Can Basic ML Techniques Illuminate Rateless Erasure Codes?

Anjali Gupta
anjali@csail.mit.edu

Maxwell Krohn
krohn@csail.mit.edu

Michael Walfish
mwalfish@csail.mit.edu

Abstract

The recently developed *rateless* erasure codes are a near-optimal channel coding technique that guarantees low overhead and fast decoding. The underlying theory, and current implementations, of these codes assume that a network transmitter encodes according to a pre-specified probability distribution. In this report, we use basic Machine Learning techniques to try to understand what happens when this assumption is false. We train several classes of models using certain features that describe the empirical distribution realized at a network receiver, and we investigate whether these models can "learn" to predict whether a given encoding will require extra overhead. Our results are mixed.

1 Introduction

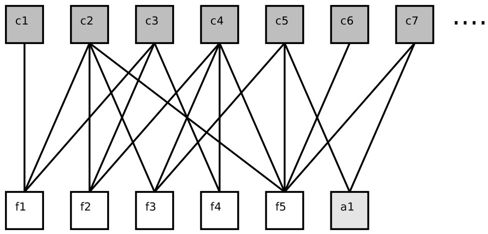
Rateless erasure codes [9, 11, 16] have several remarkable properties. First, encoding and decoding are extremely rapid, in some schemes linear in the size of the file. Moreover, using rateless codes, a file of n input symbols is transformed into a practically infinite set of output symbols, and meanwhile *any* $(1 + o)n$ of these output symbols—where o is typically only several percentage points—are sufficient to reconstruct the original file. In contrast, other erasure coding schemes (*e.g.*, Tornado Codes [10]) associate a file of n input symbols to a particular set of output symbols that may number, for example, $2n$.

In spite of their relatively recent introduction, rateless codes have already been adopted in a variety of academic, commercial and file-sharing systems [1, 4, 6]. They are particularly useful for decentralized, many-to-many large file transfers: many sources can send to many receivers concurrently simply by encoding a given file with rateless erasure codes. This scheme requires neither upstream feedback, nor retransmission of dropped packets, nor continuous communication with any particular sender; the approach is thus practical, efficient, and scalable. Among other systems, SplitStream [1] and Bullet [6] use this method.

The process by which a rateless encoder creates an output symbol is almost bafflingly simple: the encoder randomly selects a weight w from probability distribution P^* (where P^* depends on the particular erasure coding scheme), then selects w random input symbols, and then sends the XOR of these input symbols over the network. (The decoder runs a simple and intuitive algorithm to reconstruct the original file.) The remarkable properties mentioned above depend on the sender following P^* when selecting the weights of output symbols.

A natural question, particularly in a peer-to-peer context in which senders may be hostile, is what happens to the required overhead that a client must receive if a sender deviates from P^*.[1] For a given deviation, $(1 + o)n$ might become, for example, an unacceptable $2.00n$ instead of the correct $1.05n$. Ultimately, we would like an on-the-fly algorithm for receivers that takes as input only a small set of received output symbols and determines whether a given sender's output symbols will lead to a value of o that is outside some acceptable tolerance. In practice, the receiver could use such an algorithm to blacklist a given malicious sender or to make an informed decision about whether to continue spending bandwidth and time in the face of an attack specifically designed to squander them.

[1] A recently proposed scheme thwarts the more obvious attack in which a malicious sender creates improper output symbols using random data, not derived from the file at all. This scheme replaces XOR with an operation that allows receivers to check received content against known cryptographic hashes [7]. As a result, in this work, we assume that receivers always get data from the correct file and that the adversary's attack mode is to deviate from P^*.



(a) Graph

$$X = \left(\begin{array}{ccccc|c}
1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)$$

(b) Adjacency Matrix

Figure 1: Two representations of the preprocessed file $\mathbf{f}' = (f_1, \ldots, f_5, a_1)$, and a derived encoding $\mathbf{c} = (c_1, \ldots, c_7) = X\mathbf{f}'$. The auxiliary block a_1 is added to the original file $\mathbf{f}$ during the preprocessing phase.

The purpose of this report is to use basic Machine Learning techniques to take a step toward such an algorithm. We attempt to train simple statistical models to predict, based on a subset of received output symbols, whether the decoding process will require extra overhead. The rest of this report concerns the development, application, validation, and analysis of such models. These models are not general-purpose or universal in any sense. Rather, like the work as a whole, the models should be considered suggestive and exploratory. We hope to demonstrate only that Machine Learning techniques may yield insight into the complex problem of inferring required decoding overhead.

2 Approach

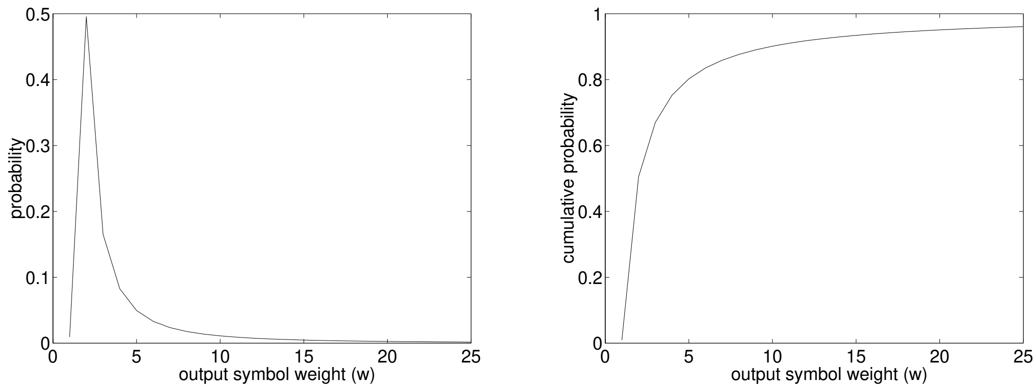
We believe that our approach in principle applies to any erasure coding scheme; due to time constraints, we examined only Online Codes [11, 12].

2.1 Online Codes: Background and Terminology

Figure 1(a) graphically depicts a sample Online Encoding of a very small file. The encoding decomposes a file $\mathbf{f}$ into a series of b-bit input symbols, also called *message blocks*, and applies a preprocessing step to add a small amount of redundancy, called *auxiliary blocks*, to the input symbols; we assume that both sender and receiver in advance know how the auxiliary blocks relate to the input symbols. The result of this step is an n'-block preprocessed file $\mathbf{f}'$. Maymounkov [11] suggests setting $n' = (1 + 2\delta)n$, where δ is the probability decoding will fail. For $\delta = .005$, this means that preprocessing expands the input file $\mathbf{f}$ by 1%. To obtain output symbols, also called *check blocks*, the encoder XORs combinations of message blocks and auxiliary blocks, according to a process we describe below.

Figure 1(b) illustrates an equivalent adjacency matrix representation of the encoding process; in this report, we call such a matrix an *encoding matrix.* Each row of the encoding matrix represents an output symbol that is the XOR of the input symbols whose corresponding cells in the row are set to 1. The encoding matrix has dimension $m \times n'$, where m is greater than or equal to the number of output symbols necessary to reconstruct the file. To create a row of X, the encoder first randomly samples a *weight* w from the probability distribution P^*, defined below. The encoder then uniformly and at random selects w cells of the row to set to one, and all other cells remain zero. Finally, the encoder computes the matrix product $X\mathbf{f}'$ (one may view the cells of X as elements in $\mathbb{F}_2$ and $\mathbf{f}$ as a vector in $\mathbb{F}_2^b$ so that addition of elements in this field is algebraically equivalent to XOR). Receivers getting the ith output symbol thus get a pair, consisting of the ith row of X, and the ith component of the vector $X\mathbf{f}'$. In practice, the foregoing means that meta-data accompanies each output symbol; this meta-data indicates to receivers which input symbols have been XORed to produce the given output symbol.

The distribution of weights P^* is as follows. Let ρ_i be the probability of selecting weight i. Following the theory in [11] and letting F, the maximum possible weight, equal $(\ln \delta + \ln(\epsilon/2))/(\ln(1 - \delta)$, we have



(a) PDF

(b) CDF

Figure 2: The distribution P^*, with parameters $\epsilon = .01$ and $\delta = .005$

that P^* is given by:

$$\rho_1 = 1 - \frac{1 + 1/F}{1 + \epsilon}, \quad \rho_i = \frac{1 - \rho_1}{(1 - 1/F)i(i-1)}, 2 \leq i \leq F, \tag{1}$$

Again following [11], we set $\delta = .005$, $\epsilon = .01$ and thus get $F = 2114$. Figure 2 depicts this distribution. Assuming the sender is encoding according to P^*, once a receiver gets the appropriate number of output symbols, resulting in a matrix X, he can "undo" the effects of applying X to the preprocessed file and can recover the original file **f**. As the size of the file n grows large, the theory predicts that this appropriate number of output symbols approaches $(1 + \epsilon)n'$ from above. In the limit, then, our parameters would yield encodings with approximately 2% overhead. If, however, the sender deviates from P^* and encodes, say, according to some distribution P_D, then the received matrix X might result in an unsuccessful decoding or might require greater than expected overhead, *i.e.*, the number of rows, m, of the matrix would have to be much greater than what is predicted by the theory.

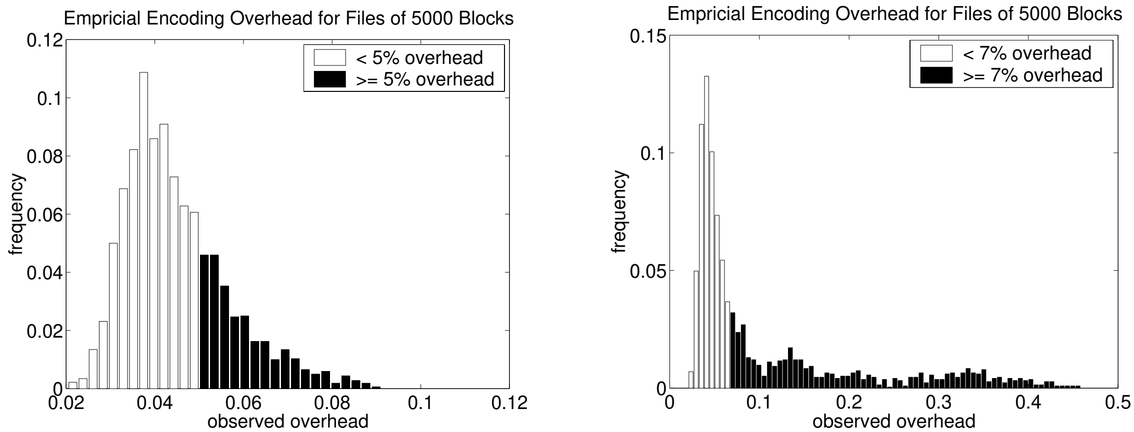
The encoding matrix X represents the essential structure of how a file is encoded, and implicitly, how a file might be decoded. The operations on X at the core of the decoding process are independent of the file **f**, and thus we can safely factor out the contents of particular files. The remainder of this report refers to encodings solely in terms of matrices X. Of course, rateless erasure encoding is a process of sampling from a distribution; there is no well-defined beginning, end, or order to the output symbols associated with a given file. So a given matrix X does not represent the only output symbols associated with the file but rather a particular set of output symbols that happened to make it from the encoder to the receiver across an erasure channel. Since channel drops are independent of the contents of output symbols, the rows of X form an empirical distribution drawn from the actual distribution followed by the encoder.

2.2 Goals

We seek data and accompanying models to help us answer the following question: Is it possible to predict, from the empirical weight distribution on a set of received output symbols, how much overhead decoding requires? At the beginning of our exploration, we noticed that when a good faith encoder follows the correct distribution P^*, the overhead for encodings of 5000 block files ranges from 1.02 to 1.13, a significant range, given that the theory predicts 1.02 (see Figure 3(a)). We therefore split our question:

(1) If a transmitter encodes according to P^*, can we predict the required overhead o, using only features of the empirical distribution of received output symbols?

(2) Alternatively, if the distribution followed by the transmitter is unknown, *i.e.*, it is P^* or some other P_D, can we predict o?



(a) GoodFaith

(b) Mix

Figure 3: Empirical overhead observations.

In both cases, our goal is to train models on features derived from a full empirical distribution, *i.e.*, derived from collections of m rows of X and to have these models be applicable when only a subset of X's rows are available, as would be true if a receiver had received only a fraction of the necessary output symbols.

2.3 Data

Using a custom encoder in MATLAB, we generated a set, GF, of sample "Good Faith" encodings according to P^*. We shall refer to the matrices that represent these encodings as $X_{g;1}, X_{g;2}, \ldots, X_{g;|GF|}$. These matrices have varying degrees of rows and $n' = (1 + 2\delta)5000 = 5050$ columns. We also generated a set, D, of sample "Deviant" encodings according to various distributions, including the uniform distribution, a scaled exponential distribution, a randomly perturbed P^* and combinations of the foregoing. In each case, we endeavored to make the mean of the deviant distribution close to the mean of P^*. (If an adversary departed significantly from the correct mean, a client could easily detect that fact.) We shall refer to the matrices that represent the deviant encodings as $X_{d;1}, X_{d;2}, \ldots, X_{d;|D|}$. Of course we have by no means exhausted the space of possible deviant distributions and doubt such a feat is possible; Section 7 further discusses this point.

Using a custom decoder in MATLAB, we generated, for each matrix X, two outputs: $z(X)$, where z is the amount of overhead actually required to decode, and $y(X)$, where y is a ± 1 label indicating that z is above or below a pre-specified threshold. Although we are interested primarily in whether we can predict $z(X)$, we are also interested in whether models can "learn" to classify based on the y labels; in this case, the model would obviously be making a much coarser distinction. As we describe in Section 3, we derived, for each matrix X, a feature vector $\Phi(X) = [\phi_1(X), \ldots, \phi_k(X)]$ with the intent of building models to predict $y(X)$ and $z(X)$ based on $\Phi(X)$.

We perform our experiments in duplicate, using two separate datasets: GoodFaith and Mix, each of which has designated training and test data. We use the GoodFaith dataset to answer question (1), above, and thus this dataset consists of all matrices $X \in GF$ and associated y and z values. For this dataset, $y = 1$ if $z <= 1.05$ and $y = -1$ if $z > 1.05$. We selected this threshold arbitrarily. GoodFaith has 3200 elements (including training and test data), approximately 24% of which are labeled -1. We use the Mix dataset to answer question (2), above. For balance, we ignore many of the elements of GF and instead use as our data $H \cup D$, where $H \subset GF$. For this dataset, $y = 1$ if $z <= 1.07$ and $y = -1$ otherwise. In this case, 1.07 works reasonably well as a threshold: only 10% of the z values are $\in [1.06, 1.08]$. Mix has 2150 elements; approximately 42% are labeled -1 and approximately 37% are members of H, *i.e.*, were generated from P^*. For both GoodFaith and Mix, 20% of each dataset is training data, and the remainder is test data.

To explore our models' ability to predict overheads given only a subset of encoding symbols, we also considered additional test sets within GoodFaith and Mix; we call these $\mathsf{Trunc}_{\mathsf{GoodFaith}}$ and $\mathsf{Trunc}_{\mathsf{Mix}}$, respectively. To construct $\mathsf{Trunc}_{\mathsf{GoodFaith}}$ (and analogously for $\mathsf{Trunc}_{\mathsf{Mix}}$), we truncate each matrix X in the test data of GoodFaith, keeping only the first $n'/5$ rows. Since, by design, our features ϕ_i are indifferent to the number of rows in an encoding matrix, our existing feature capture code runs without modification. For a given matrix X, the truncated version receives the same y and z values as the original matrix X, which models using a subset of output symbols to predict the eventual result. For convenience, rather than subscripting, we sometimes use the label Trunc. The important point is that $\mathsf{Trunc}_{\mathsf{GoodFaith}}$ and $\mathsf{Trunc}_{\mathsf{Mix}}$ each amount to a second set of test data within their respective dataset.

3 Features

Our goal was to select features to help a model learn characteristics of the ideal distribution, P^*, as well as, perhaps, the fundamental reasons that deviations from P^* result in high decoding overhead. We included characteristics that we as human observers expected to be useful in describing the divergence between probability distributions. To ensure these characteristics would be useful in a linear regression model, we mapped them to metrics such that the metrics would increase or decrease monotonically with changes in perceived divergence. In future work, we intend to look at techniques such as Partial Least Squares [8] that help in deriving a small number of complex "super-features" from a large set of features of varying relevance to the data. After describing our features below, we discuss our models and feature selection process in the next section. The features are categorized as follows:

(1) **Standard metrics of distance between two distributions, specifically the empirical distribution and the ideal, P^*.**

- *χ^2 tests [3, 5]:* The χ^2 goodness-of-fit test is most effective when each cell is equiprobable [3] and has an empirical population of at least five [5]. With these guidelines, we chose seven different bucketing schemes such that each bucket is composed of consecutive weights. In addition, we included χ^2 values computed on portions of the distribution representing the first 95%, 97% and 99% of the total probability mass. We converted the χ^2 statistics to confidence values and used these confidence values as features.
- *Kullback-Liebler distance [2]:* Let $\mathcal{W}$ be the set of weights empirically realized in a given encoding, let P^* be as above and let $\mathcal{P}_E$ be the empirical distribution. Then the K-L distance, a common way to express the divergence between two probability distributions, is defined by $D(\mathcal{P}_E||P^*) = \sum_{w\in\mathcal{W}} \mathcal{P}_E(w) \log \frac{\mathcal{P}_E(w)}{P^*(w)}$.
- *Matusita distance [14]:* Commonly known as the Hellinger distance (for continuous distributions), this distance measure is appropriate for heavy-tailed distributions. Here $\mathcal{W}$ is the set of all possible weights (in our example $\mathcal{W} = \{1, \ldots, 2114\}$). The measure is defined by $M(\mathcal{P}_E, P^*) = (\sum_{w\in\mathcal{W}}(\sqrt{\mathcal{P}_E(w)} - \sqrt{P^*(w)})^2)^{\frac{1}{2}}$, with $\mathcal{P}_E$ and P^* as above.

(2) **Central moments of the empirical weight distribution**: In addition to the mean and variance, the tail and the peak at $w = 2$ are important characteristics of the ideal distribution, P^* (see Figure 2). A heavy tail permits the encoder to select high weight output symbols with non-vanishing probability. The theory predicts that these high weight symbols are key toward the end of the decoding process when only a few input symbols have yet to be decoded—in those cases, having high weight output symbols makes it considerably more likely that the the missing input symbol is "represented".[2] Similarly, the theory predicts that output symbols with $w = 1, 2$ permit the decoder to bootstrap.

Our feature set thus includes *Skewness*, to check the right-skew in the empirical distribution [13], and *Kurtosis*, to verify the peak at $w = 2$ [13]. Let $\mathcal{W}$ be the set of weights realized empirically. Skewness

[2] Luby [9] gives an elegant and clear treatment of the decoding process; while the theory there is slightly different from that used by Online Codes, the principles and intuition are similar.

and Kurtosis are defined as follows:

$$\text{Skew}(\mathcal{P}_E) = \sum_{w \in \mathcal{W}} \mathcal{P}_E(w) \frac{(w - \mu_{\mathcal{P}_E})^3}{\sigma^3_{\mathcal{P}_E}} \qquad \text{Kurt}(\mathcal{P}_E) = \sum_{w \in \mathcal{W}} \mathcal{P}_E(w) \frac{(w - \mu_{\mathcal{P}_E})^4}{\sigma^4_{\mathcal{P}_E}}$$

Our feature set also includes the mean, variance, and third central moment of the empirical weight distribution. To transform these statistics into metrics, we define our features as the magnitudes of offsets from the corresponding statistic of the ideal distribution P^*. We also normalize because, under regularization, relative magnitudes of the features can affect modeling. For example, our feature representing the mean of $\mathcal{P}_E$ will be $|\mu_{\mathcal{P}_E} - \mu_{P^*}|/\mu_{P^*}$.

(3) **Specific characteristics of the weight distribution:**

- *Probability ratios of specific weights:* We have used $|\mathcal{P}_E(w) - P^*(w)|/P^*(w)$ for $w \in \{1, \ldots, 40\}$ as features. We had initially included the probability ratios of the first 20 weights only, based on our belief that deviations in these weights would have the greatest impact on decoding. We included the next 20 weights as we observed one of our models favoring probability ratios of higher weights.
- *Thickness of Tail:* We included five features measuring the thickness of the tail of the empirical distribution. In this case, "tail" means all the probability mass after a given weight w^*, where w^* is chosen to be, for the five features, the 93^{rd}, 95^{th}, 97^{th}, 98^{th} and 99^{th} percentiles under P^*. The difference in probability mass is converted into a feature by normalizing with respect to the expected thickness of the ideal distribution.

(4) **Columnwise Characteristics:** Up to now, we have been concerned with statistics taken over sums of of the rows of the encoding matrix, in attempts to capture information about the weights of the output symbols. But recall that once a row's weight, w, is sampled from the distribution P^*, the encoder must then randomly choose w elements uniformly at random to set to one. We hypothesized that even for good faith encoders, outlying events in this uniform selection process might result in poor encodings.

Working from the encoding distribution P^* over weights w, we can deduce that any cell in the $m \times n'$ encoding matrix should be set to one with probability $E_{P^*}\{w\}/n'$. Since the encoding process independently generates the rows, we can view each column as m independent flips of a biased coin, and hence the column sums are distributed binomially with parameter $p = E_{P^*}\{w\}/n'$. Note that for our setting of the encoding parameters, $E_{P^*}\{w\} \approx 8.17$.

The three features we included are the observed mean, variance and third central moment of the sums of the columns; we again used magnitudes of offsets and normalization, this time with respect to the Binomial Distribution.

Table 1 in the appendix lists all of the features we used to train our models.

4 Predictive Models and Results

Using the data, X, and associated labels y and z, as described in Section 2.3, we trained and tested various predictive and classification models using the features just presented. After discussing our predictive models in this section, namely regularized least squares and forward fitting, we focus on approaches to classification in the next. Recall that the two questions we are trying to answer concern two datasets, GoodFaith and Mix; we accordingly train and test each of our models twice.

4.1 Least Squares Regression

We began by attempting to train a simple least squares regression model for GoodFaith and Mix. As usual, for a given dataset, our input matrix is given by the feature representation of the training data; for example, our GoodFaith input matrix is $\mathbf{X} = [\Phi(X_{g;1})', \ldots, \Phi(X_{g;|GF|})']$. Because we have comparatively many features, $\mathbf{X}$ has many columns, and, as a result, $\mathbf{X}^T\mathbf{X}$ is a singular or near-singular matrix; hence we were not able to compute directly the optimal least square parameters $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{z}$, where $\mathbf{z}$ is the vector of overhead values z for the training data. We address this problem three separate ways: (1) using an L_2 norm penalty to regularize our solution and break the singularity of $\mathbf{X}^T\mathbf{X}$; (2) using L_1 norm penalty regularization; (3) selecting features using as a heuristic the pairwise mutual information between individual features and the actual overhead. We measure the effectiveness of each approach on given samples using "prediction error", which we define as the average error in predicted overhead: $\sum_{\text{sample}} |z - \hat{z}|/n$, where n is the sample size.

Regularization with L_2 norm penalty. We choose to instantiate L_2 regularization by minimizing the following objective with respect to **w**; J depends on a pre-specified regularization parameter, λ:

$$J(\mathbf{w}, \lambda) = (\mathbf{z} - \mathbf{X}\mathbf{w})^T(\mathbf{z} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T\mathbf{w}$$

We obtain our solution by differentiating $J(\mathbf{w}, \lambda)$ with respect to **w** and setting it to 0. Solving for **w**:

$$-\mathbf{X}^T\mathbf{z} + \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = 0 \implies \mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{z}$$

We did not observe good learning with this technique for any dataset. The prediction error of these models on GoodFaith and Mix datasets was almost exactly what the prediction error would be if our prediction was always the mean of the training data. So, essentially the models are learning no more than the mean of **z**. Figure 4 shows the behavior of these two models.

Regularization with L_1 penalty. The average prediction error for the standard test data in GoodFaith and the $\text{Trunc}_{\text{GoodFaith}}$ test data is slightly better than in the L_2 case, above. The features included by the model are a mix of χ^2 statistics (such as 3, 9, 11, 28) and higher-weight probability ratios (such as 64, 71, 79, 81, 83). As in the L_2 case, the prediction error for Mix data is again only slightly better than if the predictor learned the mean z value, only, and nothing about the relationship between the features and z.

Feature Selection with Mutual Information. Using training data, we constructed joint pairwise empirical probability distributions over each individual feature and the associated z values. The features are continuous variables, so we divided each into quantized cells. We experimented with different cell partitions and compiled the results. For the GoodFaith dataset, the features with highest mutual information are the probability ratios of weights from 1-40, with the higher weights yielding higher mutual informations. The mean of the weights distribution and variance in column sums also have high mutual information with the training values z. Our hypothesis is that the high weight probability ratios are informative because higher weight symbols are more likely to encode auxiliary blocks. In turn, auxiliary blocks are crucial to decoding a file to completion. Due to time constraints, we were unfortunately unable to verify the hypothesis for this report and must leave doing so for future work.

Results from the Mix dataset are entirely different. The Matusita distance (feature 35) has the highest mutual information with the observations, z. Next are the various χ^2 features (1-28). All these features are measures of "difference" between the ideal distribution and the empirical distribution realized at the receiver. That measures capturing the entire probability distribution are informative indicates that the shape of the distribution matters everywhere along the distribution. We speculate that deviation anywhere, particularly in the tail (as measured by the Matusita distance) leads to increased decoding overhead.

We select the top 50 features for both GoodFaith and Mix as regressors in a least squares model with L_2 regularization. The advantage of having 50 features (as opposed to 80) is avoiding the singularity of $\mathbf{X}^T\mathbf{X}$; we can thus compute exact least squares parameters and directly compare to results under regularization. For both datasets, no regularization leads to the lowest prediction error. Prediction errors for the standard test data in GoodFaith and the $\text{Trunc}_{\text{GoodFaith}}$ test data are similar to previous cases. We see a slightly improved performance for Mix—the prediction error goes down to 0.022 from 0.03. However, this performance can hardly be considered satisfactory.

4.2 Forward Fitting

Using mutual information as a heuristic for selecting features ignores interaction among the features or the presence of redundant features; in contrast, an iterative technique for feature selection might be able to capture some of these interactions. We therefore test the following iterative forward fitting technique: at each stage of the iteration, we search for the weak regression model $f(X;\theta) = w \cdot \phi_k(X), \theta = \{k, w\}$ that minimizes the previous iteration's residual squared error. Our results for the GoodFaith case are summarized in Figure 6, where the depicted prediction error is, as above, $\sum |\hat{z} - z|/n$. The graph in Figure 6(b) strongly indicates that the model quickly begins to overfit: the test error begins to rise and, also, both the test and training error follow a regular zig-zag pattern suggesting that the model, rather than learning anything new, is simply eliminating noise. The features that appear to reduce the test and training error are the first eight: a measure of the tail probability (29), probability ratios of individual weights (39, 37, 64), and a χ^2 test (3). Forward fitting's performance on the Trunc data is disappointing: apparently the best predictor in this case is simply the guess 1.0453; every successive regressor adds to the prediction error. In this respect, its behavior echoes the least squares models in Section 4.1 which also appear to "learn" the mean.

Like the least squares models, forward fitting performs quite poorly on Mix data. Here the performance is slightly worse than always predicting the mean of the training data. Figure 7 shows the learning progress. That all four column characteristics are included suggests to us that the distribution over column sums affects decoding overhead; we plan to conduct further analysis of this hypothesis in future work.

5 Classification Models and Results

We use logistic regression and boosting to try to learn to classify according to the y labels. We were also curious whether the predictive models from the last section would be useful for classification, and so we "retrofit" those models so that overhead predictions were mapped to labels of ± 1, according to whether the predicted overhead is above or below a threshold, as discussed in Section 2.3.

5.1 Standard Classification Models

Logistic Regression. We used standard logistic regression regularized with both the L_1 and L_2 norms to classify our various datasets. In general, we found both varieties of logistic regression fared poorly when making classification decisions among the samples in GoodFaith. We did not observe much consistency between the features that the L_1 model selected and those that the L_2 model favored. With both regularization penalties, training and test errors on the complete dataset did not drop significantly below 27%. Since a predictor based solely on the *a priori* class probabilities could do just as well, we can assume this classification scheme is modeling little aside from noise. We might conjecture that the GoodFaith dataset is not linearly separable. Our attempts to train the model on quadratic features yielded insignificant performance gains, despite an explosion in required computational cost.

Logistic regression does much better when classifying samples from the Mix dataset. In particular, we note that regularizing with the L_1 norm penalty $\lambda \approx 5$ generalizes well over the truncated dataset, as shown in Figure 10. To arrive at these classifications, the L_1 model favors a mixture of χ^2 parameters (such as features 21,11,10), tail mass metrics (such as 30), and higher-weight pointwise features (such as 50,49,83,82 and 81). By contrast, the L_2 model is not as likely to select χ^2 parameters. This distinction conforms to our intuition. Since the L_1 model needs to "economize" the total number of features it selects, it must select at least some features that encapsulate aggregate information over the entire distribution, namely, the χ^2 parameters. Because the L_2 model has no such constraint, it is free to approximate aggregate features by combining many more pointwise features. Indeed both models seem to require data from all regions of the distribution to make accurate classifications.

Boosting. We used AdaBoost [15] as a combined feature selection and classification technique. We separately train the model for both GoodFaith and Mix, use $\Phi(X)$ as our feature representation and $y(X)$ as the ± 1 labels. Our weak learners are simply decision stumps: $\text{sign}(w_1 \phi_i(X) - w_0)$ with $\theta = \{i, w_1, w_0\}$ and $w_1 \in \{-1, 1\}$. We use two different loss functions: the standard exponential loss function and the logistic loss function $L_l(z) = \log(1 + e^{-z})$. The results for the GoodFaith data are partially summarized in Figure 8. The model learns to predict with almost no test error, and the algorithm selects Features 64-80, only, with a decided preference for the highest numbered features. This preference is consistent with the informative nature of these features, as discussed in Section 4.1.

Although the boosting algorithm learns to classify correctly when presented with the full empirical distribution, it struggles to make any sense out of the truncated data. At 27% classification error, the model performs slightly worse than chance (recall from Section 2.3 that 24% of the elements in GoodFaith are labeled -1). This is not cause for concern, however. The primary purpose of the GoodFaith data is to help us understand what aspects of the *entire* empirical distribution most affect decoding overhead, and, indeed, boosting has been an admirable success in that regard: it successfully discriminates between various encodings in an identical population. We are not surprised that the truncated features cannot capture essential qualities of the empirical distribution. For example, the expected number of weight 40 symbols is $5000 \cdot \rho_{40} = 3.18$; when only 20% of the output symbols have been received, the truncated version of this feature is unlikely to give any information at all.

The results for the Mix data are partially summarized in Figure 9. In this case, for both loss functions, the model learns to predict with 95% accuracy, but all of the learning occurs with the selection of the first feature! The remainder of the features decrease training error but leave test error largely constant, meaning that, conditioned on the first feature having been selected, the subsequent decision stumps are overfitting. The magic feature, the first one selected, is feature number 24, a χ^2 test; its associated decision stump is $w_0 = 2.6 \cdot 10^{-9}$. Since this value is quite small, and since our desired χ^2 features can only be approximated using MATLAB's `gammainc` function which yields feature values as small as 10^{-320}, the quality of this feature might be a numerical artifact and thus suspect. However, the decision stump is not making random decisions; indeed, despite our discomfort with the minute value of w_0, this stump correctly classifies 95% of our test data correctly. Another way of looking at this is as follows: if a numerical artifact has arisen, it is a highly consistent one, and thus we can view our weak learner as "the output of a particular MATLAB function". This decision stump has 15% classification error on the Trunc test data; the logistic models achieved similar rates, after some regularization. Thus, in this case, AdaBoost has discovered a single weak learner that does as well as a full logistic regression model.

5.2 Retrofitting Predictive Models for Classification

Least Squares Regression. We used predictions made by the least squares regression model with L_1 norm penalty to classify test data in our datasets. Overall, we found that the model performed quite poorly. On the GoodFaith data, for both the standard test data and the $\text{Trunc}_{\text{GoodFaith}}$ test data, the model performed no better than chance. While classification error on the Mix test data was a reasonable 11%, the model performed significantly worse than chance on the $\text{Trunc}_{\text{Mix}}$ test data. The poor performance is likely a result of the model's objective; it is not overly penalized for uncertain predictions nor is it sensitive to the demarcation between the ± 1 halfspaces. By contrast, the boosting technique introduces a heavy penalty for a misclassified sample.

Forward Fitting. We found that forward fitting was similarly ineffective. For the GoodFaith dataset, forward fitting performed no better than chance, even on the training data. For the Mix dataset, forward fitting after 15 iterations produced a test classification error of approximately 25% and a classification error on the Trunc test data of 30%, hardly impressive results. This poor performance likely results from the same factors described above for the least squares case.

6 Discussion and Summary

Our first goal in this work, embodied by the GoodFaith dataset, was to understand what aspects of an empirical probability distribution over weights affect decoding overhead. Although our work here is preliminary, the boosting experiment was unambiguous about preferring the high weight probability ratios when doing feature selection. None of our other models was as decisive as the ones derived from AdaBoost, though some, such as the logistic regression model, incorporated similar high weight probability ratios into their favored feature sets. Together, these findings suggest that the encoding overhead is particularly sensitive to the empirical frequencies of high weight symbols. We intend to verify the hypothesis discussed in Section 4.1—that these features are selected because higher weight output symbols are more likely to have auxiliary blocks XORed into them and that auxiliary blocks are crucial for decoding a file using low overhead.

Our second goal in this work was to understand if a receiver could predict, based on a sample of output symbols, what the decoding overhead would be. In response to this question, our predictive models clearly failed. They perform no better, and in some cases worse than, the very weak model $z(X) = 1.05$, which makes a constant prediction. We intend to rectify this failure in future work by revisiting our model choice and our feature choices. We also wish to understand if there are interactions—*e.g.*, second order, third order or something more complex—among the features that could make them informative in groups but ineffective when taken as individuals.

Despite our embarrassing predictive models, our classification schemes are reasonably effective. Logistic regression with L_1 norm regularization generalizes to the truncated Mix test data set with only 7% classification error, and boosting, although unable to generalize to the truncated data, achieves 5% error on the standard test data for the Mix dataset. We admit to disappointment that our models do not select the same, or even similar, sets of features as each other. Our hypothesis for this divergence is that we have not successfully captured the ultimate causes of high overhead in decoding, either because the information exists in interactions between features, which we do not model, or because the information cannot be captured by our current feature set at all.

In conclusion, we are disappointed by our failure to produce quality predictive models but are nonetheless encouraged by the moderate success of some of our classifiers. Our experience leads us to believe that better models, features, and feature selection techniques might result in a decent predictive model.

7 Future Work

Some of the shortcomings and gaps in this work have been imposed on us by time constraints whereas others result from the exploratory nature of this work and from the intermediate nature of our analyses. Following this distinction, we break our future work into two categories: (1) exploring further models, features, and analyses and (2) revising some of our simplifying assumptions.

Performing More Analyses. These are avenues of investigation that time constraints rendered inaccessible: (1) Investigating additional linear model classes, including, *e.g.*, Support Vector Machines; (2) Including higher order features in our models, which are currently limited by their inability to capture interactions among features; (3) Applying *a priori* feature selection techniques, beyond the mutual information heuristic, to avoid having to present a regression model with a number of terms polynomial in the number of features; (4) Using more involved feature selection techniques, such as neural nets, in which the presence or absence of similar features could be learned and (5) Augmenting our feature set in various ways. These augmentations include: (a) adding more features relating to the distribution over input symbols, *e.g.*, capturing whether particular input symbols are underrepresented among the output symbols by including the minimum input symbol frequency as a feature; (b) Including specific features to determine whether the number of auxiliary blocks—which are important, both in theory and in practice—is near the expectation; and (c) Being more systematic in selecting features by, for example, determining which features are closely correlated in practice and then including only one of such a set in a model.

Revising Simplifying Assumptions. First, we assumed that test data was immediately and freely available at the time we trained our models. This assumption conveniently removed the need for cross-validation measures on training data to test our models' effectiveness. We instead directly examined the models' performance on test data. Our justification is that the purpose of this work is to gain an understanding of whether ML techniques would even be useful, not to build a production predictor or classifier. In the future, however, we would have to incorporate cross-validation measures into our process.

Second, we assumed, for lack of an alternative, that we could generate truly adversarial data. This assumption is of course false. Moreover, we (knowingly) ignored an arsenal of encoding deviations, such as sending mostly linearly dependent output symbols but doing so according to the correct weight distribution. Our working assumption is that the various avenues for attack are in some sense orthogonal, and that an eventual complete solution to our problem might involve a combination of independent statistical and algebraic tests. Intuitively, the algebraic properties of the encoding matrix X determine if it contains enough information to be decoded, without regard to practicality, while the statistical properties of X indicate if decoding can happen efficiently.

Our ultimate hope is that by combining a refined version of the work presented here with a scheme that forces adversaries to send linearly independent output symbols, we can, with high probability, provably detect all attacks an encoder could mount.

Acknowledgment

We thank Tommi Jaakkola for useful discussions; this memo was a class project report for his Machine Learning course. Any errors here, either technical or conceptual, are solely the authors' responsibility.

References

[1] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *18th ACM Symposium on Operating Systems Principles (SOSP)*, Bolton's Landing, NY, October 2003.

[2] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[3] R. D'Agostino and M. Stephens. *Goodness-of-Fit Techniques*, pages 64–71. Mark Dekker, Inc, 1986.

[4] Digital Fountain, Inc. `http://www.digitalfountain.com/`.

[5] D. Knuth. *The Art of Computer Programming, Volume 2*, pages 45–52. Addison-Wesley, 1998.

[6] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *18th ACM Symposium on Operating Systems Principles (SOSP)*, Bolton's Landing, NY, October 2003.

[7] M. Krohn and M. Freedman. On the fly verification of erasure-encoded file transfers. IRIS Student Workshop 2003. `http://www.project-iris.net/isw-2003/program.html`.

[8] S. Lambert-Lacroix. Classification using partial least squares using penalized logistic regression. Technical Report TR 0331, IMAG, 2002.

[9] M. Luby. LT codes. In *43rd Annual Symposium on Foundations of Computer Science*, Vancouver, Canada, Nov. 2002.

[10] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 150–159. ACM Press, 1997.

[11] P. Maymounkov. Online codes. Technical Report 2002-833, NYU, November 2002.

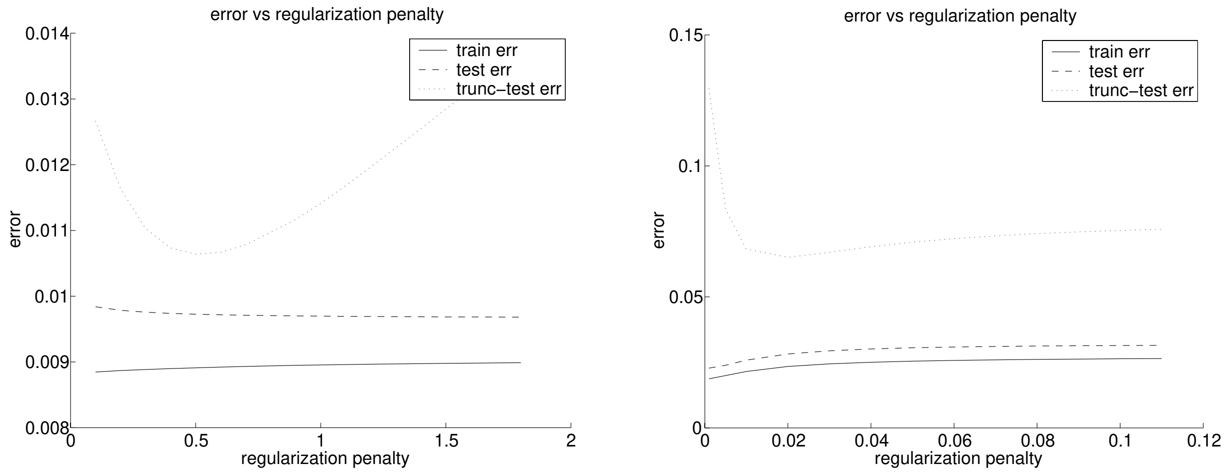
[12] P. Maymounkov and D. Mazières. Rateless codes and big downloads. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, February 2003.

[13] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, pages 604–609. Cambridge University Press, 1992.

[14] T. Read and N. Cressie. *Goodness-of-Fit Statistics for Discrete Multivariate Data*, page 105. Springer-Verlag, 1988.

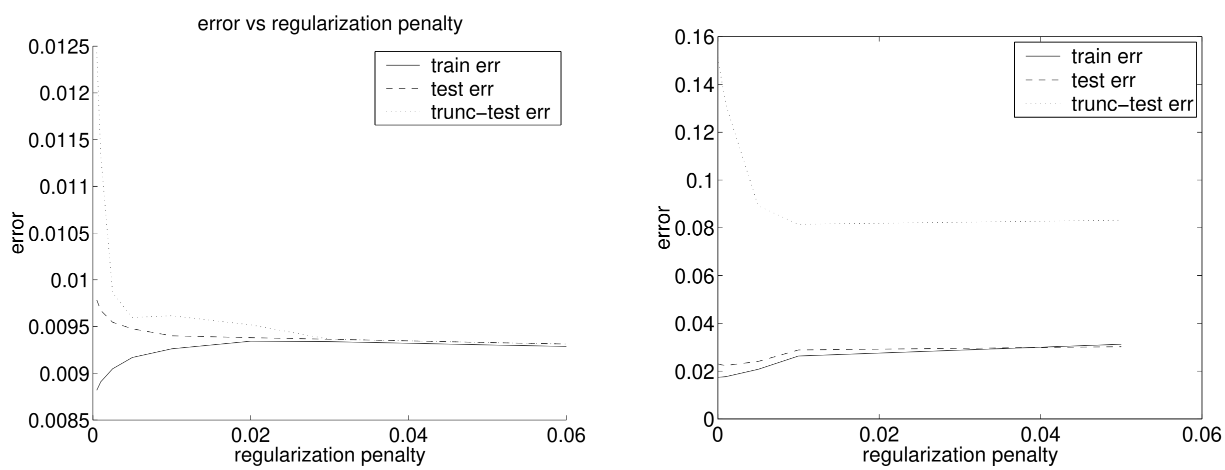
[15] R. E. Schapire. A brief introduction to boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1401–1406, 1999.

[16] A. Shokrollahi. Raptor codes. Technical Report DF2003-06-001, Digital Fountain, Inc., June 2003.



(a) GoodFaith (b) Mix

Figure 4: Results of Least Squares Regression with L_2 norm penalty.

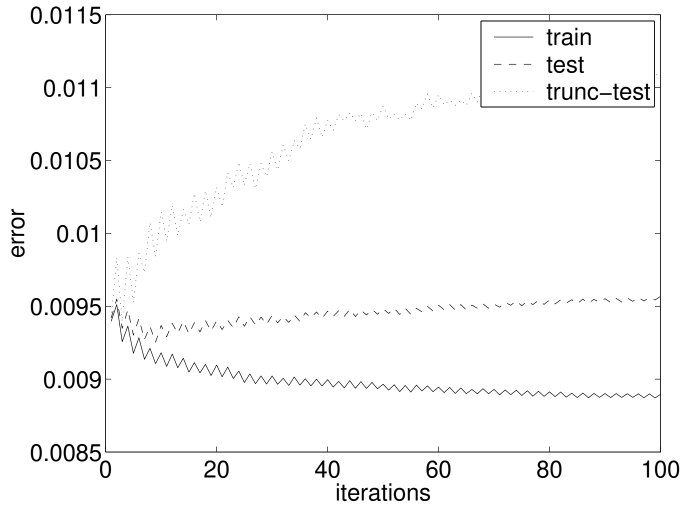


(a) GoodFaith (b) Mix

Figure 5: Results of Least Squares Regression with L_1 norm penalty

feature	w
0	1.0453
29	0.0188
39	-0.0284
37	0.0746
64	-0.0038
29	0.0158
3	-0.0013
37	0.0482
76	-0.0015
79	0.0013

(a)

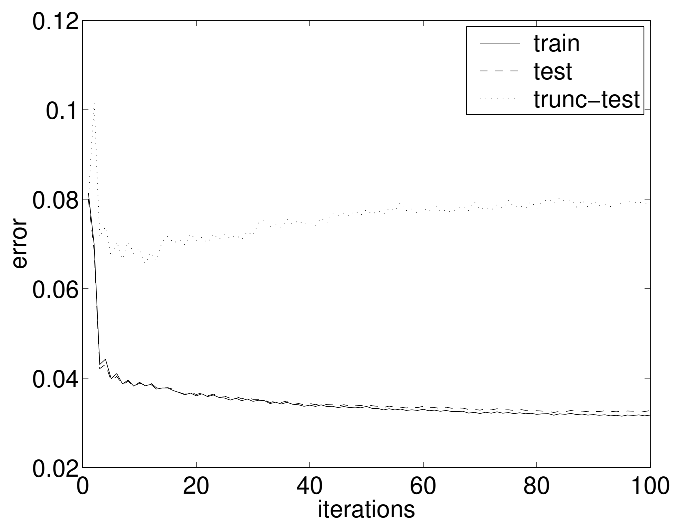


(b)

Figure 6: (a) First 10 weak regressors selected when forward fitting the GoodFaith dataset. (b) Prediction error, defined as the average difference between the predicted overhead and the actual overhead. After the first several regressors, the model is overfitting.

feature	w
0	1.1093
46	0.0971
21	-0.1076
62	0.0503
39	-0.0838
60	0.0172
58	-0.0115
66	0.0143
0	-0.0071
63	0.0260

(a)

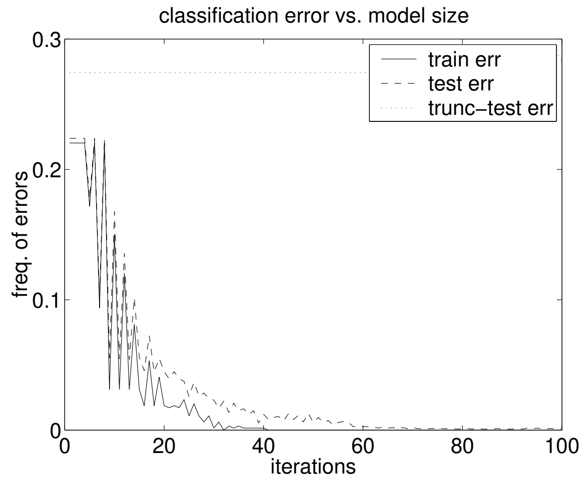


(b)

Figure 7: (a) First 10 weak regressors selected when forward fitting the Mix dataset. (b) Prediction error, defined as the average difference between the predicted overhead and the actual overhead.

feature	alpha	w_1	w_0	iter
76	0.6319	+1	0.0146	1
83	0.3824	-1	0.1006	4
83	0.3812	-1	0.1006	58
83	0.3718	-1	0.4004	54
80	0.3715	-1	0.2320	13
79	0.3714	-1	0.2737	20
83	0.3700	+1	0.1991	38
81	0.3679	+1	0.3505	47
74	0.3524	+1	0.4302	74
77	0.3506	-1	0.1377	27

(a)

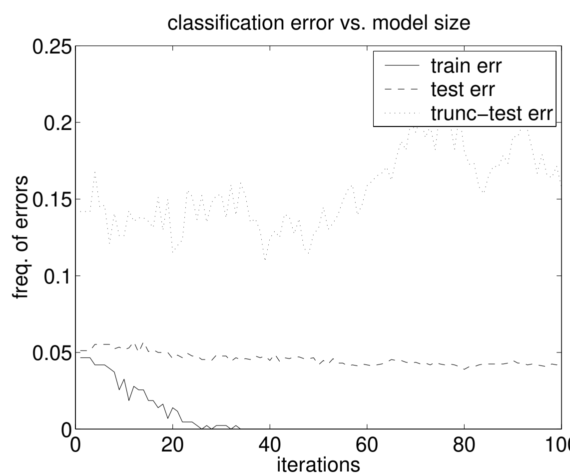


(b)

Figure 8: (a) Top 10 decision stumps selected by AdaBoost on the GoodFaith dataset using exponential loss. The iter field indicates in which iteration the particular stump is selected. Features capturing pointwise deviances for high weight symbols are overwhelmingly favored. (b) AdaBoost can discriminate within the GoodFaith dataset but cannot do better than chance on the Trunc test data.

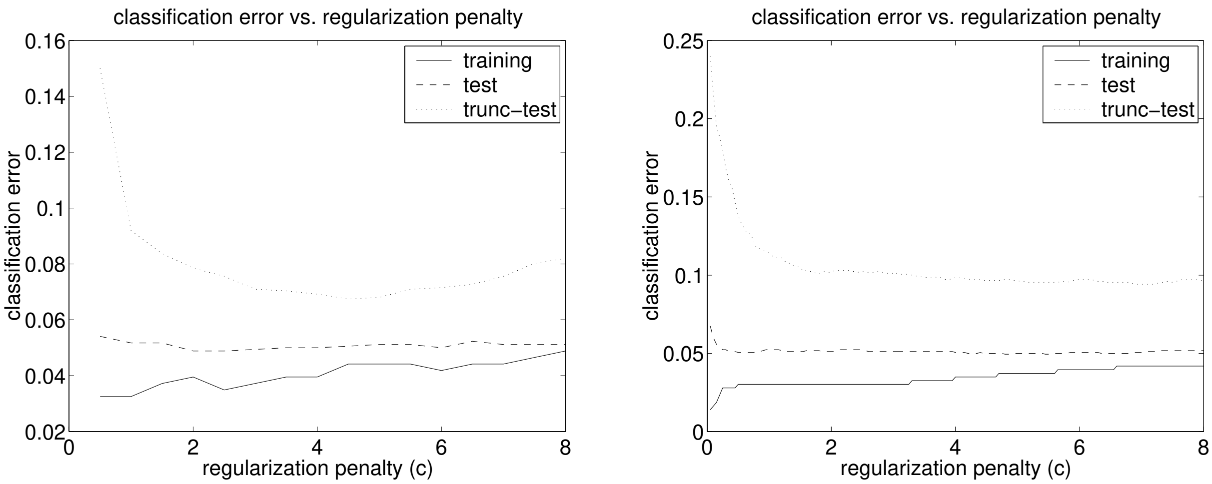
feature	alpha	w_1	w_0	iter
24	1.5102	+1	$\approx 10^{-9}$	1
62	0.5976	-1	0.1411	2
37	0.5636	-1	0.0207	4
66	0.4958	-1	0.2362	3
37	0.4882	-1	0.0433	7
46	0.4771	-1	0.1670	49
73	0.4664	+1	0.1471	9
73	0.4652	-1	0.0155	32
74	0.4583	+1	0.5881	11
83	0.4581	+1	0.4756	5

(a)



(b)

Figure 9: (a) Top 10 decision stumps selected by AdaBoost on the Mix dataset using exponential loss. (b) The first weak learner classifies test and training data with 5% error.



(a) L_1 Regularization

(b) L_2 Regularization

Figure 10: Logistic Regression Results For the Mix Dataset

i	ϕ_i
0	the constant 1
1-7	χ^2 tests, bucketing 7 different ways; using entire distribution
8-14	χ^2 tests, again bucketing 7 different ways; ignoring tail after $F^{-1}(.95)$
15-21	χ^2 tests, again bucketing 7 different ways; ignoring tail after $F^{-1}(.97)$
22-28	χ^2 tests, again bucketing 7 different ways; ignoring tail after $F^{-1}(.99)$
29	difference in tail probability mass, measuring from weight $F^{-1}(.93)$
30	difference in tail probability mass, measuring from weight $F^{-1}(.95)$
31	difference in tail probability mass, measuring from weight $F^{-1}(.97)$
32	difference in tail probability mass, measuring from weight $F^{-1}(.98)$
33	difference in tail probability mass, measuring from weight $F^{-1}(.99)$
34	Kullback-Liebler distance
35	Matusita distance
36-55	pointwise probability difference at weights 1-20 respectively
56	mean (normalized)
57	variance (normalized)
58	third central moment (normalized)
59	kurtosis (normalized)
60	skewness (normalized)
61	mean column sum (normalized)
62	variance of the column sums (normalized)
63	third central moment of the column sums (normalized)
64-83	pointwise probability difference at weights 21-40 respectively

Table 1: List of features