominal (non-numeric) outcome with values such as "engineer", "lawyer", "plumber", etc. In contrast to the case of income, there is no obvious way to order the outcomes. Now, there are many professions for which a college degree is essential. So the college degree context function w should be very useful to predict profession. But unless the outcomes are specifically ordered to reflect their dependence on education (i.e. "professor", "lawyer", "doctor" on one side

of the ordering, "plumber", "construction worker", and "farmer" on the other), then the histograms associated with w will not necessarily be uneven.

The same basic problem occurs when the outcomes are letters. The obvious ordering of Latin letters is alphabetic ("a,b,c,... y,z"). But this is not a very useful ordering, since the letters are not grouped together by their phonological function. For our purposes it would be better if, for example, all the vowels were grouped together. This is because there are many cases where observing a certain prefix (such as "thr" or "gr") makes it highly likely that the next letter will be a vowel. Other prefixes indicate that the next letter will be a consonant, or that the word is about to end.

To deal with this problem, we simply define an extended context function $w' = \{w, O\}$ where w is a basic context function, and $O = \{o_1, o_2 \ldots o_{27}\}$ is an letter outcome ordering. There are many possible ways to order the letter outcomes, but we use a simple scheme based on a letter set S. The ordering $O = O(S)$ simply moves the letters contained in S to the head of the outcome ordering. As explained in Section 3.9, the PIT values and model CDF updates can easily be redefined based on an alternate outcome ordering, without changing anything else about the way PITA works.

8.2.2 Context Functions

We now define the basic context functions w, which are used in conjunction with the outcome orderings O to build "extended" context functions $w' = \{w, O\}$. We used three sets of context functions, as shown in Table 8.2. Each type of function is parametrized by a set S of letters, and by a number L which indicates how far back in the history to search against. The notation $\{X_{-L}, \ldots X_{-1}\}$ indicates the history up to a position L letters before the current outcome. The following list shows some examples of the meaning of the context functions in terms of the parameters L and S. Section 8.2.4 describes the parameter sets used for L and S.

- PREFIX-UNIV, $L = 2$, $\in$, S ="a,e,i,o,u" : true if the two previous letters are vowels. Examples: position 'v' in "believe", 't' in "eat".

- PREFIX-EXIST, $L = 3$, $\notin$, S ="a,e,i,o,u,y" : true if none of the previous 3 letters are vowels (including 'y'). Examples: positions 's,t,r,e' in "stream", positions

't,h' in "thought". This test is very powerful because all words in English must include a vowel.

- POSITIONAL, $L = 1$, $\in$, S ="p,b,t,d,k,g" : true if the previous letter is a Stop Consonant. Examples: 'r' in "proud", 'i' in "king". It is rare for two different stop consonants to occur together (combinations such as "dk", "bg", "gp" occur very rarely).

8.2.3 Variable-Width Histograms

As explained in Section 3.10, a PIT histogram is not guaranteed to reveal a randomness deficiency in a subset of PIT values, even if one exists. It is possible for a PIT histogram to be completely flat, even if the PIT values have a very strong randomness deficiency. For example, consider a PIT distribution where 49% of the values lie in the $[0, 1/4]$ range, 49% lie in the $[3/4, 1]$ range, and 1% lie in both the $[1/4, 1/2]$ and the $[1/2, 3/4]$ ranges. If this distribution is viewed with an evenly spaced 4-bin histogram, a savings of .86 bits per outcome can be achieved. However, if viewed using a standard 2-bin histogram, then the distribution will appear to be completely flat.

In this chapter, PIT logging was performed using 32-bin histograms. To obtain the model update, the 32-bin histograms were collapsed into 2-bin variable width histograms. This method is explained in Section 3.10.

8.2.4 Size of Context Function Battery and Memory Requirements

We now analyze the size of the battery of extended context functions $\mathcal{W}'$. Each extended context function is a combination of a basic context function and an outcome ordering: $w' = \{w, O\}$. Every basic context function is paired with every outcome ordering, so the size of the extended battery is $|\mathcal{W}'| = |\mathcal{W}| \cdot |\mathcal{O}|$.

Both the basic context functions and the outcome orderings depend on letter sets S. The letter sets were defined used some very rough knowledge of linguistic phonology, as shown in Table 8.1. In addition to the 8 sets shown in the table, we also use 27 single-letter sets, corresponding to the 26 letters of the Latin alphabet plus the word-end character. So the total number of letter sets is 35, and the number of outcome orderings is $|\mathcal{O}| = 35$.

The number of basic context functions w can be calculated as follows. There are three types of function PREFIX-UNIV, PREFIX-EXIST, and POSITIONAL. Each of these depends on a parameter L, a parameter S, and a choice of $\in$ vs. $\notin$. There are five choices for L (1,2,3,4,5). So the total size of the basic package is $|\mathcal{W}| = 3 * 2 * 5 * 35 = 1,050$, and the size of the extended package is $|\mathcal{W}'| = 1,050 \cdot 35 = 36,750$.

Finally, to implement the variable-width histogram idea, each extended context function w' logs 32 histogram bin counts. The total number of counts required is $32 \cdot 1,050 = 1,176,000$. These counts are the major determining factor in the total memory requirements for the algorithm.

It is worth analyzing the complexity of the PITA model from an MDL perspective. Specifying a test costs $\log_2(|\mathcal{W}'|) \approx 20.2$ bits. Recall that we are logging using 32 bin histograms, but performing model updates based on variable width 2-bin histograms. To specify the variable bin width requires $\log_2(31) \approx 4.95$ bits. The expansion factor can be encoded with sufficient precision using 16 bits. Thus, the total number of bits required for a single model update (one PITA round) is about 42 bits.

Table 8.1: Letter sets S used in defining the statistical tests.

Vowels	a,e,i,o,u
Vowels + Y	a,e,i,o,u,y
Stop Consonant	p,b,t,d,k,g
Nasal Consonant	m,n
Fricative Consonant	s,z,c,h
Glide Consonant	l,r,j,w,h
Unvoiced Consonant	p,t,k,f,s
Voiced Consonant	b,d,g,m,n,v,z,w

8.3 Data Sources and Preprocessing

For our datasets, we use a collection of novels downloaded from the internet. For the training data, we used the novel "Little Women" by Louisa May Alcott. For the test data, we used novels "Huckleberry Finn" by Mark Twain, "Democracy in America" by

Table 8.2: Three types of context functions used in the word morphology experiment.

PREFIX-UNIV	$\forall\{X_{-L}, \ldots X_{-1}\} \in S$	$\forall\{X_{-L}, \ldots X_{-1}\} \notin S$
PREFIX-EXIST	$\exists\{X_{-L}, \ldots X_{-1}\} \in S$	$\exists\{X_{-L}, \ldots X_{-1}\} \notin S$
POSITIONAL	$X_{-L} \in S$	$X_{-L} \notin S$

Table 8.3: Book information. The book "little" was used as training data.

book	unique words	total words	total outcomes
little	8638	140250	707390
huckfinn	4583	81108	380410
democracy	8190	157750	869210
anna	10192	262660	1337600
mutual	12123	231660	1175900

Alexis de Tocqueville (English translation), "Anna Karenina" by Leo Tolstoy (English translation), and "Our Mutual Friend" by Charles Dickens[1]. Some statistics about the texts are shown in Table 8.3.

The following preprocessing was used to obtain the raw words from the electronic text of the novels. First, we split the text into tokens by splitting whenever a white space character was found. Next, all tokens that contained punctuation or capital letters were removed. This left us with a list of common words corresponding primarily to real words that exist in the English language. We also maintain counts of the words, and factor these counts into our analysis (thus, the model attempts to assign shorter codes to words that appear often). As noted above, we also append a special word-end letter '] ' so that the words are self-delimiting: no extra word-length variable is necessary.

In this application we use bigrams for the initial models over which the PITA updates are layered. The parameters for the bigram models are estimated by simple counting on the training data. The counts were initialized to 1 for each bigram letter pair to prevent the model from assigning zero probability to a pair not observed in the training data.

[1] All of these texts can be found online at www.gutenberg.org.

8.4 Experimental Results

The PITA algorithm was run on the training data (novel "little") using the set of context functions described above. Approximately 2200 PITA rounds were performed before the algorithm terminated due to a lack of further progress.

Figure 8.1(a) shows the progress made by PITA on the training data in terms of codelength savings per round. As can be seen, the PITA algorithm achieves substantial reduction in the codelength required to express the data set (recall that a reduction in codelength is equivalent to an increase in the log likelihood). Before PITA modeling is applied, the total codelength of the data set is $2.37 \cdot 10^6$. After 2200 PITA rounds, the codelength has been reduced to $1.78 \cdot 10^6$, for a net savings of $5.9 \cdot 10^5$, or about 25%. Note that the base codelength of $2.37 \cdot 10^6$ is achieved using the bigram models.

Figure 8.1(a) shows two similar curves. The solid blue curve shows the codelength savings achieved on the "real" data $\{X^k\}$. The green dotted curve describes the codelength savings achieved on the fine-grained data outcomes $Z^k = \{X^k, Y^k\}$. Evidently, the codelength savings for the original data set is strictly larger than the codelength savings for the fine-grain data set. This confirms the intuition that all savings reported on the fine-grain data $\{Z^k\}$ actually comes from the real data $\{X^k\}$ and not from the auxiliary data set $\{Y^k\}$.

Figure 8.1(b) shows the discrepancy between the savings on the fine grain dataset $\{Z^k\}$ and the savings on the real data set $\{X^k\}$ for each round. The curve fluctuates noisily around zero, indicating that in some rounds the predicted fine-grain savings are greater than the real achieved savings, and sometimes the opposite is true. On average, the fine-grain score underpredicts the real savings, but only slightly: the average discrepancy is 17.62 bits.

Figure 8.1(c) illustrates the generalization ability of the PITA model. The graph shows the codelength in bits per outcome for each book, for both the training data ("little") and the test data. If overfitting were a problem, the curves corresponding to the test data would go down at the beginning, and then bottom out and start rising again. As can be seen, this does not happen: the codelength for the test data decreases monotonically as the number of PITA layers increases.

Further evidence of the generalization ability of the PITA model is shown in Figure 8.1(d). The blue dotted curve in the graph corresponds to a set of nonsense words.

These words are random sequences of letters generated by sampling from the bigram models (some examples of these generated words are shown in Section 8.4.1 below). As the graph shows, as the training proceeds, the PITA model assigns higher and higher codelength to the nonsense words. The green solid curve shows the codelength required for a set of English words that were not part of the training data. As can be seen, the green curve decreases monotonically with the number of PITA rounds, indicating again that overfitting is not a problem. Note that the increasingly wide gap in the codelength required for nonsense words versus real English words indicates that the model can be used to discriminate between the two.

8.4.1 Sampled Words

One way to assess the quality of a model is to sample from it, and then compare the sampled data to the real data. If the model is good, the sample data should "look like" the real data. In the present case, we can judge the quality of the model by examining the similarity between the generated words and real English words.

To generate a word, one starts by sampling from the initial model distribution $Q(x|\emptyset)$. The resulting outcome then becomes the context for the next step. Thus if the outcome from the first sample is 'd', then on the second step we sample from $Q(x|c =$"d"). This is continued until the special word-ending character is selected.

The following words are generated by sampling from the bigram model:

> a abangivesery ad allars ambed amyorsagichou an and anendouathin anth ar as at ate atompasey averean cath ce d dea dr e ed eeaind eld enerd ens er evedof fod fre g gand gho gisponeshe greastoreta har has haspy he heico ho ig iginse ill ilyo in ind io is ite iter itwat ju k le lene lilollind lliche llkee ly mang me mee mpichmm n nd nder ng ngobou nif nl noved o ond onghe oounin oreengst otaserethe oua ptrathe r rd re reed reroved sern sinttlof suikngmm t tato tcho te th the toungsshes ver wit y ythe

There are several problems with the above word list. One is that there are too many long words: words of length 10, 11, or above are common. At the same time, there are several single letter words like "r" and "k". These words violate a basic rule of English morphology, which is that all words must have a vowel. Another problem is that several

words have long sequences of consonants such as “mpichmm” and “suikngmm”. Long sequences of consonants such as these words contain are rare in English.

The following words are generated by sampling from the PITA+bigram models:

> a ally anctyough and andsaid anot as aslatay astect be beeany been bott bout but camed chave comuperain deas dook ed eveny fel filear firgut for fromed gat gin give givesed got ha hard he hef her heree hilpte hoce hof ierty imber in it jor like lo lome lost mader mare mise moread od of om ome onertelf our out over owd pass put qu rown says seectusier seeked she shim so soomereand sse such tail the thingse tite to tor tre tro uf ughe umily upeeperlyses upoid was wat we were wers whith wird wirt with wor

This list appears to be a substantial improvement over the bigram-only models. The length of the words is about right, there are no single-consonant words, and there are no words with long consonant sequences. The list includes many actual English words such as “like”, “lost”, and “mare”. Many of the other words are very close to real English, such as “bott”, “mader”, “rown”, and “seeked” (which would be the past tense of “to seek” if not for the irregular form “sought”). Of course, the PITA model is not perfect. For example, the word “beeany” contains the sequence “eea”, which is very rare in English. Some of the long words such as “upeeperlyses” and “soomereand” seem very strange.

8.5 Summary

This chapter discussed an application of the PITA algorithm to the problem of modeling word morphology. The application mostly followed the standard PITA recipe: find a database, define the outcomes and contexts, write down a set of context functions, and go. The application made use of the layering property of PITA by using bigrams for the initial models. The results section showed that the algorithm could obtain substantially improved models without overfitting.

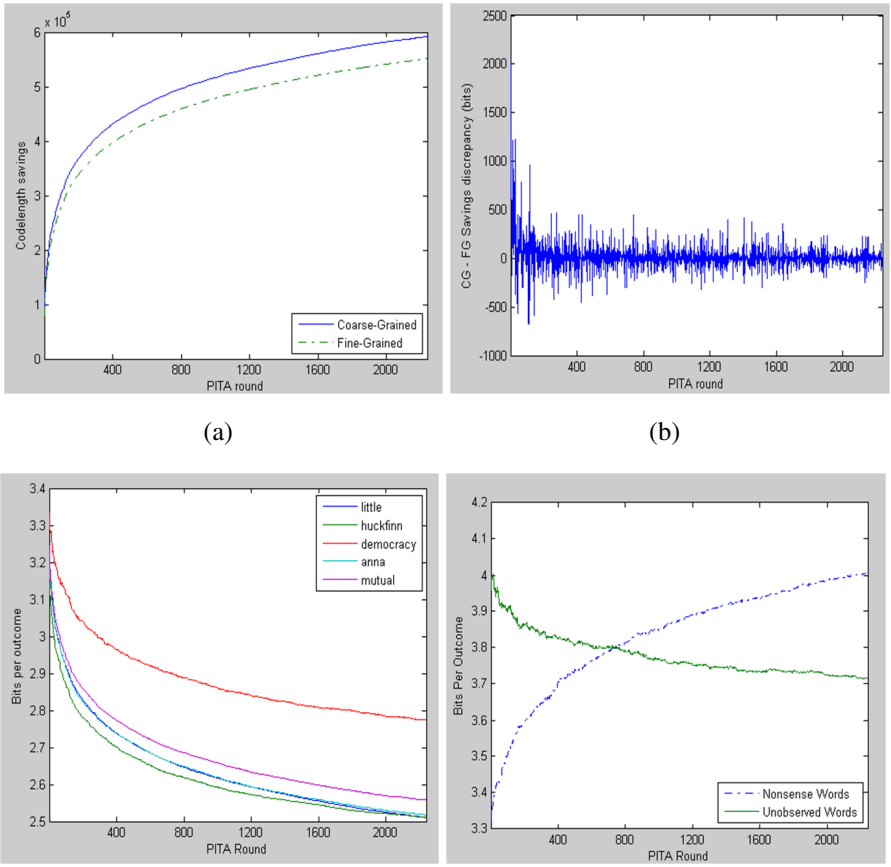
The main points of interest of this chapter are the use of two special techniques: outcome reordering and variable-width PIT histogram updates. The outcome reordering method is necessary to allow the algorithm to work for distributions with non-

numeric outcomes. Some PIT distributions have randomness deficiencies that are invisible when viewed in a standard 2-bin histogram. The variable-width histogram scheme allows the algorithm to search for randomness deficiencies using 32-bin histograms, but avoids excessive complexity by collapsing the large histograms into variable width 2-bin histograms to perform model ensemble updates. The results of the chapter show that these techniques work.

Another important point of this chapter was the confirmation that the fine-graining idea of Section 3.11.1 works. Let us revisit the argument in light of the evidence in Figures 8.1(a) and 8.1(b). The fine-graining idea is to select PIT values based on a set of fine-grained outcomes $Z^k = \{X^k, Y^k\}$ instead of the original set of outcomes $\{X^k\}$. This strongly reduces the effect of the Bin Overlap problem, meaning that the histogram KLD scores are very accurate predictions of the codelength savings that will be achieved on the joint data set Z^k. The argument is that since the set of $\{Y^k\}$ is random, codelength savings predictions made for the $\{Z^k\}$ must really correspond to savings on the $\{X^k\}$ data. The result shown in Figure 8.1(b) shows that while the fine-grained savings prediction is not exact, it is not systematically biased. Figure 8.1(a) lends further support to the argument, by showing that the savings achieved on the $\{Z^k\}$ is nearly equal to the savings achieved on the real data $\{X^k\}$.

This chapter also demonstrated a technique for qualitative model assessment. This was to sample from the model, and then compare the sampled data to real data. In this case the real data are English words, and the sampled data are invented words like “astect”, “rown”, and “beeany”. The fact that the PITA+bigram model produces generated words that are quite similar to real English words is a further confirmation of the quality of the model.

An interesting point, more strongly emphasized in Chapter 9, is that the PITA models are quite complex, but are justified by the codelength savings they achieve. As shown in Figure 8.1(a), the PITA model saves about $5.9 \cdot 10^5$ bits on the training data, compared to the pure bigram models. In comparison, a conservative estimate of the complexity of the model is about $2200 \cdot 42 \approx 9.2 \cdot 10^4$ bits. This is actually quite large compared to the models used in most machine learning research, where small data sets require the use of minimally complex models. We are able to justify the use of such a complex model because we are attempting to model the raw data (in this case, words) as opposed to a set of labels.



(a) (b)

(c) (d)

Figure 8.1: Figure (a) shows the total codelength savings achieved by PITA for both the coarse-grained outcomes $\{X^k\}$ and fine-grained outcomes $\{Z^k\}$. Figure (b) shows the discrepancy between the codelength savings achieved on the fine-grain data $\{Z^k\}$ and the savings achieved on the real data $\{X^k\}$. Figure (c) shows the codelength in bits per outcome required for each of the books. Figure (d) shows the codelength in bits per outcome required for a batch of nonsense words, compared to a batch of real English words that were not observed in the training data. The x-axis for all four graphs is the number of PITA rounds. See text for discussion.

Chapter 9

PIT Analysis of Speech Data

9.1 Introduction

This chapter discusses the application of the PITA idea to the problem of speech modeling. A speech signal, and sound in general, is just a time series sequence of samples. The problem of speech modeling is just the problem of predicting, or compressing, this time series.

Statistical methods have recently become very popular in the speech processing field [Picone *et al.* (1993); Sohn *et al.* (1999)]. A central goal of this research is to model the relationship between a sound segment and a label, which designates whether or nor the segment contains speech. If this relationship can be learned well, then the model can be used to determine if a newly presented segment contains speech or not. This is called the Voice Activity Detection (VAD) problem. VAD is a close cousin of several well-studied problems in computer vision and machine learning such as face detection and handwritten digit recognition.

Instead of modeling the relationship between a sound segment and a label, the goal of the research in this chapter is to model speech *itself*. This research topic is not as popular as VAD, but there is some previous work on it [Davenport Jr (1952); Gazor & Zhang (2003b); Richards (1964)]. This previous research adopts the perspective of traditional statistics, and assumes that speech can be described well using a simple model. In contrast, our approach is to use complex models to describe speech. We show that when attempting to model raw speech data, complex models can achieve much better performance without overfitting.

An interesting point is made in this chapter about the use and justification of complex models. The main lesson of statistical learning is that, in order to avoid overfitting, one must avoid the use of complex models in low-data regime problems [Rissanen (1978); Vapnik (1998)]. However, when attempting to model raw speech data, an enormous quantity of data is available, justifying the use of highly complex models. The speech database used in this chapter has a baseline information content of more than $4 \cdot 10^8$ bits. We demonstrate that a model requiring about $2 \cdot 10^6$ bits to encode (larger by far than the models used in traditional machine learning applications) can be used to save over $8 \cdot 10^7$ bits when encoding the speech data.

In addition to the VAD problem, another potential application of the research discussed in this chapter is for lossless compression of speech data. This application can be achieved because our method models the entire speech signal. Other methods model an extracted feature vector rather than the speech signal itself, meaning that they cannot be used for lossless compression.

9.2 Background on Speech Processing

One standard approach to the VAD problem involves the use of binary state Hidden Markov Models [Cho & Kondoz (2001); Sohn *et al.* (1999)]. In these models, one state corresponds to the presence of speech and the other corresponds to non-speech. Given a trained HMM and a new sound segment, one runs the Viterbi algorithm, which infers the most probable state trajectory. This state trajectory is then used as the guess about what sections of the sound segment contain speech.

The binary state HMM strategy requires the solution of many technical problems, such as the clipping problem and the hangover problem [Sohn *et al.* (1999)]. These problems basically involve the fact that the HMM states tend to have too much inertia: they resist transitioning from speech to non-speech or vice versa. Another important question is how to perform online noise parameter estimation, so that the VAD system can adapt to the noise present within a single episode [Fujimoto & Ishizuka (2008); Fujimoto & Nakamura (2005)] If the noise parameters can be successfully estimated, subsequent detection rates are greatly enhanced.

Another critical requirement for successful HMM-based VAD is to find good observation probability models $Q_S(O)$ and $Q_N(O)$. These are estimates of the probability

of sound observation O in the speech state (S) and the noise state (N). Note that if very good models $Q_S(O)$ and $Q_N(O)$ could be found, they would constitute a complete solution to the VAD problem. However, obtaining good observation models is a difficult problem.

The dominant strategy for finding $Q_S(O)$ and $Q_N(O)$ is to search for good speech features[1]. Instead of modeling the probability of the full sound segment O, we apply some feature function F and model the probability of the result. Thus the models are now of the form $Q(F(O))$. Some commonly used feature functions are the fundamental frequency, the power, and the cepstrum transform [Picone *et al.* (1993)]. The motivating belief is that if good features can be found, then the models $Q_S(F(O))$ and $Q_N(F(O))$ will become very simple and distinct. In other words, a value of a good speech feature function should be very large when speech is active, and very small otherwise. Thus, most research builds the observation models by combining "smart" features with simple statistical models such as the multivariate Gaussian [Ishizuka & Kato (2006); Nemer *et al.* (2001)].

The work of [Gazor & Zhang (2003b)] takes the opposite approach. These authors searched for "smart" statistical models, and modeled the sound observations O directly (i.e. no feature functions were used). They found that Laplacian models were a much better fit to the speech data than the often-used Gaussian models. In later work, they connected their improved Laplacian speech model to an HMM-based VAD system, and showed that the improved model achieved significantly increased detection rates [Gazor & Zhang (2003a)].

In this research, we propose to take this idea a step further. Like [Gazor & Zhang (2003b)], we attempt to model sound observations O directly, without using any feature functions. Instead, we attempt to obtain good models $Q_S(O)$ and $Q_N(O)$ by constructing complex models using the PITA algorithm. We see no reason to believe that speech follows any kind of simple distribution such as the Laplacian or Gaussian. Since a large amount of data is available, a complex model can be used without risk of overfitting. This belief is vindicated by our findings, which show that the complex PITA models provide substantially better performance than the optimal Laplacians, and this improved performance generalizes to the test data.

[1]Note that the use of the word "feature" in speech processing is slightly different from the use in this thesis. In this document, "feature" is just another word for "context function".

9.3 Adapting the PITA Algorithm for Speech Modeling

The goal of this work is to obtain strong models $Q_S(O)$, the probability of an sound observation O given that speech is occurring. Each sound episode is broken up into non-overlapping windows containing D samples. The Discrete Cosine Transform is then applied to the window, resulting in D DCT coefficients. These coefficients become the X data used in the PITA algorithm. Thus the sound window O_t is transformed into a vector X_t within elements X_t^d. A simple process is used to convert X_t^d into X^k data.

The contexts C required for the PITA algorithm are simply the histories of the sound data at previous time steps: $C_t = \{X_{t-1}, X_{t-2} \ldots\}$. The dependence of the X on the context C^k will be mediated by the set of context functions $\mathcal{W}$ discussed below.

An important point is that we actually construct D separate PITA models, one corresponding to each DCT coefficient. This means that there are D versions of the PITA algorithm running in parallel. Note that one round of ϕ-statistics logging for all D processes can be done using a single database traversal. So running T learning rounds for D models, requires only T database traversals, and not $D \cdot T$. The training process for each coefficient model draws from the same set of context functions $\mathcal{W}$, but may make different choices for the optimal tests w_t^*. Note that the context information used by the dth model is not restricted to $X_{t-1}^d, X_{t-2}^d \ldots$. For example, the model for $d = 5$ may use context functions that refer to X_{t-1}^{48}.

The statistical tests $\mathcal{W}$ used in this work are shown below. While in principle these functions can depend on the history in arbitrary ways, in this work we used functions that depend only on the previous time window X_{t-1}. Several of these tests are very simple functions of the DCT coefficients in the previous time window. We also used a set of Mel-frequency Cepstral Coefficients and the related audio spectrum and power spectrum features. These features were generated using the "melfcc" command of the RastaMat toolbox for Matlab distributed by Ellis (2005), using 24 Cepstrum coefficients and 24 power bands. These were calculated using non-overlapping windows. The context functions, with the associated parameters, are shown below:

- ALLSMALL(T): true if $|X_{t-1}^d| < T$ for all d.
- PREVCOEF(d, T_{min}, T_{max}): true if $T_{min} < X_{t-1}^d < T_{max}$.

- CEPSTRA:(N, T): true if the Nth Cepstra coefficient was greater than T.
- AUDIO SPECTRUM(N, T): true if the Nth audio spectrum coefficient was greater than T.
- POWER SPECTRUM(N, T): true if the Nth power spectrum coefficient was greater than T.

The various threshold parameters T, T_{min}, T_{max}, for the tests were chosen to roughly span the response range of the various functions on the training data. The functions in the above list are all binary context functions. These were used in combination with the 4-bin PIT histograms. In addition to these, a Shifted Prediction Laplacian Transformation of Chapter 4 was used, based on the following simple predictor::

$$f(C_t) = \frac{X^d_{t-1} - X_{min}}{X_{max} - X_{min}} \tag{9.1}$$

Where X_{min}, X_{max} are the minimum and maximum values that the DCT coefficients are allowed to take.

As noted above, we used double Laplacians as the initial models in the PITA algorithm. The double Laplacian depends on two parameters, the mean μ and the scale parameter b. Optimal values for these parameters were calculated from the training data. Separate μ_d, b_d parameters were used for each model Q^d_S.

9.4 Experimental Results

9.4.1 Data Sources and Pre-processing

For training data, we used 1000 files from the training section of the TIMIT database [Fisher *et al.* (1986)]. The stream of samples was truncated so that its length would be equal to an integral multiple of 160. The stream was then partitioned into 10 millisecond windows, each of which contained 160 samples. The discrete cosine transform was applied to each of these windows, and coefficients C such that $|C| > C_{max} = 2 \cdot 10^5$ were reduced such that $|C| = C_{max}$. The test data was 100 files from the test section of TIMIT, preprocessed in the same way. For evaluation purposes we also use a package

of noise data. This was street noise, segmented out of the CENSREC-1-C database[1]. The noise data was transformed into DCT coefficients and scaled so that the each DCT coefficient had the same variance as the speech training data.

9.4.2 Codelength Reduction

Figure 9.1 shows the progress achieved by the training phase, measured by the codelength savings achieved as a function of the number of PITA rounds. As is typical of learning curves the start of the process shows rapid progress, after which improvement starts to slow. Note that the word "codelength" refers to the negative log-likelihood of the data given the model:

$$\sum_k -\log_2 Q_S(X^k|C^k) \tag{9.2}$$

So that a codelength savings is equivalent to achieving an increase in the log-likelihood of the data given the model. Also, the savings shown in Figure 9.1 are measured relative to the initial double Laplacian models. Thus the PITA-enhanced models achieve substantially improved performance compared to the Laplacian.

Figure 9.2 illustrates the generalization power of the model obtained by the training algorithm. The graph shows the codelength progress of the unseen test data as a function of the number of PITA rounds. As can be seen, the codelength of the test data decreases monotonically as the layer index increases. This demonstrates that the model has not overfit the data. If overfitting were a problem, the codelength of the test data would decrease at the outset, but then reverse course and begin to increase. The fact that this does not occur indicates that overfitting is not a problem here. Note that Figure 9.1 shows codelength savings, while Figure 9.2 shows total codelength.

Figure 9.2 also charts the codelength assigned by the model to the package of noise data. The large gap between the codelength of the noise data and the speech data implies that the model can be used in conjunction with a likelihood ratio test to discriminate between noise and speech. Oddly, the codelength of the noise data peaks around the test index #10, and then decreases slightly after that.

[1] http://sp.shinshu-u.ac.jp/CENSREC/

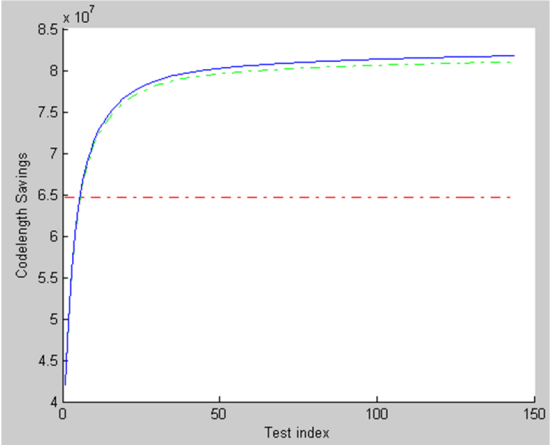


Figure 9.1: Codelength progress achieved by the learning algorithm on the training data. The dotted horizontal line at about $6.5 \cdot 10^7$ marks a reduction of 15% compared to the starting codelength achieved by optimal double Laplacian models. The green dotted line shows the savings achieved on the fine-grain data set $\mathcal{Z}$, while the blue line shows the savings on the original data $\mathcal{X}$.

The fine-graining trick of Section 3.11.1 is also used, and Figure 9.1 also shows a comparison of the codelength savings achieved on the fine-grained data set $\mathcal{Z}$ and the original data $\mathcal{X}$. As can be seen, the two curves are very close to one another.

9.5 Summary

This chapter described how the PITA algorithm could be used to model speech data. The speech modeling application is very well-suited to PITA for several reasons. First, the number of data outcomes is quite large ($X_{max} - X_{min} = 40000$). This means that the CDF outcome windows are very thin, so the Bin Overlap problem is quite unimportant. Second, the database is very large, and the data is quite "rich", so that complex models are useful, and can be justified without overfitting. Furthermore, the large size of the dataset means that for most algorithms, the training process will require huge amounts of computational power (this is particularly true for Maximum Entropy models, see Chapter 5).

Regarding the justification of complex models, consider the following MDL style cost-savings analysis of the PITA model. Note the large scale of the Y-axis in Fig-

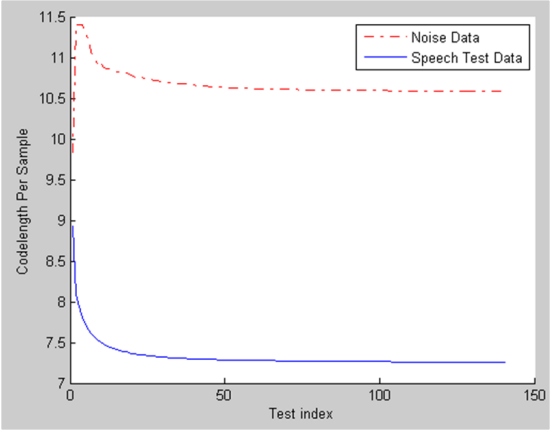


Figure 9.2: Plot showing the changes in codelength produced by the model as a function of the number of PITA rounds. The codelength of the unseen test data declines monotonically, indicating that overfitting is not a problem.

ure 9.1. At the right end of the graph, the model has achieved savings of more than $8 \cdot 10^7$ bits. A very conservative estimate of the cost of the model is as follows. In each training round, a new context function is added to each coefficient model. Thus when the training completes after round 145, the algorithm has added a total of $145 \cdot 160 = 23,200$ context functions w_t^*. The information in each function consists of a test index and 3 histogram expansion parameters. Assume that the expansion parameters cost 32 bits a piece; the test index costs $\log_2 |\mathcal{W}| \approx 10$. Then the total cost of the model is about $23200 * (3 * 32 + 10) \approx 2.5 \cdot 10^6$. Thus, while the model is far more complex than those used in traditional speech modeling research, this complexity is entirely *justified* by the MDL principle.

Another point of interest in this chapter was the empirical evidence justifying the fine-graining trick of Section 3.11.1. In that section, we argued that because the auxiliary data set $\mathcal{Y}$ is random, all savings reported on the joint data set $\mathcal{Z}$ must really correspond to savings on the real data $\mathcal{X}$. Figure 9.1 supports this argument. As can be see in the figure, savings achieved on the fine-grain data set is almost exactly equal to the savings achieved on the real data set. Thus codelength savings predictions made on the basis of the fine-grain data (where the Bin Overlap problem is insignificant) are accurate predictions of the codelength savings made on the real data set.

Chapter 10

PIT Analysis of Motion Capture Data

10.1 Introduction

This chapter describes the application of the PITA algorithm to data obtained using a motion capture system. A motion capture system records a time series description of the joint angle configuration of the user's body as he performs a motion. Since the human body has many joints, and is capable of many kinds of motions, the data obtained in this way can be quite rich and complex.

One important question in the field of motion understanding research is how to recognize or categorize a motion sequence. Here we assume we have a training data set of motions labeled with categories such as "running", "walking", and "jumping". Then the task is to recognize the category of some new unseen motion trajectory.

One major obstacle to success in the motion recognition task, noted by [Shimosaka *et al.* (2004)], is the difficulty of obtaining motion capture data. This process requires substantial effort and time on the part of the human subjects and experimenters. Therefore, when measured by the number of trajectories, the amount of data available is typically quite small (the experimental results section of this chapter involves 245 trajectories). This obstacle motivates research into ways to extract informative features from a limited data set. These features should correspond to human intuition and also achieve good generalization performance [Mori *et al.* (2004)]. Another way of attacking the problem is to build human prior knowledge about motion into the recognition system. The issue here is how to express this qualitative knowledge in terms the recognition system can understand [Shimosaka *et al.* (2004)].

Another major theme in the motion understanding literature is imitation. An important goal here is to allow the user to "program" a robot simply by demonstrating a motion, which the robot then imitates. A key conceptual question is how to transform the observed human-centric motion into a robot-centric motion that the robot can perform. An early work in this area constructed a symbolic description of the observed motion, and then performed a robotic motion based on the symbolic description [Kuniyoshi *et al.* (1994)]. This approach requires two things: 1) the visual ability to segment and parse an observed motion in terms of a symbol language and 2) the symbol language itself. Other research focused on ways to obtain a good motion symbol language automatically. The approach of [Kuniyoshi *et al.* (2003)] involved self-learning. The robot watched itself performing various actions, and then learned a model of the relationship between actions and visual patterns. Then this model could be used to interpret a newly observed motion pattern in terms of the robot's own motor system.

An important body of work in the field of motion understanding involves the use of Hidden Markov Models (HMMs). In the approach of [Inamura *et al.* (2004)], a symbolic language of motion emerged as a result of the HMM learning procedure. The learned HMM symbols could then be used both to recognize human motions and to generate such motions on a robot. In another HMM-based approach [Lee & Nakamura (2006)], an observed motion is interpreted using a model based on previously learned motions. Then a new motion is generated using the state sequence inferred from the observed motion, resulting in an imitated motion that is conditioned on previous knowledge.

There are two basic approaches to motion recognition. These can be called the "discriminative" approach and the "generative" (or "descriptive") approach. In the discriminative approach, one uses a labeled database of pairs $\{T_i, L_i\}$ where T_i is a motion trajectory, and L_i is the label or category. These pairs are then used as training data to train a classifier, using an algorithm such as the SVM or AdaBoost. The limitation of this approach is that the number of trajectories is usually very small. This puts a limit on the complexity of the classifier that can be learned without overfitting.

In the generative approach, one partitions the training data into sets based on the label L_i. Then one uses these sets to learn multiple different models $Q_L(T)$ of the motion data itself. To recognize a new motion T_{new}, one simply guesses the label L^* corresponding to the model that assigns the highest probability to T_{new}:

$$L^* = \arg\max_L Q_L(T_{new})$$

The generative approach is therefore very different in philosophy from the discriminative approach. Importantly, the models $Q_L(T)$ constructed in this way can be much more complex than the discriminative models $Q(L|T)$, because the trajectory data contains much more information than the label data.

The HMM is basically a generative model, not a discriminative one, so when taking an HMM-based approach to motion recognition the generative approach is most often used. To train a model $Q_L(T)$, one invokes the HMM training procedure on the data subset corresponding to the label L.

One subtle point about the generative approach to recognition is that merely obtaining good models $Q_L(T)$ is not necessarily sufficient. To be successful it is not only important that $Q_L(T)$ assigns high probability to trajectories with label L, but also that it assigns low probability to trajectories with some other label $L' \neq L$.

The application of this chapter also takes the generative approach to motion recognition. The basic difference is that we use the PITA algorithm to produce models $Q_L(T)$ instead of the HMM. We also use a hybrid scheme, where the PITA algorithm is layered over initial models $\{u^k\}$ generated by the HMM.

10.2 Adaptation of PITA Algorithm for Motion Data

The motion capture data used in this chapter is presented in the form of a time series sequence of joint angles. At time t, a D-dimensional joint angle vector X_t^d is reported. The context C_t used to predict outcome X_t^d includes the joint angle vectors of all previous timesteps.

One simple approach to modeling the X_t^d data would be to lump all the joint data together into one batch, and use this batch as training data for PITA. We use a different approach, which is to split the data into separate batches corresponding to each joint angle d. Then we run D separate PITA instances, one for each element of the joint angle vector, resulting in models $Q_d(x|c)$. However, all D models use the same shared set of context information, and the same set of context functions $\mathcal{W}$. Thus the model

$Q_5(x|c)$ may depend somehow on the value of joint angle $d = 38$. From an implementation standpoint, this makes the learning process more efficient, because the logging for all D models can be done with one database traversal per PITA round.

The data used in the modeling process comes in packages of multiple trajectories. Thus, in general we can write $X^d_{u,t}$, where u indexes trajectories. The data outcomes $X^d_{u,t}$ are transformed into a batch $\{X^k\}_d$ used to train the dth PITA model using a basically uninteresting process. The important point is that for a data set $\{X^k\}$, there will be several k indices corresponding to $t = 1$ timesteps for which the context information will be empty ($C^k = \emptyset$).

For the results discussed below relating to standalone PITA, uniform initial models $u^k(x) = \frac{1}{200}$ were used (in the preprocessing step, the data is quantized into 200 bins). For the hybrid HMM+PITA case, the output of the trained HMM was used for the initial models $\{u^k\}$. The method used to obtain initial models from the HMMs is discussed below.

10.2.1 Context Functions

The list below describes the context functions used in the current application. The items marked "SPLT" indicate that the Shifted Prediction Laplacian Transformation described in Chapter 4 was used. The functions marked "BCF" indicate binary context functions, used in conjunction with 4-bin histograms. For the threshold parameters T, T_0, T_1 mentioned, the following set of values was used: $T_i = 20i, 1 \leq i \leq 9$. If the context function refers to values $k-1$ or $k-2$ but these values are not available because the k is near the beginning of a trajectory, a "null" prediction is reported.

- Basic SPLT (m): predict $f(C^k) = \frac{X^m_{k-1}}{200}$.
- Linear Interpolation SPLT (m): predict $f(C^k) = 2\frac{X^m_{k-1}}{200} - \frac{X^m_{k-2}}{200}$.
- Threshold BCF (m, T, I): $f(C^k) = 1$ if $\text{sign}(X^m_{k-1} - T) = I$, where $I = \pm 1$ and T is a threshold.
- Double Threshold BCF $(m, n, T_0, T_1, \circ)$: $f(C^k) = 1$ if $(X^m_{k-1} < T_0) \circ (X^n_{k-1} < T_1)$, where T_0, T_1 are thresholds and $\circ$ is a parameter denoting one of the four possible ways to combine two boolean values.

- Window BCF (m, T_0, T_1): $f(C^k) = 1$ if $T_0 < X_{k-1}^m < T_1$, where T_0, T_1 are thresholds.
- Time Lag SPLT/BCF (Δ, f_0): Predict $f(C^k) = f_0(C^{k-\Delta})$, where f_0 is some other context function/predictor listed above, and $\Delta \in \{2, 3, 4\}$.

As is generally the case in applications mentioned in this research, not much effort was expended attempting to find an ideal set of context functions. It is very likely that putting more time and thought into developing better context functions would provide better results. However, as we show below, the above list provides quite good results.

10.3 Hidden Markov Models

In this section we give a very brief description of the HMM idea, and refer the interested reader to the references [Bilmes (1998); Rabiner (1989)].

A basic Markov Model describes a process that transitions through a discrete number of states. At each time step, it potentially moves from one state to another. The "Markov" property means that the probability of a transition depends only on the current state. In a Hidden Markov Model, the state of the system is unobserved. Instead, the process is assumed to emit symbols or observations. Like the transition probabilities, the observation probabilities depend only on the current state.

Notationally, an HMM is defined by a five-tuple $\{S, \Pi, A, K, B\}$. There are $Q = |S|$ states $S = \{s_1, s_2, \ldots, s_Q\}$. Π is a Q-element distribution giving the initial probability of the process starting in a given state. A is a $Q \times Q$ matrix giving the probability of transitioning from one state to another. The set K denotes the observations that the process emits, while B is a table or function giving the probability of the process emitting a given observation in a given state:

$$B(o, s) = p(O_t = o | S_t = s)$$

Where S_t is the state of the process, and O_t is the observation recorded at timestep t. If the set K is countable and small, then B can take the form of a $|K| \times Q$ matrix. In the current research, and as is typical in the HMM-based motion literature, the observations correspond to D-dimensional joint angle vectors. Therefore $|K| \propto \exp D$

and so representing B as a matrix will require far too much memory. Instead, B is represented in terms of a set of state-conditional means and variances μ_s, Σ_s. Then we can write $B(o, s)$ as a Gaussian distribution:

$$B(o, s) = p(O_t = o|S_t = s) = N(O_t, \mu_s, \Sigma_s)$$

The joint angle observation O_t is a D-dimensional vector, as is the state mean vector μ_s. In principle Σ_s is a $D \times D$ covariance matrix. However, it is quite difficult to learn full covariance matrices effectively, and doing so tends to lead to overfitting, since the matrices are large. For these reasons, diagonal covariance matrices are used in this work.

To increase the expressive power of the model without increasing its complexity by too much, a mixture of Gaussians can be used for the observation probabilities. Now instead of having a single Gaussian $\{\mu_s, \Sigma_s\}$ for each state we have M Gaussians $\{\mu_{s,m}, \Sigma_{s,m}\}$. There is also a mixture weight matrix $W_{s,m}$, normalized so that $\sum_m W_{s,m} = 1$. The probability of an observation becomes:

$$p(O_t|S_t = s) = \sum_{m=1}^{M} W_{s,m} N(O_t, \mu_{s,m}, \Sigma_{s,m}) \tag{10.1}$$

For an HMM with D-dimensional observation vectors, Q hidden states, and M mixture components, the total number of parameters is:

$$Q^2 + MQD + MQD + MQ$$

Where the terms refer to the transitition matrix, the means $\mu_{s,m}$, the diagonal covariances $\Sigma_{s,m}$, and the mixture matrix $W_{s,m}$. In comparison, the number of parameters for an HMM with single Gaussian (no mixtures) but full covariance matrices is:

$$Q^2 + QD + \frac{1}{2}QD(D-1)$$

For this research, the number of hidden states was chosen to be $Q = 16$ and the number of mixture components was chosen to be $M = 6$. These values were chosen as the result of a preliminary analysis based on an AIC-style comparison of achieved log-likelihoods and number of parameters. For these choices of M and Q along with $D = 53$, the mixture component based HMM uses 10528 parameters while the full

covariance HMM uses 23152 parameters. So by using mixtures instead of full covariance, we can substantially reduce the complexity of the model. These parameter choices seem consistent with standard choices made in the HMM motion understanding literature [Kulic *et al.* (2007)].

The process of obtaining real probability distributions from the Guassian models is somewhat subtle due to the issue of normalization. Simply calculating the value of the Gaussian function at many points (Matlab "normpdf" command) will not in general yield a normalized distribution. In fact it is possible that for some points the value of the Gaussian is larger than one. When using the HMM for motion recognition, the normalization issue does not matter very much. This is because we are comparing unnormalized probabilities from model A to unnormalized probabilities from model B. However, in order to layer the PITA algorithm over the HMM models, it is necessary to produce legitimate probability distributions for the observations. This is discussed in the next section.

To train the HMM, the standard Expectation-Maximization algorithm was used. The EM algorithm was terminated when the relative improvement in log-likelihood fell below 10^{-3} per round. The HMM toolbox for Matlab by [Murphy (2005)] was used for almost all the HMM-related tasks discussed in this research, including the EM algorithm and the Forward-Backward algorithm. In particular the motion recognition results for the standalone HMM were obtained using only standard software and algorithms. The only HMM-related component developed specially for this research was the small component required to produce normalized initial models for the hybrid PITA+HMM scheme.

10.3.1 Real Distributions from HMM

In the experimental section below we report results for a hybrid model, generated by layering PITA over the models provided by the HMM. Obtaining initial models $\{u^k\}$ from the HMM is easy in principle, but in practice requires some care.

The key property of the HMM is that it supports efficient inference of the state occupation probabilities given the observations. That is to say, given a sequence of observations $O_1, O_2, \ldots O_{t-1}$ we can easily calculate the probability $p(s|O_{t-1} \ldots O_1)$

that the process is currently in state s. This is done using a simple calculation called the "forward" algorithm.

By combining the state occupation probability obtained using the forward algorithm with the observation models $p(o|s)$, we can easily find the implied probability distribution for a given observation:

$$p(O_t|O_{t-1}, \ldots O_1) = \sum_{s \in S} p(O_t|s)p(s|O_{t-1} \ldots O_1)$$

These distributions are then used as the initial models $\{u^k\}$ in PITA. Note however that some care must be taken to ensure that the distributions are normalized. The Gaussian function is normalized when integrated from $\{-\infty, \infty\}$, but we are using a quantized outcome space $O \in [1, 200]$. To produce normalized distributions, a normalization factor is computed:

$$Z = \sum_{o=1}^{200} p(o|s)$$

Where $p(o|s)$ is the standard Gaussian based on $\mu_{s,m}$ and $\Sigma_{s,m}$. Then the normalized probabilities $p'(o|S) = Z^{-1}p(o|S)$ were used.

10.4 Experimental Results

10.4.1 Data Sources and Preprocessing

We used the CMU Graphics Lab Motion Capture Database[1] as source data for this experiment. Unfortunately the motion sequences included in this database are not clearly categorized. Rather each trajectory is annotated with a qualitative description of the type of motion it contains. We identified 13 motion categories that could be unambiguously determined from the qualitative descriptions. These 13 categories included 245 trajectories.

The data files are represented as large sequences of joint angles recorded by the motion capture system. The following preprocessing was performed. First, the information corresponding to the base X/Y/Z position was discarded. Then, the minimum

[1] Available online at http://mocap.cs.cmu.edu/

and maximum values for a given sensor reading were found by scanning the entire database (all categories). Some sensors reported very small variation over the entire database; these sensors were discarded. That left 53 remaining sensor values with associated minimum and maximum values. The sensor readings were then quantized according to the formula:

$$x = \left\lceil 200 \frac{(x - x_{min})}{(x_{max} - x_{min})} \right\rceil \tag{10.2}$$

This ensures that $\{X^k\}$ data for all sensor readings was distributed in the range $[1, 200]$. Note that all three learning schemes (HMM, PITA, Hybrid) use the same set of preprocessed data.

The original data is made up of a relatively small number (245) of relatively large trajectories. Results for this version of the problem are given below, but it is somewhat difficult to make good comparisons between techniques due to the small number of samples. We also study another version of the problem in which the trajectories were split into subsequences of length $T = 50$. This has the effect of making the recognition problem harder, and of increasing the total number of trajectories. This version of the problem could correspond to applications where it is necessary recognize motions such as falling, seizures, or sudden violent activity from a short number of motion frames.

The following cross-validation procedure was used to separate the data into training and testing sets. Each category was divided into three partitions A, B, C. Then the train/test process was run three times, with one subset being used for testing and the other two used for training. So the second cross-validation round uses subsets $A + C$ as the training data and subset B as the test data. When doing the recognition test, the competitor models trained using the same subsets were used. So for recognizing subset B, the models trained using subsets A, C from each category ("soccer", "basket", "signals", etc) were used to assign likelihoods to the subsequences in B.

Table 10.1 shows the results of the basic version of the recognition test in which the trajectories were not split. We see that PITA correctly recognizes 17 more trajectories than the HMM, correspond to a 7% difference in error rate (29% vs. 22%). On this version of the problem, the hybrid model is less successful, recognizing 8 fewer trajectories than the HMM. This is probably because the hybrid model is more complex, and thus tends to overfit when trained on a small number of trajectories.

Table 10.2 shows the results of the recognition test on the split trajectory version of the problem. As can be seen from the table, both PITA and the hybrid model show very strong performance relative to the standalone HMM. When measured in terms of total error on the entire database, standalone PITA cuts the error rate in half compared to the HMM, and the hybrid does even better. Total error rates are 75% for the HMM, 47% for standalone PITA, and 38% for the hybrid. However, the total over the entire data set may be an inappropriate measure, since it biases the analysis toward the larger data sets, where the PITA and hybrid models have an advantage. If one counts only the number of domains, then the score for HMM vs. PITA is 6-7 in favor of PITA. For the HMM vs. hybrid, the score is 5-8 in favor of the hybrid.

Figure 10.2 shows a comparison of the codelengths achieved by the various models on both the training and test data sets. In every case, standalone PITA gives the best codelengths, followed by the hybrid. Furthermore, since the codelengths assigned to the unseen test data are not much greater than those assigned to the training data, it appears that the standalone PITA models are not particularly vulnerable to overfitting. In contrast, the hybrid model is more vulnerable to overfitting, probably because it is more complex than the other models.

Figure 10.1 shows an interesting visual result related to the distribution of PIT values. These grids show frequency data related to the distribution of consecutive PIT values for four different models (flat uniform, HMM alone, PITA alone, and hybrid). Given a pair of data outcomes X^d_{t-1}, X^d_t, the corresponding PIT values ϕ^d_{t-1}, ϕ^d_t are calculated based on the models. Then a frequency count is logged for the pair, corresponding to its location in the $[0,1]^2$ plane. Obviously there will be a very strong correlation between X^d_{t-1} and X^d_t. This is illustrated in Figure 10.1(a), which uses a uniform model so that $\phi^d_t \propto X^d_t$. On the other hand, an ideal model should produce PIT values that are evenly distributed on $[0,1]^2$. Standalone PITA produces the most uniform grid, followed by the hybrid model and then the HMM.

10.5 Summary

In this chapter we showed how the PITA algorithm could be used to build models of data obtained from a motion capture system. The process was a relatively straightfor-

Table 10.1: Number of trajectories categorized correctly by each modeling technique, for the unsplit problem.

Domain	Total	HMM	PITA	PITA+HMM
soccer	6	0	5	0
playground	11	11	10	11
basket	12	8	11	8
cockrobin	14	9	4	9
teapot	16	1	9	0
walk	18	10	14	10
jump	18	16	17	14
signals	19	5	6	7
dance	21	15	16	14
alaska	23	21	19	21
sit	23	17	16	13
run	27	24	27	22
uneven	37	37	37	37
total	245	174	191	166

ward application of the recipe for using PITA: identify the data outcomes and contexts, write down a set of context functions, and go.

The experimental results show that the resulting PITA models are very good at the *descriptive* task. That is to say, the models assign very high probability (low code-length) to the observed data. Thus, if the goal is purely to describe or compress a motion data set, the PITA algorithm should be preferred to the HMM.

PITA also does very well in comparison to the HMM on the motion recognition task. PITA achieves significantly better recognition rates on both the split and unsplit version of the recognition task. A hybrid model built by layering the PITA models over the HMM achieves even better performance for the split version of the task.

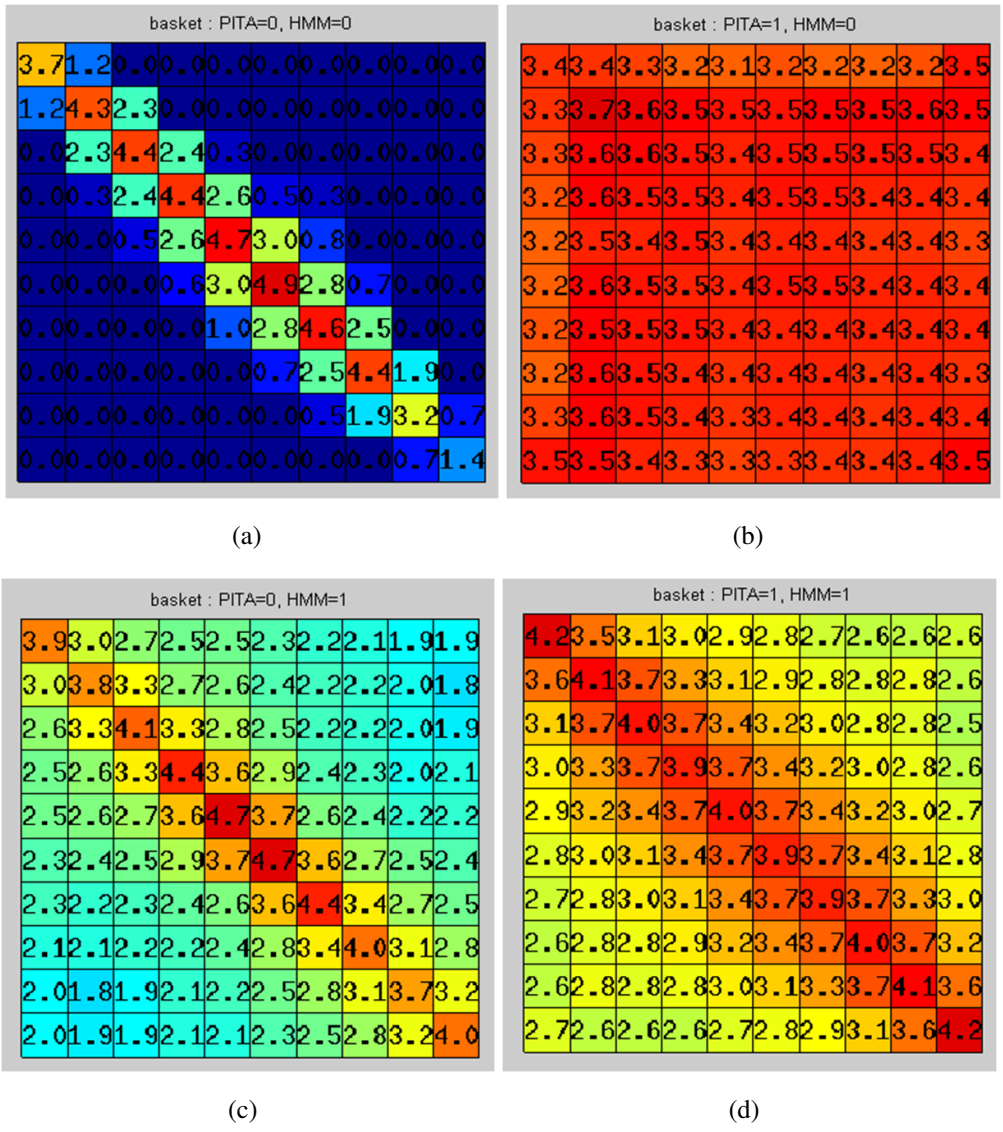
The PITA algorithm also has several practical advantages with respect to the HMM. One advantage is its simplified training process. The HMM training process is based on Expectation Maximization, and requires an initial guess for the parameter values (Gaussian mean/variance, mixture matrix, transition matrix, and initial state distribu-

Table 10.2: Number of trajectories categorized correctly by each modeling technique, for the $T = 50$ split version of the problem.

Domain	Total	HMM	PITA	PITA+HMM
soccer	64	56	22	43
run	73	72	70	71
basket	148	141	108	135
walk	149	98	87	99
jump	157	126	54	118
dance	355	153	149	257
teapot	681	240	427	117
playground	835	145	456	613
sit	1064	219	567	679
signals	1498	381	581	390
cockrobin	2050	112	640	898
uneven	2857	997	2465	2806
alaska	3003	439	1192	1764
total	12934	3179	6818	7990

tion). The outcome of the training process depends strongly on this initial guess. In practice, researchers generally run the training process several times with different initial starting values. This is clearly an awkward solution. The PITA learning process is much more reliable and does not require any initial guess for the parameters.

Furthermore, to use the HMM one must select values for the Q and M parameters (number of hidden states and number of mixture components). Selecting the optimal parameters appears to be more of an art than a science. In contrast, for the PITA algorithm one must select the battery of context functions $\mathcal{W}$. It may seem like developing $\mathcal{W}$ is much more difficult than choosing Q and M. However, the key difference is that when choosing $\mathcal{W}$ there is no strong penalty for making a mistake. If some "bad" functions are included in $\mathcal{W}$, the PITA algorithm will simply decline to select them. In contrast a bad choice for Q or M will lead to reduced performance for the HMM.



(a) (b)

(c) (d)

Figure 10.1: Illustration of correlation between PIT values for the adjacent time steps for the "basket" data set. Matrix values indicate the log frequency of PIT value pairs, where a pair is obtained by looking at the PIT values ϕ^d_{t-1}, ϕ^d_t corresponding to adjacent data outcomes X^d_{t-1}, X^d_t for the same joint angle d. If a perfect model for the outcome data were used to generate the PIT values, the adjacent PIT values would be independent and the frequency matrix would be uniform. It can be seen that for this dataset, PITA (b) gives the best result, followed by the hybrid (d) and the HMM (c). Note that the numbers indicate frequency $\log_{10}$, so that an box marked 3.0 has ten times more hits than a box marked 2.0.

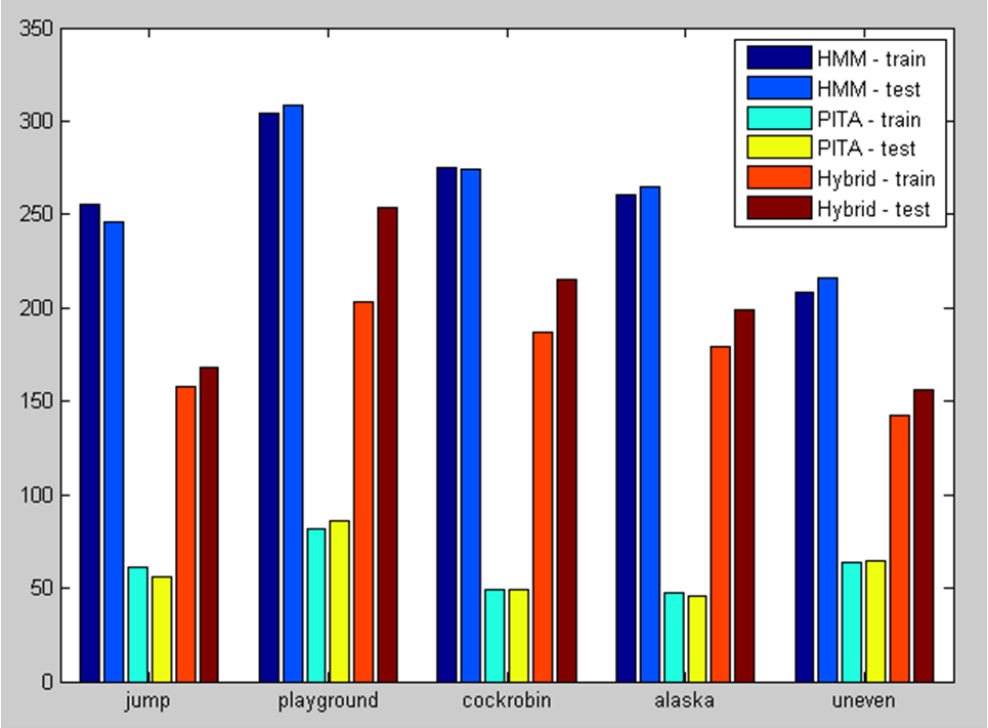


Figure 10.2: Code length comparison bar plot for five motion categories, for different models on test and train data. The Y-axis shows average bits per timestep, which is just the average negative log likelihood assigned to a full observation vector by the model. Uniform models would require $53 \cdot \log_2(200) \approx 405$ bits per timestep.

Chapter 11

Conclusion

11.1 AIT View of Statistical Modeling

Historically, most research in statistical modeling has been based on the Maximum Likelihood Principle of traditional statistics. This thesis demonstrated that a different perspective is possible. This new perspective, called the AIT view, is based on the idea that, when encoding a data set using a perfect model of it, the resulting encoded data should be completely random. If a randomness deficiency is detected in the encoded data, this indicates that the model is imperfect and provides a way to improve it.

Until now, two important conceptual barriers prevented the AIT view from being used for statistical modeling. The first was a limited understanding of what kinds of statistical tests (or context functions) could legitimately be used to detect randomness deficiencies. This thesis demonstrated that it is possible to use statistical tests based on either the original form of the data, or some external data source (e.g. in the pattern recognition problem). The second barrier was simply that bit strings are difficult to use. The solution to this obstacle is to to represent the encoded data as Probability Integral Transform values instead of bits. This representation is much easier to work with, since the number of PIT values does not change. Instead of reducing the number of PIT values, the modeling process expands the region of the unit interval the PIT values are allowed to inhabit.

These insights were used to develop a practical algorithm called PITA. However, it should be emphasized that PITA is only one of many possible algorithms that can be designed on the basis of the AIT view. The view of traditional statistics has been

developed for decades, and contains hundreds of methods. The AIT view is just beginning to emerge, but probably contains an equivalent number of methods, the majority of which are waiting to be discovered.

11.2 PITA Algorithm

The AIT view suggests the following layered, iterative approach to modeling. Start by transforming the data set into a bit string, using any naïve encoding method. Then search the bit string for randomness deficiencies. If one is found, exploit it to reencode (compress) the bit string. Repeat this search and reencode process until no further randomness deficiencies can be found.

The PITA algorithm uses this search and reencode process, except using PIT values instead of bit strings. A randomness deficiency in a PIT value sequence can be detected by constructing a histogram of PIT values. As discussed in Chapter 2, this PIT histogram can then be used to define a simple model update. This trivial-seeming update has two powerful properties. First, the update can be applied to an entire *ensemble* of models. Second, the codelength savings achieved by applying the update can be predicted accurately given only the information in the PIT histogram. Thus, even when there are ten million distinct models describing ten million different data points, the 8 (or 10, 12, etc) histogram bin counts contain all the information necessary to define the update and to predict the savings.

Chapter 3 completes the description of the basic PITA algorithm, which uses the model ensemble update method along with a battery of context functions. In each training round, every context function is scored by measuring the randomness deficiency it reveals in the encoded data. The context function with the highest score is selected and used to update the model ensemble and PIT values. The algorithm terminates when no further randomness deficiencies can be detected. The output of PITA is a complex context-dependent model $Q(x|c)$, which is represented implicitly by the context functions and histogram data obtained in each round. By using the PIT values (encoded data) instead of the original data, the hard problems of feature selection and feature combination are easily solved.

Two thorny technical issues need to be addressed to realize the full potential of PITA: the problem of non-numeric data and the Bin Overlap problem. Non-numeric

data is troublesome because PITA depends on the use of model CDFs, and many different CDFs can correspond to the same PDF, depending on the outcome ordering. The Bin Overlap problem is due to a failure of the key assumption that underlies the codelength savings calculation. Solutions to these issues are also presented in Chapter 3.

Chapter 4 describes a set of alternative model ensemble updates based on parametric transformations. The key here is to define transformations for which an optimal parameter can be calculated from a small set of logged statistics relating to the PIT values. The most powerful transformation discussed is the Shifted Prediction Laplacian, which depends on a new type of "predictor" context function. If the PIT value can be predicted accurately, substantial codelength savings can be achieved. This new type of context function greatly expands the scope of PITA.

A powerful feature of the PITA algorithm is that it can be *layered* over some other model. Here, an arbitrary, black-box algorithm is used to initialize the models. The PIT values are generated using the initial models, and then the PITA algorithm proceeds as before. No changes to the algorithm are necessary for this to work. Applying PITA in the layering mode is a "win-win" proposition. If randomness deficiencies are revealed by the context functions, then the resulting updated models will be better than the initial models. If no randomness deficiencies are revealed, this provides evidence that the initial models are very good.

Chapter 5 discusses the relationship between PITA and other methods in machine learning and statistics, especially Maximum Entropy modeling and Boosting. With regard to the former, a significant advantage of PITA is its ability to scale up to data sets that are too large to be held in memory. The optimization methods related to Maximum Entropy require multiple sums over the entire data set, incurring a huge computational cost when the data set is large. In contrast, PITA requires only a single data set traversal to obtain all the statistics necessary both to score the context functions and to update the models. With regard to boosting, the primary advantage of PITA is that it produces probability distributions instead of point predictions. This allows PITA to be used in layering mode, as well as for raw data modeling and data compression applications.

11.3 Applications

Five applications of the PITA algorithm were discussed in this thesis. The broad set of applications demonstrates PITA's generality. Each application contained a different insight or lesson.

Chapter 6 discussed an image compression application. This application was examined first because lossless data compression is a rigorous test. If there were some critical flaw or error in the logic behind PITA, that flaw would be detected when the decompressed image failed to match the original image exactly. The compression rates achieved by PITA were comparable to PNG, a widely used lossless compression format. This result is not revolutionary, but note that it was achieved using a very simple set of context functions. A more sophisticated set of context functions should produce better compression rates.

In Chapter 7 it was shown that PITA can be used for binary pattern classification, a core problem in machine learning. A slightly modified version of PITA was compared to AdaBoost, and showed competitive results. On two domains, the modified PITA algorithm achieved a decisive performance improvement over AdaBoost. The new algorithm also seemed to achieve consistently better results on the larger domains.

Chapter 8 discussed an application to word morphology modeling. Several aspects of this problem make it interesting. First, in this domain the layering ability of PITA was put to good use. Standard bigrams were used for the initial models. The resulting PITA-enhanced models achieved much better codelength than the bigram models alone. Another interesting aspect of this domain is that the outcomes are non-numeric. The CDF reordering scheme and the variable width histogram scheme were used to deal with this issue, with successful results.

The speech modeling application of Chapter 9 showed how PITA could be used to construct highly complex models. The PITA model used for this application required about 10^6 bits to encode, but resulted in a savings of more than $8 \cdot 10^7$ bits. This means that the complexity of the model is *justified* by the MDL principle, and for this reason overfitting is not a problem. Furthermore, this highly complex model was constructed efficiently, due to the fact that PITA requires only one database traversal per round. This is of crucial importance, since the speech database is too large to fit in memory.

Chapter 10 presented an application of PITA to the problem of motion modeling and recognition. In terms of recognition performance, PITA achieved substantially better results than an HMM-based system. A hybrid scheme, in which PITA updates were layered on top of an HMM, resulted in even better performance on one version of the problem. Furthermore, the pure PITA models were *much* better than the HMMs in terms of their ability to describe the data (codelength). PITA also seems easier to use. The HMM requires the user to make difficult choices related to the number of hidden states and observation models. Also, the EM-based HMM training process can get trapped in local minima. PITA does not have these problems.

11.4 Future Work

The approach to statistical modeling as a search for randomness deficiencies opens many doors for future work. PITA is only one of many algorithms that can be defined based on the AIT view. There are many types of randomness deficiencies that may be present in a bit string or PIT value sequence, and many ways to exploit such deficiencies when they are found.

More concretely, two important pieces of future work involve addressing limitations in PITA. The first limitation is PITA's requirement that the user define a set of context functions $\mathcal{W}$. To some extent, the user can simply define a very large batch of context functions, and rely on PITA's feature selection mechanism to choose the best ones. However, the computational cost of the algorithm scales with the size of $\mathcal{W}$. Thus, there is a need for automatic methods to search for good context functions. One possibility is a kind of genetic algorithm where each "phenotype" corresponds to a context function. By using mutation and selection methods, it may be possible to evolve powerful context functions from a simple set of primitive operations.

Another limitation of PITA is that it does not contain the ability to represent hidden abstractions. A very intuitive concept in modeling is the notion of a hidden cause that produces some observations. For example, in a picture of a face, it is conceptually useful to think of the raw pixel observations as being caused by the presence of a "face" abstraction. A major attraction of techniques such as HMMs and graphical models is the ability to represent hidden abstractions. Thus, another piece of future work is to develop a modified version of PITA that has this ability as well.

An interesting possibility is to apply the PITA algorithm to the semi-supervised learning problem [Zhu (2005)]. In this problem, one has a large quantity of unlabeled data samples as well as a much smaller set of labeled samples. The goal of semi-supervised learning is to exploit the large set of unlabeled samples to improve performance beyond what could normally be achieved using the small labeled set. PITA could be applied to this problem by first using it to model the raw data. Then the features chosen by PITA on the raw data modeling phase could be reused for the supervised learning phase.

The final piece of future work is to develop a theory of the brain based on the AIT view. Many brain theories have been developed that employ concepts from traditional statistics and information theory. Most relevant to this thesis is the theory of redundancy reduction [Barlow (2001)]. This idea suggests that the function of the cortex is to reencode sensory data entering the brain so as to reduce redundancy (i.e. compress it). This theory is quite attractive, but contains some critical flaws. For one thing, if the cortex acts to compress the sensory signals as they travel up through higher brain areas, then higher brain areas should be smaller than lower areas. In fact the opposite tendency is observed.

In the revised theory of redundancy reduction based on the AIT view, the original data is the sensory signal, and the set of synaptic weights play the role of the model. Some neural quantity such as firing rate is identified as the encoded data. When presented with a sensory signal, the neurons are forced into a narrow region of their full potential range, in the same way that a particular data outcome forces the PIT values into a specific region of the unit interval. The size of the confined region depends on the specific sensory signal and the model (synaptic weights). In this view, the goal of learning is to expand the aggregate hypervolume of the regions that the "neural PIT values" are confined to, by adapting to the statistics of the sensory signals. Thus, when a mature brain is presented with a common sensory signal, the neural PIT values are allowed to inhabit a relatively large region.

By viewing the neural activities as actual physical quantities, analogous to the positions and velocities of atoms in an ideal gas, an extremely simple view of learning becomes possible. Learning is simply the process of maximizing the *physical* entropy of the neural PIT values. In classical statistical mechanics, entropy is maximized while obeying the law of energy conservation. In the present view of learning, entropy is

maximized while obeying the rule that confines the neural PIT values into a specific region depending on the sensory signal and the model. If this interpretation is true, then what is perhaps the most mysterious natural phenomenon known to man - intelligence and learning - is revealed to be nothing more than a special case of the most basic and ubiquitous physical process of all: entropy maximization.

Appendix A

Information Theory

This appendix presents a very brief overview of the some of the core concepts of information theory required to understand the thesis. For a fuller treatment, see [Cover & Thomas (1991)].

The motivating question of information theory is: given a data set $\mathcal{X}$, how can we encode $\mathcal{X}$ in a digital form using the smallest possible number of bits? In spite of the very technical nature of this question, the answer turns out to be quite deep.

The central difficulty of the data compression problem is that the number of short codes is intrinsically limited. For example, there are only $2^4 = 16$ codes of length 4 bits. Thus if there are more than 16 possibles outcomes in the data set, then some of those outcomes must be assigned codes of length greater than 4 bits. This means that to achieve good compression (small net codelength), it is necessary to reserve the short codes for the common outcomes, while assigning long codes to rare outcomes. This observation immediately raises the question: what is the precise relationship between the probability of an outcome and the length of the corresponding code? The answer is given by Shannon's famous codelength-probability relationship:

$$L(x) = -\log_2 P(x) \tag{A.1}$$

The above equation specifies the *lengths* of the optimal code. How then can the code itself be constructed? That is to say, given a model $Q(x)$, how do we transform a series of outcomes $\{X^1, X^2 \ldots\}$ into a bit string S that satisfies A.1? This is called the encoding problem, and it is non-trivial. Fortunately, however, the encoding problem

has been completely solved. The first method proposed that was guaranteed to achieve near-optimal codelengths is called Huffman encoding [Huffman (1952)]. The second method, used in this thesis, is called arithmetic encoding [Rissanen (1976); Witten *et al.* (1987)]. The basic advantage of arithmetic encoding over Huffman encoding is that the latter basically requires the model $Q(x)$ to be pre-specified. Since the applications of this thesis involve conditional models $Q(x|c)$, and c is constantly changing, it is difficult to use Huffman encoding.

Given the codelength-probability equivalence relationship, it is natural to define the *entropy*:

$$H(P) = \sum_x P(x)L(x) = -\sum_x P(x)\log_2 P(x) \tag{A.2}$$

The entropy of a distribution $P(x)$ is the expected codelength achieved by the optimal code for $P(x)$. There are several important facts to notice about the entropy. First, it is a function of the distribution $P(x)$, not the data set $\mathcal{X}$. Second, upper bounds for the entropy can easily be found. For example, if there are 16 possible outcomes, then the entropy is as most 4 bits. Generally speaking, the greater the number of outcomes in the distribution, the larger the entropy.

While the entropy is a fascinating concept, it is not actually very useful in statistical modeling, because the real distribution $P(x)$ is generally unknown. The goal of statistics is to obtain approximations (or models) $Q(x) \approx P(x)$ on the basis of the empirical data set $\mathcal{X}$. In statistics a more useful quantity is the empirical codelength:

$$\sum_{k=1}^{N} L(X^k) = -\sum_{k=1}^{N} \log_2 Q(X^k) \tag{A.3}$$

Note that the codelength is formally equivalent to the negative log likelihood. For this reason, the goal of finding a model that can be used to compress a dataset to the smallest possible size, is exactly equivalent to the goal of finding a model that assigns the highest probability to the data (Maximum Likelihood Principle).

Again, in statistics we never know the real distribution $P(x)$. Instead we have an approximation $Q(x)$. Since $Q(x)$ is imperfect, the codelengths specified by $Q(x)$ will also be imperfect. The codelength penalty for using imperfect codes based on the model $Q(x)$ instead of perfect codes based on the real distribution $P(x)$ is:

$$
\begin{aligned}
D(P||Q) &= \sum_x P(x)L(x) - \sum_x P(x)L_q(x) \\
&= \sum_x P(x) \log_2 \frac{P(x)}{Q(x)}
\end{aligned}
$$

The quantity $D(P||Q)$ is known as the Kullback-Liebler divergence. It is easy to show that the Kullback-Liebler divergence is always positive. This implies that no scheme for assigning codelengths to data outcomes can achieve better codelengths on average than $L(x) = -\log_2 P(x)$. It is possible to get better codes for certain outcomes. But by reducing the codelength of some outcomes, it is necessary to increase the codelength of other outcomes, and this will hurt the overall average.

For the purposes of this thesis information theory contains two key points. The first is the equivalence between statistical models and compression programs. A compression program is a bijective mapping $X^N \leftrightarrow \mathcal{S}$ between data outcomes and bit strings. A compression program can easily be built out of a model $Q(x)$ simply by hooking it up to an arithmetic encoder. Conversely, any mapping $X^N \leftrightarrow \mathcal{S}$ must implicitly contain a probability model. The second key point is the idea that a better model leads to shorter codes, and vice versa.

In the light of these ideas, consider the following argument. Given a model $Q(x)$ for a data set $\mathcal{X}$, define E_Q to be the corresponding compression program. Let $S = E_Q(\mathcal{X})$. Now consider some arbitrary data compression program Z (such as "zip"). Z is a mapping from bit strings to bit strings: $\mathcal{S} \leftrightarrow \mathcal{S}$. Assume the compression program succeeds in compressing S so that $S' = Z(S)$ and $|S'| < |S|$. Now consider the function $E'_Q = Z(E_Q)$. This is a mapping $X^N \leftrightarrow \mathcal{S}$, so from the probability-codelength equivalence idea, E'_Q must implicitly contain a model Q'. Furthermore, since E'_Q achieves a shorter code than E_Q, it must be the case that Q' is a better model than Q.

The key assumption above is that the data compression program Z could actually compress the encoded bit string S. This motivates the question of when in general a bit string can be compressed. As discussed in Appendix B, a string S can be compressed if it has a randomness deficiency. This shows that searching for randomness deficiencies in the string S is equivalent to searching for ways of improving the model Q.

Appendix B

Algorithmic Information Theory

This appendix presents a very brief overview of some of the core concepts of algorithmic information theory (AIT) necessary to understand the thesis. AIT is a rich mathematical theory. This thesis uses primarily the mindset developed by AIT and not the technical apparatus. Interested readers are referred to the text [Li & Vitanyi (1997)].

In AIT the essential quantity of interest is called the *Kolmogorov complexity*. The Kolmogorov complexity of a string S is the length of the smallest possible program p_U that will output S when run on some universal Turing machine U. Initially, it would appear that the Kolmogorov complexity depends completely on the choice of U. However, it turns out that if S is long enough, then its Kolmogorov complexity becomes essentially an absolute quantity. This is because of the universality of computation: a universal Turing machine U can be made to simulate any other Turing machine A by using a translation program T_{UA}. Thus given a program p_A that generates S on A, a program p_U can be found easily by prepending T_{UA} to p_A. The length of T_{UA} is a constant independent of S, so as S becomes large (and complex) its Kolmogorov complexity effectively becomes an absolute quantity.

For an intuitive understanding of the meaning of the Kolmogorov complexity, consider the following strings:

- 00000000000000000000000000000000000000000
- 10101010101010101010101010101010101010101

- 01101010001101000100101110111101100110 11

Clearly the first and second strings are nonrandom. One could easily write a program that generates these strings, and if they continued on for a million bits, the program would be much shorter than strings themselves. Thus the Kolmogorov complexity of the first two strings is much smaller than their actual length. In contrast, the third string has no discernible structure. It is not possible to write a short program that outputs this string. Thus, its Kolmogorov complexity is approximately equal to its length.

The main drawback of the concept of Kolmogorov complexity is that the problem of calculating it for a given S is uncomputable. The best we can do is to search for a series of increasingly tight upper bounds. Given some program p_U that generates S, if some program p'_U can be found that is substantially shorter than p_U and generates p_U as its output, then the Kolmogorov complexity of S cannot be much greater than $|p'_U|$. This gives a new upper bound on the Kolmogorov complexity of S, and the process can be repeated by searching for programs that produce p'_U.

The Kolmogorov complexity concept is deeply related to the idea of randomness [Martin-Löf (1966)]. The key realization is: if a string S is random, then its Kolmogorov complexity is approximately equal to its length. If on the other hand S is nonrandom, then its Kolmogorov complexity is substantially smaller than its length. This is fine, but what does it mean for a string to be nonrandom? A string is nonrandom if and only if it contains a randomness deficiency. A randomness deficiency is observed when a statistical test is applied to a string, and the result deviates substantially from what would be expected if the string were random. One simple example of a statistical test is to count the frequency of 1's in the bit string. In a random string, this frequency should be close to 50%.

A key point is that there are a vast number of different statistical tests that can be applied to a bit string. In order to be random, the string must satisfy all of these tests. However, in practice the number of tests that can actually be used is limited by considerations of computational complexity. For this reason, it is practically impossible to know if a string is truly random; one can only prove conclusively that a string is nonrandom. This situation is analogous to the situation in statistical modeling. In

modeling, given two models $Q_a(x)$ and $Q_b(x)$, it is possible to decide if Q_a is better than Q_b (or vice versa). But it is impossible to know if Q_a is the best possible model.

One way of defining a statistical test is to use a binary context function w. Let b_i denote the ith bit in string S, and let $S_{1:i-1}$ denote the substring of S leading up to b_i. The function w selects a subset of the bits of S:

$$B_0 = \{b_i : w(S_{1:i-1}) = 0\}$$
$$B_1 = \{b_i : w(S_{1:i-1}) = 1\}$$

Now for the string S to be random, the number of 1's in both subsets B_0, B_1 must be about 50%. This must be true for every choice of w. Furthermore, this context function based method is only one way of detecting randomness deficiencies. The huge variety of options that can be used to search for randomness deficiencies is a central motivating factor for this thesis.

Appendix C

Layering Example

The use of PITA in layering mode, where PITA model ensemble updates are applied on top of some black box model, was mentioned in Section 3.8. This Appendix gives a more detailed example where layering can be useful.

In some sense, whenever PITA is used, it is used in layering mode. Each PITA round is basically a new layer laid over the previous model. The first illustration of the model update idea, in Figure 2.1 of Chapter 2, showed how it could be used to improve a Gaussian model. However, that simple example may not be compelling. In that case, it was quite obvious that the data set could not be described well using a Gaussian. This could be verified simply by plotting the data outcomes against the Gaussian curve. Furthermore, the improved model obtained using the model ensemble update was not particularly impressive either, as a better model could be found by using a simple scheme such as kernel density estimation.

The example presented here uses the same basic process as the updated-Gaussian one. There is a model that has been optimized to fit a data set (recall that the Gaussian example used the optimal mean and variance), and the model is used to generate the PIT values. The uneven distribution of the PIT values indicates that the model is imperfect, and so the PIT value histograms are used to update it. The difference is that in this case both the data and the model are quite complex. The model is a type of Markov Random Field (MRF) and the data is an image. The MRF assigns a different probability distribution to each pixel depending on the values of the neighboring pixels. Thus, one cannot simply graph the distribution of pixel values and compare this to the model distribution. One *can*, however, generate PIT values for each pixel, and then search for

randomness deficiencies in the PIT value distribution. If randomness deficiencies are found, an improved model can be obtained by applying the model ensemble update.

C.1 Markov Random Field Model

The complex statistical model used in this example is a kind of Markov Random Field (MRF). This type of model is used widely in computer vision, often for the purpose of image segmentation [Bouman & Shapiro (1994); Deng & Clausi (2004)]. However, the point of the example is not to demonstrate a specific way of improving models used for computer vision. The example illustrates a general problem in statistical modeling: one has a trained model that produces some log-likelihood value (codelength) on the data set. The model may be good compared to other models in the same class (e.g. other MRFs defined by different parameters). But there is no way to tell if the model is good in absolute terms, since there is no way to tell what the best achievable codelength is.

The type of MRF used in this example is actually quite similar to an HMM. There are some number of hidden states, and each hidden state has a different observation probability distribution associated with it. The main difference between the HMM and the MRF model used here, is that for the MRF, the probability of a hidden state depends on not one but two neighboring hidden states. The probability of a hidden state (i, j) is influenced by the hidden state at positions $(i-1, j)$ and $(i, j-1)$:

$$p(S_{i,j} == s) = \frac{1}{Z} \exp(-U_{i,j}(s)) \tag{C.1}$$

Where Z is a partition function to ensure normalization, and $U(\cdot)$ is an "energy" function depending on the neighboring values:

$$U_{i,j}(s) = \beta(\delta(s, S_{i-1,j}) + \delta(s, S_{i,j-1})) \tag{C.2}$$

Where $\beta < 0$ is a parameter of the model. Thus states s that match their neighbors are assigned lower energies, which correspond to higher probabilities.

Each hidden state contains a different pixel observation probability, which is a Gaussian with mean (μ_s, Σ_s), where Σ_s is a diagonal covariance matrix. The full

distribution for a pixel is found by taking the sum of the observation probabilities, weighted by the state occupation probabilities:

$$p(O_{i,j} == o) = \sum_s p(o|s)p(S_{i,j} == s|S_{i-1,j}, S_{i,j-1}) \tag{C.3}$$

Where $p(o|s)$ is a Gaussian with mean μ_s and covariance Σ_s.

Note that the probability $p(S_{i,j} == s|S_{i-1,j}, S_{i,j-1})$ is the state occupation probability *before* the pixel "observation" is made. After making the observation, the updated probability is:

$$p(S_{i,j} == s) = \frac{p(O_{i,j} == o|s)p(s|S_{i-1,j}, S_{i,j-1})}{\sum_{s'} p(O_{i,j} == o|s')p(s'|S_{i-1,j}, S_{i,j-1})} \tag{C.4}$$

These updated state occupation probabilities are then used to calculate the state occupation probability of the neighboring pixel.

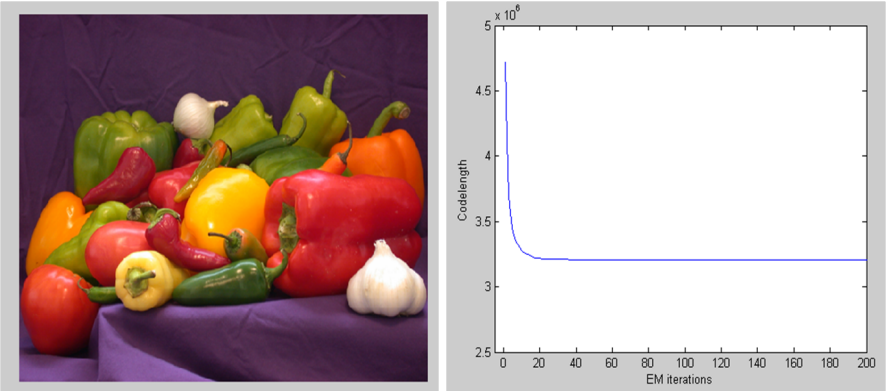
C.2 Training the Model

To optimize the MRF for the data set, it is necessary to find good values for the hidden state Gaussian parameters μ_s and Σ_s, and the state influence parameter β. The parameters are randomly initialized, and then iteratively improved using the Expectation-Maximization algorithm as follows:

- Given current parameters μ_s and Σ_s, use the inference algorithm to guess the most likely hidden state $\hat{s}_{i,j}$ for each pixel.
- Given the guess of the hidden states $\hat{s}_{i,j}$, find the maximum likelihood values for each state's Gaussian parameters:

$$\begin{aligned} \mu_s &= \mathbf{mean}\{O_{i,j} : \hat{s}_{i,j} = s\} \\ \Sigma_s &= \mathbf{var}\{O_{i,j} : \hat{s}_{i,j} = s\} \end{aligned}$$

- Calculate the optimal state influence parameter β using a binary search.
- Calculate the new log-likelihood corresponding to the updated model.



(a) (b)

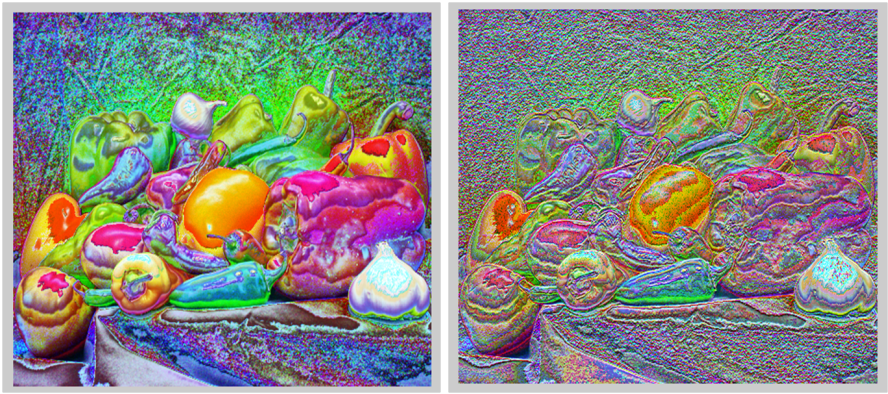
Figure C.1: Figure (a) shows the Matlab "peppers" image, used as training data in this example. Figure (b) shows the learning curve produced by using the Expectation-Maximization algorithm to train the MRF model.

The above loop is repeated until the log-likelihood values converge. This EM algorithm was run on the Matlab "peppers" image shown in Figure C.1(a). The codelength (negative log likelihood) progress achieved as a function of the number of EM iterations is shown in Figure C.1(b).

C.3 PITA Update

Once the EM algorithm has converged, we have a trained model. As shown in Figure C.1(b), the model can be used to achieve a codelength of $3.2 \cdot 10^6$ bits. At this point it is very difficult to know whether the model is good or not. Surely it is better than the MRF model defined by the initial parameter guess used to initialize the EM algorithm; that model required about $4.5 \cdot 10^6$ bits to encode the image. It is also better than a uniform model that would require $384 \cdot 512 \cdot 3 \cdot 8 \approx 4.7 \cdot 10^6$ bits to encode the image. But it is very difficult to know whether or not some better model could achieve an even lower codelength for the image.

PITA ideas can be used to address this question. One way of visualizing the quality of the model is by looking at the PIT value image. Recall that there is a 1-1 correspondence between PIT values and data outcomes (pixels). Therefore, we can calculate the PIT values using the MRF model, and then view the PIT values as an image, where



(a) (b)

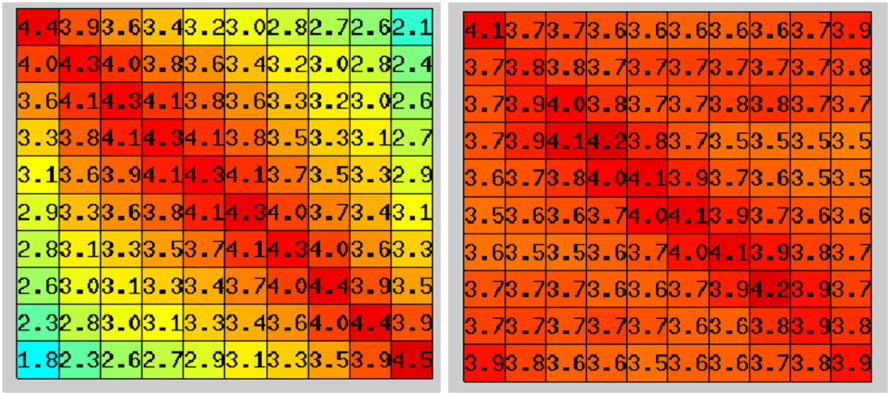
Figure C.2: These images are produced by viewing the PIT values $\phi_{i,j}$ corresponding to the original pixels $X_{i,j}$. The PIT value image produced by a perfect model should be completely random. Figure (a) shows the PIT values produced using the trained MRF. Figure (b) shows the values produced using the updated MRF+PITA model.

each PIT value is in the same position as the corresponding pixel. The PIT value image generated using the MRF model is shown in Figure C.2(a).

What should this image be expected to look like? The basic idea of the AIT view of modeling is that if the model is perfect, then the encoded data should be random. Clearly, the PIT value image shown in Figure C.2(a) is not random. There are obvious regions, shapes, and edges corresponding to the original positions of the peppers. It is certainly "more random" than the original image, but not completely random. This implies that the MRF model is imperfect.

Another way of visualizing the PIT values is by looking at the frequencies of adjacent PIT value pairs, as shown in Figure C.3(a). Each bin shows the log frequency of neighboring PIT value pairs $\{\phi_{i-1,j}, \phi_{i,j}\}$. Again, because a perfect model should produce a random set of PIT values, Figure C.3(a) provides evidence that the model is imperfect. The grid shows substantial correlation between the neighboring PIT values; if the model were perfect there would be no such correlation.

Knowing that the model is imperfect provides a motivation to attempt to update it using the PITA idea. Doing so is not very difficult. In this example we have used four



(a) (b)

Figure C.3: These grids show the log frequency of PIT value pairs $\{\phi_{i-1,j}, \phi_{i,j}\}$. Figure (a) shows the PIT values produced using the trained MRF, while (b) corresponds to the updated MRF+PITA model. A perfect model should produce a uniform grid (but a uniform grid does not imply a perfect model).

context functions that depend on the value of the previous PIT value:

$$w_r(C_{i,j}) = \begin{cases} 1 & : T_{r-1} \leq \phi_{i-1,j} < T_r \\ 0 & \end{cases} \tag{C.5}$$

Where $1 \leq r \leq 4$ and $\{T_0 = 0, T_1 = .25, T_2 = .5, T_3 = .75, T_4 = 1\}$. Note that defining the context function in terms of the previous PIT value introduces some technical complications when attempting to do the actual compression, but is completely legitimate from a inferential standpoint. The PIT value histograms resulting from the use of these context functions are shown in Figure C.4.

The histograms of Figure C.4 are then used to update the model ensemble generated by the MRF. The result is a layered MRF+PITA model. The codelength achieved by the new model is $2.8 \cdot 10^6$ bits, corresponding to a savings of about 400,000 bits.

After the update is applied, new versions of the PIT value image and the PIT value grid are produced. These are shown in Figure C.2(b) and Figure C.3(b). The PIT value image is clearly much more random as a result of the update, implying an improved model. Of course, it is still not completely random - faint outlines of the shapes and edges can be seen, implying that there is still room to improve the model. The problem now is that it is difficult to write down simple context functions that will reveal the

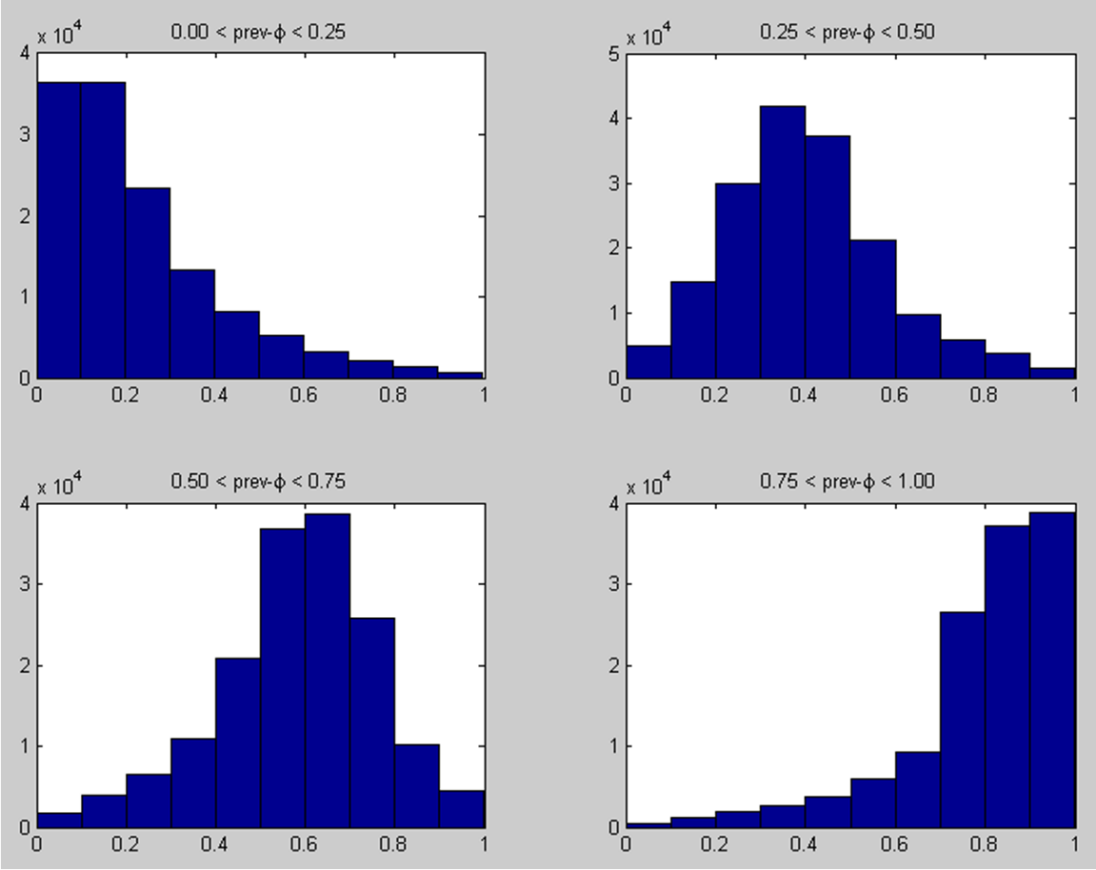


Figure C.4: PIT value histograms corresponding to four context functions that depend on the value of the previous PIT value $\phi_{i-1,j}$. The unevenness in these histograms indicates that they can be used to improve the model ensembles.

randomness deficiencies in the image. Also, the PIT value grid of Figure C.3(b) is much more uniform.

C.4 Summary

The purpose of this Appendix was to illustrate an example of updating a complex model using the PITA idea. In this example, an MRF model was trained using an image as the data set. The trained model involved many parameters. Because the MRF assigns a different observation probability to each pixel depending on the state occupation probabilities, there is no way to visualize the fit of the model to the pixel data. Furthermore, while the trained MRF was good compared to other MRF models, it is difficult to know if the model is good in an absolute sense.

By visualizing the PIT values, and noting that the distribution was obviously non-random, it was easy to see that the MRF model was in fact quite imperfect. By applying a few simple PITA updates, an improved model was obtained. This MRF+PITA model resulted in a savings of $4 \cdot 10^5$ bits compared to the previous one.

The point of the example is not specifically related to MRF image modeling. It is not clear if the base MRF configuration used in the example is the "optimal" or state of the art type, or if such a designation would even be meaningful. The situation that arises when attempting to model an image with a trained MRF is a very typical situation in statistical modeling. A complex model is trained on the complex data set, and it achieves some log-likelihood. But there is no way to know if the model is actually good.

The PITA idea is thus a "win-win" proposition. If the analysis of PIT values shows that they are random, this provides evidence that the model is good. If randomness deficiencies are found, this indicates a way to improve the model.

References

ANGUS, J. (1994). The probability integral transform and related results. *SIAM Review*, **36**, 652–654. 9, 20, 80

ASLAM, J. (2000). Improving algorithms for boosting. In *Proc. 13th Annu. Conference on Comput. Learning Theory*, 200–207, Morgan Kaufmann, San Francisco. 75

ASUNCION, A. & NEWMAN, D. (2007). UCI machine learning repository. 97

BARLOW, H.B. (2001). Redundancy reduction revisited. *Network: Computation in Neural Systems*, **12**, 241–253. 141

BERGER, A.L., PIETRA, S.D. & PIETRA, V.J.D. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, **22**, 39–71. 70

BILMES, J. (1998). A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. *International Computer Science Institute*, **4**. 126

BOUMAN, C. & SHAPIRO, M. (1994). A multiscale random field model for Bayesian image segmentation. *IEEE Transactions on Image Processing*, **3**, 162–177. 150

BURFOOT, D. & KUNIYOSHI, Y. (2009). Compression rate methodology for pure empirical vision science. In *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition*, 519–525. 17, 91

BURFOOT, D., LUNGARELLA, M. & KUNIYOSHI, Y. (2008). Toward a theory of embodied statistical learning. In *SAB: Proceedings of the 10th International Conference on Simulation of Adaptive Behavrior*, 270–279. 17

CHERNOFF, H. & LEHMANN, E. (1954). The use of maximum likelihood estimates in χ^2 tests for goodness of fit. *The Annals of Mathematical Statistics*, 579–586. 78

CHO, Y. & KONDOZ, A. (2001). Analysis and improvement of a statistical model-based voiceactivity detector. *IEEE Signal Processing Letters*, **8**, 276–278. 115

COVER, T.M. & THOMAS, J.A. (1991). *Elements of Information Theory*. New York: Wiley. 143

DARROCH, J.N. & RATCLIFF, D. (1972). Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, **43**, 1470–1480. 41, 70

DAVENPORT JR, W. (1952). An Experimental Study of Speech-Wave Probability Distributions. *The Journal of the Acoustical Society of America*, **24**, 390. 114

DENG, H. & CLAUSI, D. (2004). Unsupervised image segmentation using a simple MRF model with a new implementation scheme. *Pattern Recognition*, **37**, 2323–2335. 150

DEVROYE, L. (1986). *Non-uniform random variate generation*. New York: Springer-Verlag. 21

DIEBOLD, F., GUNTHER, T. & TAY, A. (1998). Evaluating density forecasts with applications to financial risk management. *International Economic Review*, **39**, 863–883. 82

DIETTERICH, T. (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, **40**, 139–157. 97

ELLIS, D.P.W. (2005). PLP and RASTA (and MFCC, and inversion) in Matlab. Online web resource. 117

EMBRECHTS, P., MCNEIL, A. & STRAUMANN, D. (2002). Correlation and dependence in risk management: properties and pitfalls. *Risk management: value at risk and beyond*, 176–223. 82

FAMA, E. (1970). Efficient capital markets: A review of theory and empirical work. *Journal of finance*, 383–417. 7

FISHER, W., DODDINGTON, G. & GOUDIE-MARSHALL, K. (1986). The DARPA speech recognition research database: specifications and status. In *Proceedings of the DARPA speech recognition workshop*. 118

FREES, E. & VALDEZ, E. (1998). Understanding relationships using copulas. *North American Actuarial Journal*, **2**, 1–25. 82

FREUND, Y. & SCHAPIRE, R. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, **55**, 119–139. 92

FREUND, Y. & SCHAPIRE, R.E. (1996). Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, 148–156. 74, 99

FRIEDMAN, J., HASTIE, T. & TIBSHIRANI, R. (2000). Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 400–407. 75

FUJIMOTO, M. & ISHIZUKA, K. (2008). Noise robust voice activity detection based on switching kalman filter. *IEICE Transactions on Information and Systems E Series D*, **91**. 115

FUJIMOTO, M. & NAKAMURA, S. (2005). Particle Filter Based Non-stationary Noise Tracking for Robust Speech Recognition. In *International Conference on Acoustics, Speech, and Signal Processing*, 257–260. 115

GAZOR, S. & ZHANG, W. (2003a). A soft voice activity detector based on a laplacian-gaussian model. *IEEE Transactions on Speech and Audio Processing*, **11**, 498–505. 116

GAZOR, S. & ZHANG, W. (2003b). Speech probability distribution. *IEEE Signal Processing Letters*, **10**, 204–207. 17, 114, 116

GENEST, C. & RIVEST, L. (1993). Statistical inference procedures for bivariate Archimedean copulas. *Journal of the American Statistical Association*, **88**, 1034–1043. 82

GNEITING, T., BALABDAOUI, F., RAFTERY, A. *et al.* (2007). Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society Series B*, **69**, 243. 82

HUANG, J. & MUMFORD, D. (1999). Statistics of natural images and models. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 541–547. 17

HUFFMAN, D. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, **40**, 1098–1101. 144

INAMURA, T., TOSHIMA, I., TANIE, H. & NAKAMURA, Y. (2004). Embodied symbol emergence based on mimesis theory. *International Journal of Robotics Research*, **23**, 363–377. 17, 123

ISHIZUKA, K. & KATO, H. (2006). A feature for voice activity detection derived from speech analysis with the exponential autoregressive model. In *International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, 789–792. 116

JAYNES, E.T. (1957). Information theory and statistical mechanics. *Phys. Rev.*, **108**, 171–190. 69

KIVINEN, J. & WARMUTH, M. (1999). Boosting as entropy projection. In *Proceedings of the twelfth annual conference on Computational learning theory*, 134–144, ACM New York, NY, USA. 75

KULIC, D., TAKANO, W. & NAKAMURA, Y. (2007). Representability of human motions by factorial hidden markov models. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, 2388–2393. 128

KUNIYOSHI, Y., INABA, M. & INOUE, H. (1994). Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, **10**, 799–822. 123

KUNIYOSHI, Y., YOROZU, Y., INABA.M. & INOUE.H. (2003). From visuo-motor self learning to early imitation – a neural architecture for humanoid learning. In *Proc. of the 2003 Intl. Conf. on Robotics and Automation*, 3132–3139. 123

LAMPERTI, J. (1996). *Probability: A survey of the mathematical theory*. John Wiley & Sons Inc. 77

LEARNED-MILLER, E. & JOHN, W. (2003). ICA using spacings estimates of entropy. *The Journal of Machine Learning Research*, **4**, 1271–1295. 82

LEE, D. & NAKAMURA, Y. (2006). Stochastic model of imitating a new observed motion based on the acquired motion primitives. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4994–5000. 123

LI, M. & VITANYI, P. (1997). *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag. 4, 146

MALOUF, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Sixth Conf. on Natural Language Learning*, 49–55. 70

MANNING, C. & SCHUTZE, H. (2002). *Foundations of statistical natural language processing*. MIT Press. 102

MARTIN-LÖF, P. (1966). The definition of random sequences. *Information and Control*, **9**, 602–619. 4, 147

MCCALLUM, A. (2003). Efficiently inducing features of conditional random fields. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. 72

MEMON, N.D., WU, X., SIPPY, V. & MILLER, G. (1997). Interband coding extension of the new lossless jpeg standard. In *Visual Communications and Image Processing '97*, vol. 3024, 47–58. 89

MEYER, B. & TISCHER, P. (2001). Glicbawls-grey level image compression by adaptive weighted least squares. In *Proc. Data Compression Conference (DCC 2001)*, 503. 86

MORI, T., SHIMOSAKA, M., HARADA, T. & SATO, T. (2004). Informative motion extractor for action recognition with kernel feature alignment. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 2, 2009–2014 vol.2. 122

MURPHY, K. (2005). Hidden Markov Model Toolbox for Matlab. Online web resource. 128

NAKAYAMA, H., HARADA, T. & KUNIYOSHI, Y. (2009). Canonical contextual distance for large-scale image annotation and retrieval. In *LS-MMRM '09: Proceedings of the First ACM workshop on Large-scale multimedia retrieval and mining*, 3–10, ACM, New York, NY, USA. 17

NEMER, E., GOUBRAN, R. & MAHMOUD, S. (2001). Robust voice activity detection using higher-order statistics in the LPC residual domain. *IEEE Transactions on Speech and Audio Processing*, **9**, 217–231. 116

PICONE, J. *et al.* (1993). Signal modeling techniques in speech recognition. *Proceedings of the IEEE*, **81**, 1215–1247. 114, 116

PIETRA, S.D., PIETRA, V.D. & LAFFERTY, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19**, 380–393. 72

RABINER, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, **77**, 257–286. 126

RATNAPARKHI, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, **34**, 151–175. 70

RICHARDS, D. (1964). Statistical properties of speech signals. *Proc. Inst. Elect. Eng.*, **111**, 941–949. 114

RISSANEN, J. (1976). Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, **20**, 198–203. 144

RISSANEN, J. (1978). Modeling by shortest data description. *Automatica*, **14**, 465–471. 4, 34, 45, 91, 93, 103, 115

ROSENFELD, R. (1996). A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, **10**, 187– 228. 17, 70

RUMELHART, D.E., HINTON, G.E. & WILLIAMS, R.J. (1986). Learning internal representations by error propagation. *Nature*, **323**, 533–536. 41

SCHAPIRE, R. (1990). The strength of weak learnability. *Machine learning*, **5**, 197–227. 92

SCHAPIRE, R., FREUND, Y., BARTLETT, P. & LEE, W. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, **26**, 1651–1686. 99

SHANNON, C. (1948). A mathematical theory of communication. *Bell Syst. Tech. Journal*, **27**, 379–423. 3

SHIMOSAKA, M., MORI, T., HARADA, T. & SATO, T. (2004). Action recognition based on kernel machine encoding qualitative prior knowledge. In *2004 IEEE International Conference on Systems, Man and Cybernetics*, vol. 2. 122

SKLAR, A. (1959). Fonctions de répartition à n dimensions et leurs marges. *Publ. Inst. Statist. Univ. Paris*, **8**, 229–231. 82

SOHN, J., KIM, N. & SUNG, W. (1999). A statistical model-based voice activity detection. *IEEE Signal Processing Letters*, **6**, 1–3. 17, 114, 115

VAPNIK, V. (1998). *The Nature of Statistical Learning Theory*. Springer. 115

VAPNIK, V. & GILAD-BACHRACH, R. (2008). Learning has just started: an interview with Professor Vladimir Vapnik. http://www.learningtheory.org/. 18

WEINBERGER, M., SEROUSSI, G. & SAPIRO, G. (2000). The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS. *IEEE Transactions on Image Processing*, **9**, 1309–1324. 86

WITTEN, I.H., NEAL, R.M. & CLEARY, J.G. (1987). Arithmetic coding for data compression. *Commun. ACM*, **30**, 520–540. 85, 144

WU, X. & MEMON, N. (2000). Context-based lossless interband compression-extending CALIC. *IEEE Transactions on Image Processing*, **9**, 994–1001. 86

ZHU, X. (2005). Semi-supervised learning literature survey. Tech. Rep. 1530, Department of Computer Sciences, University of Wisconsin, Madison. 141

ZIV, J. & LEMPEL, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, **24**, 530–536. 85