THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCE

ARTIFICIAL PREDICTION MARKETS FOR CLASSIFICATION, REGRESSION AND DENSITY ESTIMATION

By

NATHAN LAY

A Dissertation submitted to the
Department of Scientific Computing
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Degree Awarded:
Spring Semester, 2013

Nathan Lay defended this dissertation on March 29, 2013.

The members of the supervisory committee were:

Adrian Barbu
Professor Directing Thesis

Anke Meyer-Baese
Co-Professor Directing Thesis

Debajyoti Sinha
University Representative

Ye Ming
Committee Member

Xiaoqiang Wang
Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with the university requirements.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Prediction markets are forums of trade where contracts on the future outcomes of events are bought and sold. These contracts reward buyers based on correct predictions and thus give incentive to make accurate predictions. Prediction markets have successfully predicted the outcomes of sporting events, elections, scientific hypothesese, foreign affairs, etc... and have repeatedly demonstrated themselves to be more accurate than individual experts or polling [2]. Since prediction markets are aggregation mechanisms, they have garnered interest in the machine learning community. Artificial prediction markets have been successfully used to solve classification problems [34, 33]. This dissertation explores the underlying optimization problem in the classification market, as presented in [34, 33], proves that it is related to maximum log likelihood, relates the classification market to existing machine learning methods and further extends the idea to regression and density estimation. In addition, the results of empirical experiments are presented on a variety of UCI [25], LIAAD [49] and synthetic data to demonstrate the probability accuracy, prediction accuracy as compared to Random Forest [9] and Implicit Online Learning [32], and the loss function.

CHAPTER 1

INTRODUCTION TO PREDICTION MARKETS

Prediction markets are forums of trade where contracts on the future outcomes of events are bought and sold. Each contract is a wager that yields payment if its corresponding outcome occurs. Each market participant has an incentive to profit and therefore an incentive to predict accurately. The trading prices of contracts are determined by supply and demand where highly demanded contracts are more expensive and represent an overall confidence that a corresponding outcome will be realized. On the other hand, less demanded contracts are less expensive and represent an overall lack of confidence that a corresponding outcome will be realized. These trading prices can be interpreted as the market's prediction of the outcome. Studies have shown that the trading prices even estimate the true probability of the outcome [38]. Prediction markets have found use in predicting elections, decision making in both government and business realms, and even sporting events [2]. Their reported accuracy and success motivated the development of our Classification Market [34, 33, 3] that attempts to mimic a real prediction market in a machine learning setting. The Classification Market has empirically proven to be a competitive classifier aggregation technique and motivates further investigation.

Since the work presented in [34, 33], the classification market has been further developed. We relate the classification market to existing machine learning techniques such as SVM and Logistic Regression, prove that the loss function is the negative log likelihood, and in addition to Random Forest, compare with an alternative online learning method, *Implicit Online Learning* presented in [32] on UCI data sets. We examine the probability estimation capabilities of the classification market with three different betting strategies on synthetic data with increasing Bayes error. We also compare the Classification Market to AdaBoost for a Lymph node detection task and demonstrate improvement over AdaBoost when aggregating its own weak classifiers. In addition to these developments, we also briefly explore an alternative offline market update rule that is revealed through the loss function derivation. We empirically demonstrate the loss function with two update rules and different betting functions. These developments have made it possible to extend and understand the behavior of the market when applied to regression and density estimation tasks.

A related machine learning topic is regression. Where the objective of classification is to predict a label from a discrete set of labels, the objective of regression is to predict a real value in the range of a function. We mathematically develop the analog of the Classification Market, the Regression Market, to deal with real values, or uncountably many "labels".

Regression markets are unusual in that contracts are no longer discrete and finite. Each contract corresponds to a real value prediction and consequently there are uncountably many such contracts for trade. We further show results on UCI and LIAAD data sets that demonstrate that the Regression Market is a viable regressor aggregation technique. We further describe the Regression Market and find the loss function it optimizes. We review the Classification Market for completeness and evolve it into the Regression Market.

The mathematical description of the Regression Market further motivates the extension of prediction markets to density estimation. Where Classification and Regression Markets aggregate classifiers and regressors, a Density Market aggregates densities to estimate some unknown distribution. The prediction market interpretation used to develop the classification and Regression Markets does not readily apply to the density estimation problem. We develop a mathematical description of the Density Market and show preliminary results that the market can effectively *fit* simple 1D and 2D mixture models. We show that the Density Market can learn more complicated distributions and can also be used to solve regression and parameter optimization problems. As with Classification and Regression Markets, we theoretically describe the Density Market and show the loss function it optimizes.

1.1 The Iowa Electronic Market

The majority of this work is based either directly or indirectly on the *Iowa Electronic Market* [53]. The Iowa Electronic Market is a forum where contracts for future outcomes of interest (e.g. presidential elections) are traded.

Contracts are sold for each of the possible outcomes of the event of interest. The contract price fluctuates based on supply and demand. In the Iowa electronic market, a winning contract (that predicted the correct outcome) pays $1 after the outcome is known. Therefore, the contract price will always be between 0 and 1. An example of this market can be seen in figure 1.1.

In the case of classification, our market simulates this behavior, with contracts for all the possible outcomes, paying 1 if that outcome is realized.

1.2 Related Work

This work borrows prediction market ideas from Economics and brings them to Machine Learning for supervised aggregation of classifiers or features in general.

Related work in Economics. Recent work in Economics [38, 40, 41] investigates the information fusion of the prediction markets. However, none of these works aims at using the prediction markets as a tool for learning class probability estimators in a supervised manner.

Some works [40, 41] focus on parimutuel betting mechanisms for combining classifiers. In parimutuel betting contracts are sold for all possible outcomes (classes) and the entire budget (minus fees) is divided between the participants that purchased contracts for the winning outcome. Parimutuel betting has a different way of fusing information than the Iowa prediction market.

The information based decision fusion [40] is a first version of an artificial prediction market. It aggregates classifiers through the parimutuel betting mechanism, using a loop that updates the odds for each outcome and takes updated bets until convergence. This insures a stronger information fusion than without updating the odds. Our work is different in many ways. First our work uses the Iowa electronic market instead of parimutuel betting with odds-updating. Using the Iowa model allowed us to obtain a closed form equation for the market price in some important cases. It also allowed us to relate the market to some existing learning methods. Second, our work presents a multi-class formulation of the prediction markets as opposed to a two-class approach presented in [40]. Third, the analytical market price formulation allowed us to prove that the constant market performs maximum likelihood learning. Finally, our work evaluates the prediction market not only in terms of classification accuracy but also in the accuracy of predicting the exact class conditional probability given the evidence.

Related work in Machine Learning. Implicit online learning [32] presents a generic online learning method that balances between a "conservativeness" term that discourages large changes in the model and a "correctness" term that tries to adapt to the new observation. Instead of using a linear approximation as other online methods do, this approach solves an implicit equation for finding the new model. In this regard, the prediction market also solves an implicit equation at each step for finding the new model, but does not balance two criteria like the implicit online learning method. Instead it performs maximum likelihood estimation, which is consistent and asymptotically optimal. In experiments, we observed that the prediction market obtains significantly smaller misclassification errors on many datasets compared to implicit online learning.

Specialization can be viewed as a type of reject rule [16, 50]. However, instead of having a reject rule for the aggregated classifier, each market participant has his own reject rule to decide on what observations to contribute to the aggregation. ROC-based reject rules [50] could be found for each market participant and used for defining its domain of specialization. Moreover, the market can give an overall reject rule on hopeless instances that fall outside the specialization domain of all participants. No participant will bet for such an instance and this can be detected as an overall rejection of that instance.

If the overall reject option is not desired, one could avoid having instances for which no classifiers bet by including in the market a set of participants that are all the leaves of a number of random trees. This way, by the design of the random trees, it is guaranteed that each instance will fall into at least one leaf, i.e. participant, hence the instance will not be rejected.

A simplified specialization approach is taken in delegated classifiers [24]. A first classifier would decide on the relatively easy instances and would delegate more difficult examples to a second classifier. This approach can be seen as a market with two participants that are not overlapping. The specialization domain of the second participant is defined by the first participant. The market takes a more generic approach where each classifier decides independently on which instances to bet.

The same type of leaves of random trees (i.e. rules) were used by [26] for linear aggregation. However, our work presents a more generic aggregation method through the prediction market, with linear aggregation as a particular case, and we view the rules as one sort of specialized classifiers that only bid in a subdomain of the feature space.

Our earlier work [33] focused only on aggregation of classifiers and did not discuss the connection between the artificial prediction markets and logistic regression, kernel methods and maximum likelihood learning. Moreover, it did not include an experimental comparison with implicit online learning and adaboost.

Two other prediction market mechanisms have been recently proposed in the literature. The first one [14, 13] has the participants entering the market sequentially. Each participant is paid by an entity called the market maker according to a predefined scoring rule. The second prediction market mechanism is the machine learning market [1, 48], dealing with all participants simultaneously. Each market participant purchases contracts for the possible outcomes to maximize its own utility function. The equilibrium price of the contracts is computed by an optimization procedure. Different utility functions result in different forms of the equilibrium price, such as the mean, median, or geometric mean of the participants' beliefs.

1.3 Overview

This section briefly reviews the three learning tasks considered throughout this work. These include, classification, regression and density estimation.

1.3.1 Classification

The objective of the classification problem is to learn a function $f(\mathbf{x})$ on a set of N labeled examples $(\mathbf{x}_n, y_n)_{n=1}^N$ that can *accurately* predict the label of any given unlabeled example. Here $\mathbf{x} \in \mathbb{R}^d$ is a tuple of real numbers, generally referred to as *features*, that numerically describe known information or some *characteristic* of the data. The labels y are discrete values that are used to label a given instance $\mathbf{x}$. These labels may be anything from integers $y \in \{1, 2, \dots, K\}$ that describe the number of rings an abalone has, to text labels such as $y \in \{\text{SPAM}, \text{NOTSPAM}\}$ for classifying emails as spam or not.

Features are often measurements or computed quantities on an otherwise non-numerical object. The `abalone` data set from the UCI repository [25], for example, includes such features as *sex*, *length*, *diameter*, *height*, among other physical measurements of an abalone. In computer vision tasks, the data is usually an image. Features on images are usually computed on the fly at a given image position rather than pre-recorded as in the case of the `abalone` data set. Commonly used features for images include intensity gradients, Haar wavelets [52] and the Histogram of Oriented Gradients [19]. In e-mail spam classification, features are also computed on the fly as e-mails are not numerical in nature. A commonly used feature for this type of problem is the bag-of-words [31]. A bag-of-words feature simply counts the occurrence of individual words in a document completely ignoring the ordering. The assumption in these problems is that one or more features are relevant to the learning problem and can be used to learn a relationship between the features and the labels. Randomized features, for example, cannot be useful for classifying e-mail as spam or predicting the number of rings an abalone has.

Examples of learning methods for classification are too numerous to list in entirety, but include nearest neighbor, Naive Bayes, Support Vector Machine (SVM), Logistic Regression, Neural Networks, Decision Trees and Forests, and Boosting. A description of these methods may be found in [30]. This work makes extensive use of Decision Trees and Decision Forests as described in [9].

The performance of these learning methods are generally measured in a couple ways. The most common measurement is the misclassification rate which is a percentage of the instances that were mislabeled by the classifier. If $f(\mathbf{x})$ is a classifier and $(\mathbf{x}_n, y_n)_{n=1}^N$ are examples, then the misclassification rate is

$$\frac{1}{N}\sum_{n=1}^{N} I(f(\mathbf{x}_n) \neq y_n) \tag{1.1}$$

where $I(\cdot)$ is the indicator function. However, the misclassification rate can be misleading if the data set includes an overwhelming number of examples with a particular class label. For example, if the data set has 1000 negative examples and only 10 positive examples and your classifier correctly labels only negative examples, then your classification rate is ≈ 0.01 which *seems* low. An alternative is the *confusion matrix*. A confusion matrix M is a matrix that counts the number of instances of class label y that are labeled as class label k. That is, each component of the matrix is computed as

$$M_{yk} = \sum_{\mathbf{x} \in X_y} I(f(\mathbf{x}) = k) \tag{1.2}$$

where $X_k = \{\mathbf{x}_n \ : \ y_n = k, \ n = 1, 2, \ldots, N\}$ is the set of examples with label k. A diagonal confusion matrix is ideal. The confusion matrix is more telling about the per-label performance than the misclassification rate especially in the case of such large label disparity.

1.3.2 Regression

Similar to the classification problem, the regression problem is to learn a function $f(\mathbf{x})$ on a set of N examples $(\mathbf{x}_n, y_n)_{n=1}^N$ that can *accurately* predict the real value associated with any given example. Here $\mathbf{x} \in \mathbb{R}^d$ is the feature vector and $y \in \mathbb{R}$ is a real value instead of a discrete value as in classification. For example, the regression task for the `housing` data set from the UCI Machine Learning repository [25] is to estimate the value of a home based on such features as *crime rate*, *proportion of residential zoning*, *number of rooms per dwelling* and so forth. In general, a regression task may predict $y \in \mathbb{R}^d$ instead of just a single real value. Examples of this include [45][18] where the objective is to predict the offset vector $y \in \mathbb{R}^3$ to an object position.

Examples of learning methods for regression tasks include nearest neighbor, linear regression, Multivariate Adaptive Regression Splines (MARS), and Regression Trees and Forests. A description of these methods may be found in [30]. This work makes extensive use of Regression Trees and Regression Forests as described in [9].

The performance of these learning methods can be measured in a couple ways. One way is to average the ℓ_2 residuals on a data set $(\mathbf{x}_n, y_n)_{n=1}^N$

$$\frac{1}{N}\sum_{n=1}^{N}(f(\mathbf{x}_n) - y_n)^2 \tag{1.3}$$

This is also referred to as the Mean Squared Error (MSE). In fact, many regression methods aim to minimize this error directly such as ordinary least squares (OLS). However, this error measurement does not necessarily factor in the noise of the data set. The data itself may be noisy which increases the ℓ_2 residual. Another way to measure the performance of a regressor is through the R^2 coefficient

$$R^2 = 1 - \frac{\frac{1}{N}\sum_{n=1}^{N}(f(\mathbf{x}_n) - y_n)^2}{\mathrm{Var}(Y)} \tag{1.4}$$

where $\mathrm{Var}(Y) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \bar{y})^2$ is the sample variance and $\bar{y} = \frac{1}{N}\sum_{n=1}^{N} y_n$ is the sample mean. This error measurement attempts to quantify the percentage reduction in variance from a constant regressor $f_{\text{constant}}(\mathbf{x}) = \bar{y}$ to the learned regressor $f(\mathbf{x})$. The larger R^2 is, the higher the reduction of variance and the better $f(\mathbf{x})$ fits the data set.

1.3.3 Density Estimation

The density estimation problem is to infer a density $p(\mathbf{x})$ given a set of N points $\mathbf{x}_n,\ n = 1, 2, \ldots, N$. There are two classes of density estimation methods, parametric and nonparametric. Parametric density estimation methods make assumptions about the underlying distributions associated with the examples. An example of a parametric density estimator is a mixture model where each point $\mathbf{x}_n$ is assumed to be associated with one of the constituent mixture distributions. Likewise, nonparametric density estimation makes no underlying assumptions about the distribution of the points. The Kernel Density Estimator is an exmaple of a nonparametric density estimation method [30].

This work briefly relates the prediction market to density estimation and EM. The latter is a method used to infer the weights and parameters of the constituent densities in a mixture model.

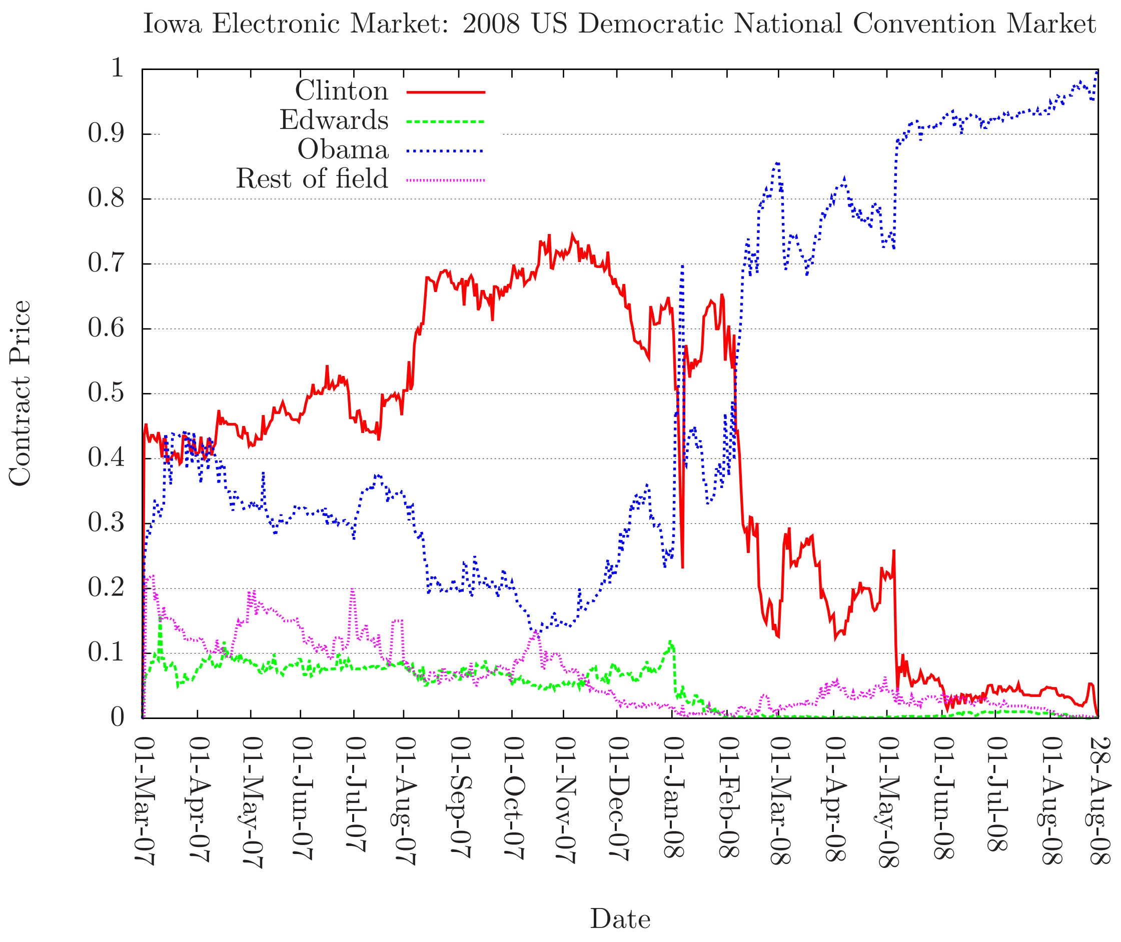


Figure 1.1: 2008 US Democratic National Convention market run by the Iowa Electronic Market. The graph illustrates closing prices for each of the candidates at points in time between March 2007 and August 2008.

CHAPTER 2

RANDOM FOREST

Market specialization and the decision tree play a major role in the development and success of artificial prediction markets. This chapter offers a brief introduction of random forests which are used extensively in this work.

Random Forest is a generic state-of-the-art machine learning framework. It was first described by [9] as a classification and regression learning method. However, it has since been extended to density estimation, online learning, and object detection [18, 45, 27]. It derives its strength and generalization from decision trees and bootstrap aggregation.

2.1 Decision Trees

A decision tree is a generic machine learning framework that describes a model as a tree graph. A tree graph is best thought of as a hierarchy of vertexes. An example would be a family tree. Much like a family tree, decision trees inherit similar terminology. A vertex may have two or more *child* vertexes. Each *child* vertex has a *parent* vertex. A vertex with no *parent* is known as the *root* of the tree. Likewise, a vertex with no *children* is known as a *leaf*. Figure 2.1 illustrates this terminology. In a decision tree, each vertex represents a question or test and the outcome determines the child vertex to visit. This process begins at the *root* vertex and continues until a *leaf* vertex is reached. In addition to a test, a vertex in a decision tree typically stores some estimation. This might be a histogram or a mean value that represents the estimated probability or regression response for a specific sequence of decisions (though it is not limited to this). Upon reaching a leaf vertex, the prediction is taken to be the stored estimate in the leaf.

Two common examples of decision trees include the classification and regression trees (collectively known as CART). The classification tree stores a test and a histogram in each vertex. The test is used to determine which child vertex is visited and the histogram describes the subset of the training sample label distribution that was observed during training. Figure 2.2 illustrates two examples of the test and histogram in each node and their affect on the feature space. The regression tree is similar except that it typically stores the sample mean of y of the subset of the training sample in each vertex that was observed.

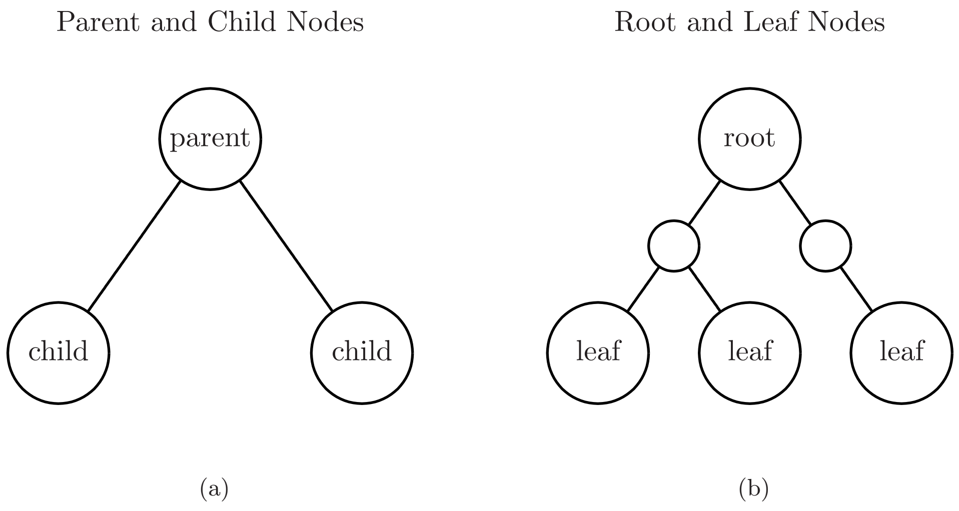


Figure 2.1: Figure (a) illustrates the definition of *parent* and *child* nodes. Figure (b) illustrates the definition of *root* and *leaf* nodes.

2.2 Training

Decision tree *training* is best described as an optimization problem. For some given vertex and some subset of the training sample, the objective is to determine a partitioning of the feature space (typically along one dimension, or feature) so that some cost function is minimized (or maximized) on the training sample. The partitioning then defines the child vertexes. This process begins with one vertex (the *root* vertex) and the entire training sample. Once a partitioning of this sample is determined, then the root grows corresponding child vertexes. The partitions become the training samples for the child vertexes. The process repeats until partitioning can no longer reduce the cost function, reduce it sufficiently, or if the training sample is too small (among other possible reasons). Figure 2.3 provides a simple example of this process for classification. The cost function is the entropy of the class label distribution in each partition. The feature space has been partitioned in such a way that each partition contains only one class label.

2.3 Classification

Decision trees for classification problems typically aim to partition the feature space so that the training sample in each partition is *pure*. A *pure* partition only contains examples with one class label. Purity is often measured in terms of the *Entropy* or *Gini index* [30]

Algorithm 1 Generic decision tree training algorithm.

```
Given data $(\mathbf{X}, Y)$ and features $\{f_i\}_{i=1}^F$. Denote $(\mathbf{X}, Y)_{\text{node}}$ to be the node's subset of data.
The decision tree is trained as follows

  1. Create the root node with $(\mathbf{X}, Y)_{\text{root}} = (\mathbf{X}, Y)$

  2. Compute the *best* splitting feature $f^* \in \{f_i\}_{i=1}^F$ over $(\mathbf{X}, Y)_{\text{node}}$. If the node describes
     the data *perfectly* then growing on the node can cease. Proceed to step 4.

  3. (a) if $\mathbf{x}_{f^*}$ is continuous, create two child nodes with
             $(\mathbf{X}, Y)_{\text{left child}} = \{(\mathbf{x}, y) \in (\mathbf{X}, Y)_{\text{node}} \;:\; \mathbf{x}_{f^*} \leq t\}$
             $(\mathbf{X}, Y)_{\text{right child}} = \{(\mathbf{x}, y) \in (\mathbf{X}, Y)_{\text{node}} \;:\; \mathbf{x}_{f^*} > t\}$

     (b) if $f^*$ is discrete, create $K$ children each with
             $(\mathbf{X}, Y)_{\text{child } k} = \{(\mathbf{x}, y) \in (\mathbf{X}, Y)_{\text{node}} \;:\; \mathbf{x}_{f^*} = k\}$
         where $K$ is the number of values $f^*$ may take.

  4. Pick an available node to grow and return to step 2. If all nodes are fully grown, then
     the algorithm terminates.
```

which are given as

$$H_{\text{Gini}}(Y) = \sum_{k=1}^{K} \frac{|Y_k|}{|Y|}\left(1 - \frac{|Y_k|}{|Y|}\right) \tag{2.1}$$

$$H_{\text{Entropy}}(Y) = -\sum_{k=1}^{K} \frac{|Y_k|}{|Y|} \log\left(\frac{|Y_k|}{|Y|}\right) \tag{2.2}$$

where $Y = \{y_1, y_2, \ldots, y_N\}$ is the set of training labels, Y_k is the set of training labels with label k. When either of these *purity* functions take their minimum value of 0, the training labels are *pure* (they exhibit one class label). Figure 2.4 gives an example of $H_{\text{Entropy}}(Y)$.

When partitioning, the objective is to determine a test $I(\mathbf{x}) = \{0, 1, \ldots, K-1\}$ that minimizes the weighted average of the purities in each partition. Here the test function $I(\mathbf{x})$ assign an instance $\mathbf{x}$ to a partition. When a feature is a real value, the test is often taken to be a threshold test admitting only two results $I(\mathbf{x}) = \{0, 1\}$ and therefore two partitions. When features are nominal, they can have more than two distinct values and the test function $I(\mathbf{x}) = \{0, 1, \ldots, K-1\}$ can produce more than two values. However, multiple threshold tests can be used to partition the training sample in the same way. The remainder of this chapter deals exclusively with binary tests and therefore binary decision trees. When considering only binary tests, the objective is to minimize the weighted average of the purities in each of thw two partitions

$$\frac{|Y_{I(X)=0}|}{|Y|} H(Y_{I(X)=0}) + \frac{|Y_{I(X)=1}|}{|Y|} H(Y_{I(X)=1}) \tag{2.3}$$

where $Y_{I(X)=0} = \{y : (\mathbf{x}, y) \in (X, Y) \wedge I(\mathbf{x}) = 0\}$ and $Y_{I(X)=1}$ is similarly defined. Typically, the binary test $I(\mathbf{x})$ is taken to be a threshold test on a single feature $I(\mathbf{x}) = I(\mathbf{x}_f > t)$ where f denotes the component index in the feature vector $\mathbf{x}$. To account for the purity of the entire training sample, the above is often recasted as a maximization of the *information gain*, given as

$$\mathrm{IG}(Y) = H(Y) - \frac{|Y_{I(X)=0}|}{|Y|} H(Y_{I(X)=0}) - \frac{|Y_{I(X)=1}|}{|Y|} H(Y_{I(X)=1}) \tag{2.4}$$

If the training sample (X, Y) is already pure, then any partitioning of this training sample minimizes the purities in each partition which is not very useful. The information gain is useful in such extreme cases as it implicitly compares the purities of the partitions to the purity of the entire training sample. If there is no average reduction of purity, then there is little to *gain* in partitioning the training sample. Figure 2.5 provides an example of two different splits and their corresponding information gain.

2.4 Regression

In regression trees, the estimator is typically chosen to be the sample mean of the responses. The objective in partitioning the training sample is to minimize the average residual (for example ℓ_2 residual) in each partition between the estimated response and the sample responses. Figure 2.6 illustrates a simple regression tree and its partitions with simple x axis features.

$$\ell(Y) = \frac{|Y_{I(X)=0}|}{|Y|} \frac{1}{|Y_{I(X)=0}|} \sum_{y \in Y_{I(X)=0}} (y - \bar{y}_0)^2 + \frac{|Y_{I(X)=1}|}{|Y|} \frac{1}{|Y_{I(X)=1}|} \sum_{y \in Y_{I(X)=1}} (y - \bar{y}_1)^2 \tag{2.5}$$

where $\bar{y}_0 = \frac{1}{|Y_{I(X)=0}|} \sum_{y \in Y_{I(X)=0}} y$ and $\bar{y}_1$ is similarly defined. This is equivalent to minimizing the average of the partition variances

$$\ell(Y) = \frac{|Y_{I(X)=0}|}{|Y|} \mathrm{Var}(Y_{I(X)=0}) + \frac{|Y_{I(X)=1}|}{|Y|} \mathrm{Var}(Y_{I(X)=1}) \tag{2.6}$$

As with information gain, the variance of the entire training sample should be considered. If, for example, the variance is 0 on the training sample, then any partitioning of the training sample gives an average variance of 0. One measure that factors in the training sample variance is the R^2 coefficient, given as

$$R^2 = 1 - \frac{\frac{|Y_{I(X)=0}|}{|Y|} \mathrm{Var}(Y_{I(X)=0}) + \frac{|Y_{I(X)=1}|}{|Y|} \mathrm{Var}(Y_{I(X)=1})}{\mathrm{Var}(Y)} \tag{2.7}$$

This describes the amount of *reduction* of the partition variance to the variance on the entire training sample. A partitioning that results in only a *small* reduction of variance is not very useful even though the partitioning may already be optimal (e.g. an average partition variance of 0).

2.5 Random Tree

Much of the success of random forest derives from bootstrap aggregation of unstable learners. [11]. Decision trees are *stable* learners in the sense that they change little, if at all, for small changes in the training set. Random trees are *unstable* variants of decision trees. Random tree learning is nearly identical to decision tree learning except that at each vertex, only a random subset of the binary tests are considered for the splitting criteria. For even the same training set, two random trees are different hypotheses of the learning problem.

2.6 Random Forest

The Random Forest is merely a collection of random trees. However, these random trees are trained on bootstrap samples, as in bootstrap aggregation [11]. Bootstrap aggregation of unstable learners can improve the overall accuracy over any individual learner. The aggregation of these trees, for at least CART, amounts to an average of these trees.

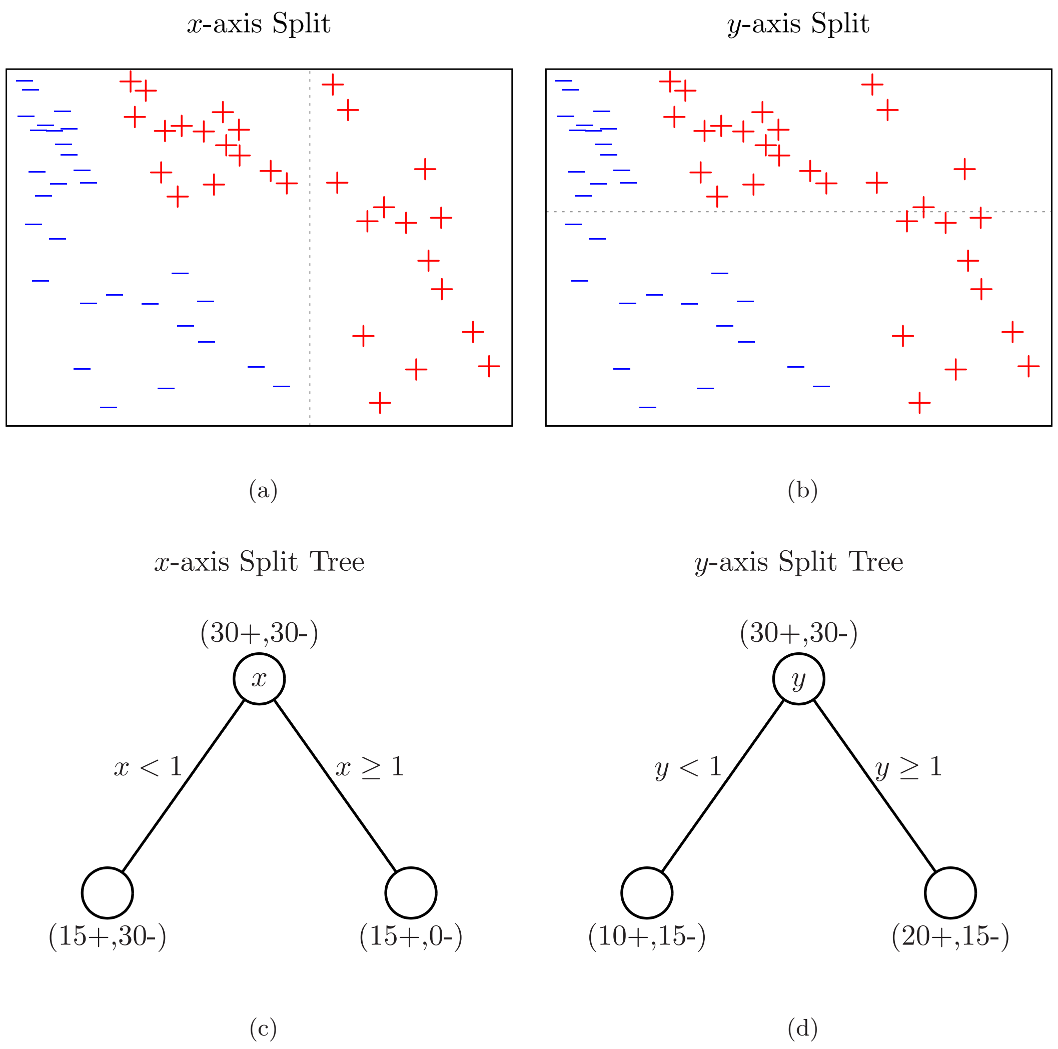
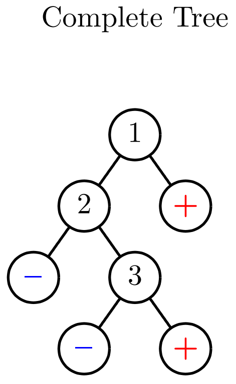
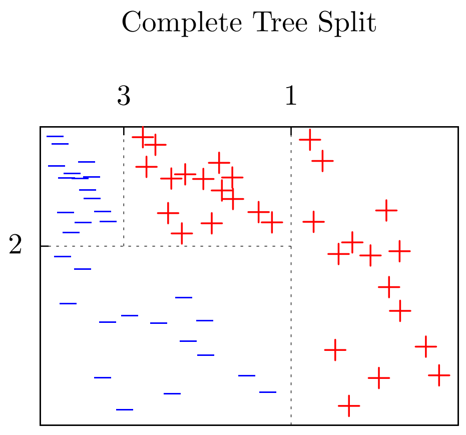


Figure 2.2: These figures are examples of splits on features x and y. Figures (a) and (b) visually depict the splits while figures (c) and (d) show their corresponding tree representation respectively. The $(n+, m-)$ notation means that there are n positives and m negatives in the region.



(a)



(b)

Figure 2.3: These figures demonstrate the tree structure (a) and feature partitions (b) if the training process continued in figure 2.2. The numbered nodes in figure (a) correspond to the region splits in figure (b).

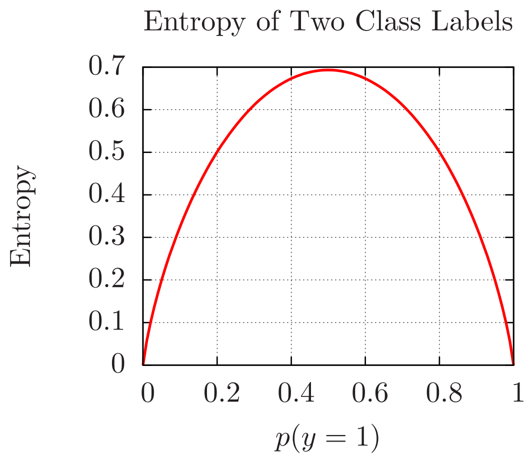
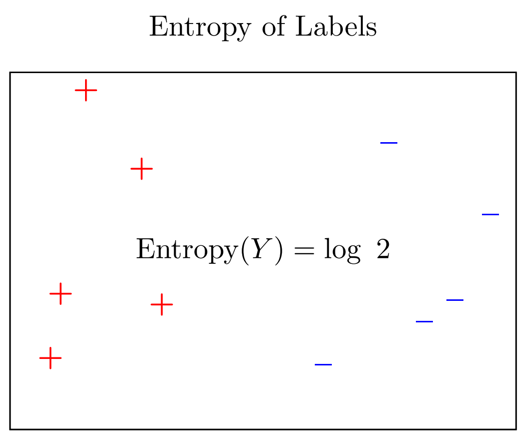
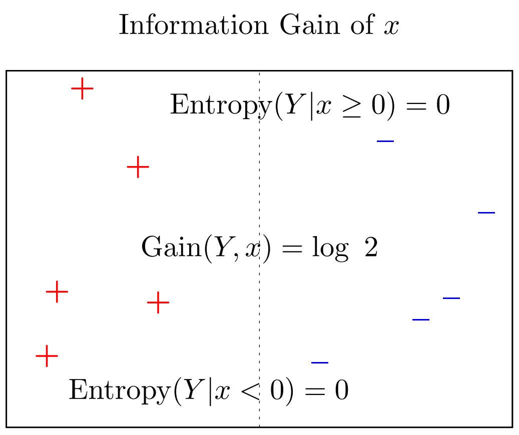


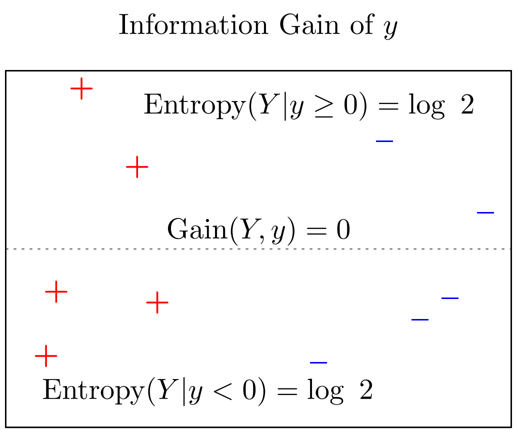
Figure 2.4: This figure demonstrates $H_{\text{Entropy}}(Y)$ in two class labels as a function of the frequency of label $y = 1$. The frequency $p(y = 1) = 0.5$ implies that predicting the label is equivalent to randomly guessing the label and corresponds to maximum entropy.



(a)

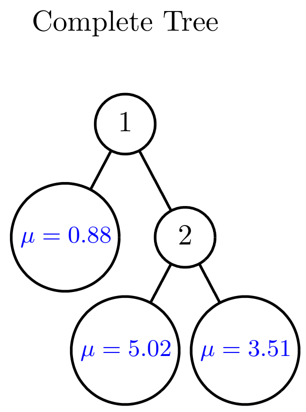


(b)

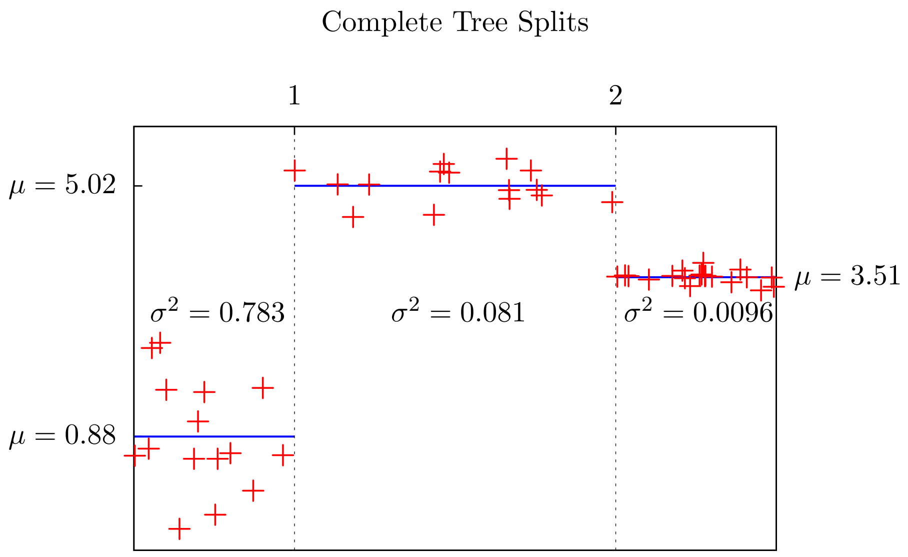


(c)

Figure 2.5: These figures are examples of computing the information gain on features x and y. There are five positives and five negatives. Figure (a) computes the entropy over all the labels Entropy$(Y) = -\frac{5}{10}\log\ \frac{5}{10} - \frac{5}{10}\log\ \frac{5}{10} = \log\ 2$. The bottom two figures split the data in x and y and compute regional entropy and resulting information gain. For figure (b) Entropy$(Y|x < 0) = -\frac{5}{5}\log\ \frac{5}{5} - 0 = 0$ and Entropy$(Y|x \geq 0) = 0 - \frac{5}{5}\log\ \frac{5}{5} = 0$ which gives Gain$(Y, x) = \log\ 2 - 0 - 0 = \log\ 2$. For figure (c) Entropy$(Y|y < 0) = -\frac{3}{6}\log\ \frac{3}{6} - \frac{3}{6}\log\ \frac{3}{6} = \log\ 2$ and Entropy$(Y|y \geq 0) = -\frac{2}{4}\log\ \frac{2}{4} - \frac{2}{4}\log\ \frac{2}{4} = \log\ 2$ which gives Gain$(Y, y) = \log\ 2 - \frac{6}{10}\log\ 2 - \frac{4}{10}\log\ 2 = 0$. Figure (b) divides the labels perfectly and this corresponds to the larger information gain.



(a)



(b)

Figure 2.6: These figures demonstrate the tree structure (a) and feature partitions (b) for a toy regression problem. The numbered nodes in figure (a) correspond to the region splits in figure (b). Each leaf stores the mean y value to predict in its partition. The σ^2 values are the sample variances of the y values in each corresponding partition.

CHAPTER 3

PREDICTION MARKETS FOR CLASSIFICATION

Prediction markets are forums of trade where contracts on the future outcomes of events are bought and sold. An event might be an election or sporting event and an outcome might be the result of the election or winning team respectively. Each contract pays some value if its outcome is realized. The incentive to profit drives the incentive to predict accurately and as a consequence, all information publicly and privately known by market constituents influences the trading prices through demand.

Real prediction markets have been used in the US Department of Defense [43], healthcare [42], to predict the outcomes of presidential elections [53], sporting events (TradeSports), and in large corporations to make informed decisions [17]. They have even been demonstrated to be more accurate than polling methods or individual experts [2]. Some examples include the Iowa Electronic Market and intrade. The Iowa Electronic Market is a research prediction market run by the University of Iowa. In this market, contracts sell for a price $\$0 < c < \$1$ and pay \$1 for correct predictions. This market serves as the model for this work.

From a machine learning perspective, prediction markets are a type of classifier aggregation. The market participants are analogous to classifiers, the event is analogous to the instance, the available information is analogous to the instance features, the outcomes are analogous to the class labels and the trading prices for each outcome are similar to the conditional probabilities for the class labels. It has even been shown that the trading prices of real prediction markets estimate the ground truth conditional probabilities [38, 28]. The claimed accuracy of real prediction markets motivated the development of the classification market [34, 33].

3.1 Problem Setup

Given instances $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}\}$ with $\mathbf{x}_n \in X \subseteq \mathbb{R}^F$, $y_n \in \{1, 2, \ldots, K\}$ and trained classifiers $\{\mathbf{h}_m(\mathbf{x})\}_{m=1}^{M}$ with corresponding budgets β_m where

$$\mathbf{h}_m(\mathbf{x}) = [p_m(y = 1|\mathbf{x}) \quad p_m(y = 2|\mathbf{x}) \quad \ldots \quad p_m(y = K|\mathbf{x})]^T$$

the objective is to compute the trading prices $\mathbf{c} \in \mathbb{R}^K_{\geq 0}$ for each instance and to update the budget β_m of each participant. Since the artificial market is modeled after the Iowa

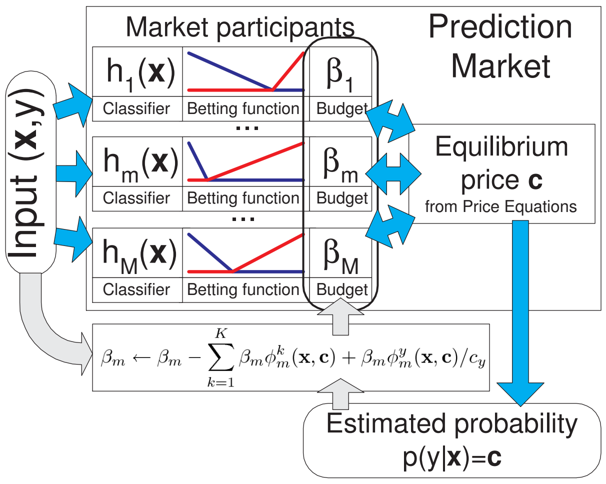


Figure 3.1: Online learning and aggregation using the artificial prediction market. Given feature vector $\mathbf{x}$, a set of market participants will establish the market equilibrium price c, which is an estimator of $P(Y = k|\mathbf{x})$. The equilibrium price is governed by the Price Equations (4). Online training on an example $(\mathbf{x}, y)$ is achieved through Budget Update $(\mathbf{x}, y, c)$ shown with gray arrows.

Electronic Market, the contracts for label k trade at $0 \leq c_k \leq 1$ and pay 1 for correct predictions. We additionally constrain $\sum_{k=1}^{K} c_k = 1$ so that c_k may be interpreted as a probability for class label k.

Since classifiers do not actually *bet*, we introduce the concept of a *betting function* (also known as a *buying function*). While classification is a function of the instance, betting is additionally a function of risk and incentive. For example, even with a relatively high predicted probability of an outcome, a high priced contract may still prove too risky to purchase. Betting functions are defined as follows

$$\boldsymbol{\phi}(\mathbf{x}, \mathbf{c}) \in [0, 1]^K \tag{3.1}$$

$$\sum_{k=1}^{K} \phi^k(\mathbf{x}, \mathbf{c}) \leq 1 \tag{3.2}$$

where $\phi^k(\mathbf{x}, \mathbf{c})$ represents the proportion of the budget to allocate for contracts on class label k. In other words, the amount bet for class label k is $\beta\phi^k(\mathbf{x}, \mathbf{c})$. Based on real-world behavior, we assume betting functions should have the following properties

$$\phi^k(\mathbf{x}, c_k = 0) \geq 0 \tag{3.3}$$

$$\phi^k(\mathbf{x}, c_k = 1) = 0 \tag{3.4}$$

$$\phi^k(\mathbf{x}, c_k + \Delta c_k) \leq \phi^k(\mathbf{x}, c_k) \quad \Delta c_k > 0 \tag{3.5}$$

These properties stem from the following interpretation of betting behavior

- Whenever contracts are free ($c_k = 0$), there is no risk and so market participants bet some non-zero quantity for such contracts.
- Whenever contracts are full price ($c_k = 1$), there is no possible gain and so market participants bet zero for such contracts.
- If contracts are slightly more expensive for label k, then a participant will bet no more for label k than when it is cheaper.

Betting functions may be backed by trained classifiers. In this sense, the trained classifier is the *experience* aspect of the betting function. Some examples of betting functions that follow these properties include [34, 33, 3]

$$\phi^k_{\text{linear}}(\mathbf{x}, c_k) = (1 - c_k)h^k(\mathbf{x}) \tag{3.6}$$

$$\phi^k_{\text{aggressive}}(\mathbf{x}, c_k) = \begin{cases} h^k(\mathbf{x}) & 0 \le c_k \le h^k(\mathbf{x}) - \epsilon \\ -\frac{h^k(\mathbf{x})}{\epsilon}(c_k - h^k(\mathbf{x})) & h^k(\mathbf{x}) - \epsilon \le c_k < h^k(\mathbf{x}) \\ 0 & h^k(\mathbf{x}) \le c_k < 1 \end{cases} \tag{3.7}$$

where $\epsilon > 0$. Others types of betting functions have been explored such as

$$\phi^k_{\text{constant}}(\mathbf{x}, c_k) = h^k(\mathbf{x}) \tag{3.8}$$

$$\phi^k_{\text{logistic}}(\mathbf{x}, c_k) = \begin{cases} c_1(x^+_m - \ln(c_1)/B) & k = 1 \\ c_2(x^-_m - \ln(c_2)/B) & k = 2 \end{cases} \tag{3.9}$$

where $x^+ = xI(x > 0), x^- = xI(x < 0)$ and $B = \sum_{m=1}^{M} \beta_m$. These functions, however, violate (3.3)(3.4)(3.5). Examples of these betting functions can be seen in figure 3.2.

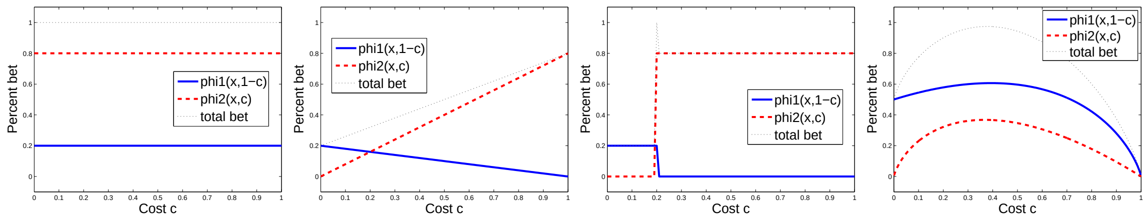


Figure 3.2: Betting function examples: a) Constant, b) Linear, c) Aggressive, d) Logistic. Shown are $\phi^1(\mathbf{x}, 1 - c)$ (red), $\phi^2(\mathbf{x}, c)$ (blue), and the total amount bet $\phi^1(\mathbf{x}, 1 - c) + \phi^2(\mathbf{x}, c)$ (black dotted). For (a) through (c), the classifier probability is $h^2(\mathbf{x}) = 0.2$.

The notion of betting functions also defines the budget update rule and subsequently the equilibrium equation to determine $\mathbf{c}$. The total bet and winnings for a given participant $\phi_m(\mathbf{x}, \mathbf{c})$ are given as follows

$$\text{Total bet} = \beta_m \sum_{k=1}^{K} \phi^k_m(\mathbf{x}, \mathbf{c}) \tag{3.10}$$

$$\text{Winnings} = \beta_m \frac{\phi^y_m(\mathbf{x}, \mathbf{c})}{c_y} = \text{Number of contracts purchased for label y} \tag{3.11}$$

where y is the ground truth class label for instance $\mathbf{x}$. The profit defines the budget update rule for β_m where profit is Winnings − Total bet or

$$\beta_m \leftarrow \beta_m - \beta_m \sum_{k=1}^{k} \phi_m^k(\mathbf{x}, \mathbf{c}) + \beta_m \frac{\phi_m^y(\mathbf{x}, \mathbf{c})}{c_y} \tag{3.12}$$

The update rule can be sensitive to incorrect predictions. For example, if $\boldsymbol{\phi}_m(\mathbf{x}, \mathbf{c}) = \mathbf{h}_m(\mathbf{x})$ and $\mathbf{h}_m^y(\mathbf{x}) = 0$ then the participant becomes immediately bankrupt ($\beta_m = 0$). As a safeguard, the maximum proportion of the budget can be capped to be $\eta\beta_m$ where $0 < \eta \leq 1$. This gives the modified update rule

$$\beta_m \leftarrow \beta_m - \eta\beta_m \sum_{k=1}^{k} \phi_m^k(\mathbf{x}, \mathbf{c}) + \eta\beta_m \frac{\phi_m^y(\mathbf{x}, \mathbf{c})}{c_y} \tag{3.13}$$

This effectively prevents instantaneous bankrupticies.

Algorithm 2 Budget Update $(\mathbf{x}, y, \mathbf{c})$

```
Input: Training example $(\mathbf{x}, y)$, price $\mathbf{c}$
for $m = 1$ to $M$ do
  Update participant $m$'s budget as
    $\beta_m \leftarrow \beta_m - \eta \sum_{k=1}^{K} \beta_m \phi_m^k(\mathbf{x}, \mathbf{c}) + \eta \frac{\beta_m}{c_y} \phi_m^y(\mathbf{x}, \mathbf{c})$   (3.14)
end for
```

However, this budget update rule assumes that the equilibrium price $\mathbf{c}$ is already known. Since participants in this artificial market can only trade within the market, the total sum of budgets should remain constant after every update. In other words, the budget sum before and after the update should equate

$$\sum_{m=1}^{M} \beta_m = \sum_{m=1}^{M} \left[\beta_m - \beta_m \sum_{j=1}^{K} \phi_m^j(\mathbf{x}, \mathbf{c}) + \beta_m \frac{\phi_m^k(\mathbf{x}, \mathbf{c})}{c_k} \right] \quad \forall k = 1, 2, \ldots, K \tag{3.15}$$

Since the ground truth label k isn't necessarily known in advance, conservation should be satisfied for all possible labels k. This gives a system of equations whose solution defines the equilibrium price. This can also be rewritten as a fixed-point equation

$$\mathbf{c} = \frac{1}{n} \sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x}, \mathbf{c}) \tag{3.16}$$

$$n = \sum_{m=1}^{M} \beta_m \sum_{k=1}^{K} \phi_m^k(\mathbf{x}, \mathbf{c}) \tag{3.17}$$

where n can be interpreted as both a normalizer (ensuring that $\sum_{k=1}^{K} c_k = 1$) as well as the total bet among all participants. This can be shown to conserve the budget sum

Theorem 3.1.1. Price Equations. *The total budget* $\sum_{m=1}^{M} \beta_m$ *is conserved after the* **Budget Update**$(\mathbf{x}, y, \mathbf{c})$, *independent of the outcome* y, *if and only if* $c_k > 0, k = 1, ..., K$ *and*

$$\sum_{m=1}^{M} \beta_m \phi_m^k(\mathbf{x}, \mathbf{c}) = c_k B(\mathbf{x}, \mathbf{c}), \quad \forall k = 1, ..., K \tag{3.18}$$

where $n = B(\mathbf{x}, \mathbf{c}$ from (3.17). The proof is given in the Appendix.

This equilibrium is unique whenever $\phi^k(\mathbf{x}, c_k)$ is monotonically decreasing in c_k and satisfies Assumption 1. The proof is given in the appendix.

Remark 3.1.2. *Denoting* $f_k(c_k) = \frac{1}{c_k} \sum_{m=1}^{M} \beta_m \phi_m^k(\mathbf{x}, c_k)$, $k = 1, 2, \ldots, K$, *if all* $f_k(c_k)$ *are continuous and strictly decreasing in* c_k *as long as* $f_k(c_k) > 0$, *then for every* $n > 0$, $n \geq n_k = f_k(1)$ *there is a unique* $c_k = c_k(n)$ *that satisfies* $f_k(c_k) = n$.

Assumption 1. *The total bet of participant* $(\beta_m, \phi_m(\mathbf{x}, \mathbf{c}))$ *is positive inside the simplex* Δ, *i.e.*

$$\sum_{j=1}^{K} \phi_m^j(\mathbf{x}, c_j) > 0, \ \forall c \in (0,1)^K, \sum_{j=1}^{K} c_j = 1. \tag{3.19}$$

Theorem 3.1.3. *Assume all betting functions* $\phi_m^k(\mathbf{x}, c_k), m = 1, ..., M, k = 1, ..., K$ *are continuous, with* $\phi^k(\mathbf{x}, 0) > 0$ *and* $\phi_m^k(\mathbf{x}, c)/c$ *is strictly decreasing in* c *as long as* $\phi_m^k(\mathbf{x}, c) > 0$. *If the betting function* $\phi_m(\mathbf{x}, \mathbf{c})$ *of least one participant with* $\beta_m > 0$ *satisfies Assumption 1, then for the* **Budget Update**$(\mathbf{x}, y, \mathbf{c})$ *there is a unique price* $\mathbf{c} = (c_1, ..., c_K) \in (0,1)^K \cap \Delta$ *such that the total budget* $\sum_{m=1}^{M} \beta_m$ *is conserved.*

Algorithm 3 Prediction Market Training

```
Input: Training examples $(\mathbf{x_i}, y_i), i = 1, ..., N$
Initialize all budgets $\beta_m = \beta_0, m = 1, ..., M$.
for each training example $(\mathbf{x}_i, y_i)$ do
   Compute equilibrium price $\mathbf{c}_i$ using Eq. 3.18
   Run Budget Update $(\mathbf{x}_i, y_i, \mathbf{c}_i)$
end for
```

3.2 Solving the Market Price Equations

In practice, a double bisection algorithm could be used to find the equilibrium price, computing each $c_k(n)$ by the bisection method, and employing another bisection algorithm to find n such that the price condition $\sum_{k=1}^{K} c_k(n) = 1$ holds. Observe that the n satisfying $\sum_{k=1}^{K} c_k(n) = 1$ can be bounded from above by

$$n = n \sum_{k=1}^{K} c_k(n) = \sum_{k=1}^{K} c_k(n) f_k(c_k(n)) = \sum_{k=1}^{K} \sum_{m=1}^{M} \beta_m \phi_m^k(\mathbf{x}, \mathbf{c}) \leq \sum_{m=1}^{M} \beta_m$$

because for each m, $\sum_{k=1}^{K} \phi_m^k(\mathbf{x}, \mathbf{c}) \leq 1$.

A potentially faster alternative to the double bisection method is the Mann Iteration [37] described in Algorithm 4. The price equations can be viewed as fixed point equation $F(\mathbf{c}) = \mathbf{c}$, where $F(\mathbf{c}) = \frac{1}{n}(f_1(\mathbf{c}), ..., f_K(\mathbf{c}))$ with $f_k(\mathbf{c}) = \sum_{m=1}^{m} \beta_m \phi_m^k(\mathbf{x}, c_k)$. The Mann iteration is a fixed point algorithm, which makes weighted update steps

$$\mathbf{c}^{t+1} = (1 - \frac{1}{t})\mathbf{c}^t + \frac{1}{t}F(\mathbf{c}^t)$$

The Mann iteration is guaranteed to converge for contractions or pseudo-contractions. However, we observed experimentally that it usually converges in only a few (up to 10) steps, making it about 100-1000 times faster than the double bisection algorithm. If, after a small number of steps, the Mann iteration has not converged, the double bisection algorithm is used on that instance to compute the equilibrium price. However, this happens on less than 0.1% of the instances.

Algorithm 4 Market Price by Mann Iteration

```
Initialize $i = 1,\ c_k = \frac{1}{K}, k = 1, ..., K$
repeat
  $f_k = \sum_m \beta_m \phi_m^k(x, \mathbf{c})$
  $n = \sum_k f_k$
  if $n \neq 0$ then
    $f_k \leftarrow \frac{f_k}{n}$
    $r_k = f_k - c_k$
    $c_k \leftarrow \frac{(i-1)c_k + f_k}{i}$
  end if
  $i \leftarrow i + 1$
until $\sum_k |r_k| \leq \epsilon$ or $n = 0$ or $i > i_{\max}$
```

3.2.1 Two-class Formulation

For the two-class problem, i.e. $K = 2$, the budget equation can be simplified by writing $\mathbf{c} = (1 - c, c)$ and obtaining the *two-class market price equation*

$$(1-c)\sum_{m=1}^{M} \beta_m \phi_m^2(\mathbf{x}, c) - c\sum_{m=1}^{M} \beta_m \phi_m^1(\mathbf{x}, 1-c) = 0 \tag{3.20}$$

This can be solved numerically directly in c using the bisection method. Again, the solution is unique if $\phi_m^k(\mathbf{x}, c_k), m = 1, ..., M, k = 1, 2$ are continuous, monotonically non-increasing and $\sum_{k=1}^{K} \phi_m^k(\mathbf{x}, c_k) > 0,\ \forall c \in (0,1)^K,\ \sum_{k=1}^{K} c_k = 1$. Moreover, the solution is guaranteed to exist if there exist m, m' with $\beta_m > 0, \beta_{m'} > 0$ and such that $\phi_m^2(\mathbf{x}, 0) > 0, \phi_{m'}^1(\mathbf{x}, 1) > 0$.

3.3 Specialization

In real world prediction markets, participants bid on events that pertain to their expertise. In a classification market, this amounts to participants that bid only on a subset of the feature space. See figure 3.3 for a simple example. This idea was explored in [34, 33, 3] and showed promising results when aggregating the leaves of decision trees. For example, in figure 3.3, each partition of the domain corresponds to a market participant. Since decision tree leaves describe disjoint partitions, the market participants are taken from the leaves of several random trees.

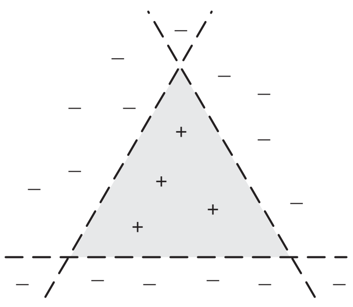


Figure 3.3: A perfect classifier can be constructed for the triangular region above from a market of six specialized classifiers that only bid on a half-plane determined by one side of the triangle. Three of these specialized classifiers have 100% accuracy while the other three have low accuracy. Nevertheless, the market is capable of obtaining 100% accuracy overall.

3.4 Loss Function

In the general market, the artificial prediction market can be shown to maximize log likelihood. The loss function is then the negative log likelihood

$$\ell(\boldsymbol{\beta}) = -\sum_{(\mathbf{x},y)\in(X,Y)} \log(c_y(\mathbf{x};\boldsymbol{\beta})) \tag{3.21}$$

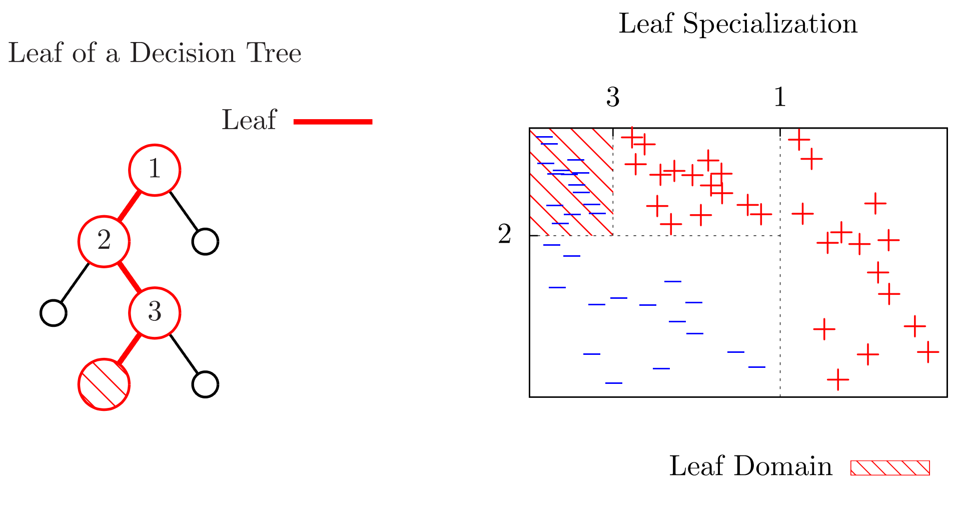
This can be shown with either stochastic gradient descent or as recasting the problem in terms of the KL divergence.

3.4.1 Stochastic Gradient

Consider the reparametrization $\gamma = (\gamma_1, ..., \gamma_M) = (\sqrt{\beta_1}, ..., \sqrt{\beta_M})$. The market price $\mathbf{c}(\mathbf{x}) = (c_1(\mathbf{x}), ..., c_K(\mathbf{x})$ is an estimate of the class probability $p(y = k|\mathbf{x})$ for each instance $\mathbf{x} \in \Omega$. Thus a set of training observations $(\mathbf{x}_i, y_i), i = 1, ..., N$, since $\hat{p}(y = y_i|\mathbf{x}_i) = c_{y_i}(\mathbf{x}_i)$, the (normalized) log-likelihood function is

$$L(\gamma) = \frac{1}{N}\sum_{i=1}^{N} \ln \hat{p}(y = y_i|\mathbf{x}_i) = \frac{1}{N}\sum_{i=1}^{N} \ln c_{y_i}(\mathbf{x}_i) \tag{3.22}$$

We will again use the total amount bet $B(\mathbf{x}, \mathbf{c}) = \sum_{m=1}^{M}\sum_{k=1}^{K} \beta_m \phi_m^k(\mathbf{x}, \mathbf{c})$ for observation $\mathbf{x}$ at market price $\mathbf{c}$.



(a) (b)

Figure 3.4: This figure is an example of a decision tree leaf (a) and its specialization domain (b). Decision tree leaves are *perfect* classifiers of the training data on their subdomain. However, they may not generalize on unseen data.

We will first focus on the constant market $\phi_m^k(\mathbf{x}, \mathbf{c}) = \phi_m^k(\mathbf{x})$, in which case $B(\mathbf{x}, \mathbf{c}) = B(\mathbf{x}) = \sum_{m=1}^M \sum_{k=1}^K \beta_m \phi_m^k(\mathbf{x})$. We introduce a batch update on all the training examples $(\mathbf{x}_i, y_i), i = 1, ..., N$:

$$\beta_m \leftarrow \beta_m + \beta_m \frac{\eta}{N} \sum_{i=1}^N \frac{1}{B(\mathbf{x}_i)} \left(\frac{\phi_m^{y_i}(\mathbf{x}_i)}{c_{y_i}(\mathbf{x}_i)} - \sum_{k=1}^K \phi_m^k(\mathbf{x}_i) \right). \tag{3.23}$$

Equation (3.23) can be viewed as presenting all observations $(\mathbf{x}_i, y_i)$ to the market simultaneously instead of sequentially. The following statement is proved in the Appendix

Theorem 3.4.1. ML for constant market. *The update* (3.23) *for the constant market maximizes the likelihood* (3.22) *by gradient ascent on* γ *subject to the constraint* $\sum_{m=1}^M \gamma_m^2 = 1$. *The incremental update*

$$\beta_m \leftarrow \beta_m + \beta_m \frac{\eta}{B(\mathbf{x}_i)} \left(\frac{\phi_m^{y_i}(\mathbf{x}_i)}{c_{y_i}(\mathbf{x}_i)} - \sum_{k=1}^K \phi_m^k(\mathbf{x}_i) \right). \tag{3.24}$$

maximizes the likelihood (3.22) *by constrained stochastic gradient ascent.*

In the general case of non-constant betting functions, the log-likelihood is

$$L(\gamma) = \sum_{i=1}^N \log c_{y_i}(\mathbf{x}_i) = \sum_{i=1}^N \log \sum_{m=1}^M \gamma_m^2 \phi_m^{y_i}(\mathbf{x}_i, \mathbf{c}(\mathbf{x}_i)) - \sum_{i=1}^N \log \sum_{k=1}^K \sum_{m=1}^M \gamma_m^2 \phi_m^k(\mathbf{x}_i, \mathbf{c}(\mathbf{x}_i)) \tag{3.25}$$

If we ignore the dependence of $\phi_m^k(\mathbf{x}_i, \mathbf{c}(\mathbf{x}_i))$ on γ in (3.25), and approximate the gradient as:

$$\frac{\partial L(\gamma)}{\partial \gamma_j} \approx \sum_{i=1}^{N} \left(\frac{\gamma_j \phi_j^{y_i}(\mathbf{x}_i, \mathbf{c}(\mathbf{x}_i))}{\sum_{m=1}^{M} \gamma_m^2 \phi_m^{y_i}(\mathbf{x}_i, \mathbf{c}(\mathbf{x}_i))} - \frac{\gamma_j \sum_{k=1}^{K} \phi_j^k(\mathbf{x}_i, \mathbf{c}(\mathbf{x}_i))}{\sum_{k=1}^{K} \sum_{m=1}^{M} \gamma_m^2 \phi_m^k(\mathbf{x}_i, \mathbf{c}(\mathbf{x}_i))} \right)$$

then the proof of Theorem 3.4.1 follows through and we obtain the following market update

$$\beta_m \leftarrow \beta_m + \beta_m \frac{\eta}{B(\mathbf{x}, \mathbf{c})} \left[\frac{\phi_m^y(\mathbf{x}, \mathbf{c})}{c_y} - \sum_{k=1}^{K} \phi_m^k(\mathbf{x}, \mathbf{c}) \right], \ m = 1, ..., M \tag{3.26}$$

This way we obtain only an approximate statement in the general case

Remark 3.4.2. Maximum Likelihood. *The prediction market update* (3.26) *finds an approximate maximum of the likelihood* (3.22) *subject to the constraint* $\sum_{m=1}^{M} \gamma_m^2 = 1$ *by an approximate constrained stochastic gradient ascent.*

Observe that the updates from (3.24) and (3.26) differ from the update (3.12) by using an adaptive step size $\eta/B(\mathbf{x}, \mathbf{c})$ instead of the fixed step size 1.

It is easy to check that maximizing the likelihood is equivalent to minimizing an approximation of the expected KL divergence to the true distribution

$$E_\Omega[KL\left(p(y|\mathbf{x}), c_y(\mathbf{x})\right)] = \int_\Omega p(\mathbf{x}) \int_Y p(y|\mathbf{x}) \log \frac{p(y|\mathbf{x})}{c_y(\mathbf{x})} dy d\mathbf{x}$$

obtained using the training set as Monte Carlo samples from $p(\mathbf{x}, y)$.

In many cases the number of negative examples is much larger than the positive examples, and is desired to maximize a weighted log-likelihood

$$L(\gamma) = \frac{1}{N} \sum_{i=1}^{N} w(\mathbf{x}_i) \ln c_{y_i}(\mathbf{x}_i)$$

This can be achieved (exactly for constant betting and approximately in general) using the weighted update rule

$$\beta_m \leftarrow \beta_m + \eta w(\mathbf{x}) \frac{\beta_m}{B(\mathbf{x}, \mathbf{c})} \left[\frac{\phi_m^y(\mathbf{x}, \mathbf{c})}{c_y} - \sum_{k=1}^{K} \phi_m^k(\mathbf{x}, \mathbf{c}) \right], \ m = 1, ..., M \tag{3.27}$$

The parameter η and the number of training epochs can be used to control how close the budgets β are to the ML optimum, and this way avoid overfitting the training data.

An important issue for the real prediction markets is the *efficient market hypothesis*, which states that the market price fuses in an optimal way the information available to the market participants [23, 5, 35]. From Theorem 3.4.1 we can draw the following conclusions for the artificial prediction market with constant betting:

1. In general, an untrained market (in which the budgets have not been updated based on training data) will not satisfy the efficient market hypothesis.

2. The market trained with a large amount of representative training data and small η satisfies the efficient market hypothesis.

3.4.2 Contraction Mapping

The Constant Market minimizes the expected KL divergence given by

$$E_\Omega[KL\left(p(y|\mathbf{x}), c_y(\mathbf{x})\right)] = \int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \log \frac{p(k|\mathbf{x})}{c_k(\mathbf{x})} d\mathbf{x} \tag{3.28}$$

This can be shown directly by posing this problem as a fixed point problem. First denote the equilibrium price as

$$c_y(\mathbf{x}; \boldsymbol{\beta}) = \sum_{m=1}^{M} \beta_m h_m^y(\mathbf{x})$$

then the loss function is defined in terms of $\boldsymbol{\beta}$ as

$$\ell(\boldsymbol{\beta}) = \int_\Omega p(\mathbf{x}) KL(p(y|\mathbf{x}), c_y(\mathbf{x}; \boldsymbol{\beta})) d\mathbf{x} \tag{3.29}$$

The loss function is strictly convex whenever $\boldsymbol{\beta} \in \mathbb{R}^M_{\geq 0} \backslash \{\mathbf{0}\}$

Theorem 3.4.3. Strict Convexity *$\ell(\boldsymbol{\beta})$ is strictly convex whenever $\boldsymbol{\beta} \in \mathbb{R}^M_{\geq 0} \backslash \{\mathbf{0}\}$.*

Now reparameterize $\boldsymbol{\beta}$ in terms of $\mathbf{u}, \mathbf{v} \in \mathbb{R}^M$, $\sum_{m=1}^{M} u_m = 1$ and $Z \in \mathbb{R}^{M \times M}$

$$\boldsymbol{\beta} = \mathbf{u} + Z\mathbf{v}$$

where Z is defined as

$$Z = \frac{1}{M}(I - \mathbf{1}\mathbf{1}^T)$$

Observe that Z is symmetric and subtracts the mean off any vector it projects, such that $Z\mathbf{x} = \mathbf{x} - \bar{\mathbf{x}}$. More importantly, any $\mathbf{y} = Z\mathbf{x}$ has a component sum of 0 so that $\boldsymbol{\beta}$ is always guaranteed to sum to 1. Taking the gradient with respect to $\mathbf{v}$ then gives

$$\begin{aligned}
\nabla_{\mathbf{v}} E[KL(p(y|\mathbf{x}), c_y(\mathbf{x}; \boldsymbol{\beta}))] &= -\int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{1}{c_k(\mathbf{x}; \boldsymbol{\beta})} \nabla_{\mathbf{v}} c_k(\mathbf{x}; \boldsymbol{\beta}) d\mathbf{x} \\
&= -\int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{1}{c_k(\mathbf{x}; \boldsymbol{\beta})} Z^T \nabla_{\boldsymbol{\beta}} c_k(\mathbf{x}; \boldsymbol{\beta}) d\mathbf{x} \\
&= -Z^T \int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{\mathbf{h}^k(\mathbf{x})}{c_k(\mathbf{x}; \boldsymbol{\beta})} d\mathbf{x}
\end{aligned}$$

where $\mathbf{h}^k(\mathbf{x}) = [h_1^k(\mathbf{x}), h_2^k(\mathbf{x}), \ldots h_M^k(\mathbf{x})]$ denotes the vector of classifiers. We want to solve

$$\nabla_{\mathbf{v}} E[KL(p(y|\mathbf{x}), c_y(\mathbf{x}; \boldsymbol{\beta}))] = -Z^T \int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{\mathbf{h}^k(\mathbf{x})}{c_k(\mathbf{x}; \boldsymbol{\beta})} d\mathbf{x} = \mathbf{0}$$

Since rows of Z sum to zero, it follows that the only solutions to the above system is a scaled $\mathbf{1}$ vector

$$\int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{\mathbf{h}^k(\mathbf{x})}{c_k(\mathbf{x}; \boldsymbol{\beta})} d\mathbf{x} = C\mathbf{1} \quad C \in \mathbb{R}$$

The constant C is determined by hitting both sides with $\boldsymbol{\beta}^T$

$$\int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{\boldsymbol{\beta}^T \mathbf{h}^k(\mathbf{x})}{c_k(\mathbf{x};\boldsymbol{\beta})} d\mathbf{x} = C\boldsymbol{\beta}^T \mathbf{1} \implies C = 1$$

where $c_k(\mathbf{x};\boldsymbol{\beta}) = \boldsymbol{\beta}^T \mathbf{h}^k(\mathbf{x}) = \sum_{m=1}^{M} \beta_m h_m^k(\mathbf{x})$. So to the solve the system

$$\int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{\mathbf{h}^k(\mathbf{x})}{c_k(\mathbf{x};\boldsymbol{\beta})} d\mathbf{x} = \mathbf{1}$$

consider solving the fixed point problem

$$g_m(\beta_m) = \beta_m f_m(\boldsymbol{\beta}) = \beta_m \forall m = 1, 2, \ldots, M \tag{3.30}$$

where

$$\mathbf{f}(\boldsymbol{\beta}) = \int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{\mathbf{h}^k(\mathbf{x})}{c_k(\mathbf{x};\boldsymbol{\beta})} d\mathbf{x}$$

with the iterative method

$$\boldsymbol{\beta}^{t+1} = \mathbf{g}(\boldsymbol{\beta}^t) \tag{3.31}$$

It is easy to check that (3.31) is equivalent to (3.23) by using the training set as Monte Carlo samples in Monte Carlo quadrture to estimate the integral. It is therefore a map

$$\mathbf{g} \; : \; B^M \to B^M$$

where $B^M = \{\boldsymbol{\beta} \in [0,1]^M \; : \; \|\boldsymbol{\beta}\|_1 = 1\}$ is the set of all admissible budget configurations. The fixed point method (3.31) will converge for some $\boldsymbol{\beta}^0 \in B^M$ if the Jacobian matrix $J_{\mathbf{g}}(\boldsymbol{\beta}^*)$ has eigenvalues with magnitude strictly less than 1.

Theorem 3.4.4. Eigenvalues of Jacobian *The eigenvalues of $J_{\mathbf{g}}(\boldsymbol{\beta}^*)$, where $\boldsymbol{\beta}^* \in B^M$ is the minimizer of* (3.29)*, have magnitude strictly less than* 1.

3.4.3 Weighted Updates

In many real world problems, the number of training instances with a particular label y greatly outnumbers training instances with other training labels, in other words $N_k \ll N_y,\; k \neq y$. In such cases, it is trivial to minimize the misclassification rate by always classifying the dominant label. Thus, the market update rule (3.13) will tend to favor those participants that bet on the dominant label. To prevent this, consider varying η based on the class label. That is, those training instances with less frequent label ought to be weighted more than those training instances with more frequent labels, or in this case $\eta_y < \eta_k,\; k \neq y$. This gives the modified update rule

$$\beta_m \leftarrow \beta_m - \eta_y \beta_m \sum_{k=1}^{k} \phi_m^k(\mathbf{x}, \mathbf{c}) + \eta_y \beta_m \frac{\phi_m^y(\mathbf{x}, \mathbf{c})}{c_y} \tag{3.32}$$

In the case of the constant market, $\phi_m^k(\mathbf{x}, \mathbf{c}) = h_m^k(\mathbf{x})$, the KL loss (3.28) can be rewritten to reveal a regularization mechanism that relates directly back to η_y in (3.32). By using Bayes rule we have

$$E_\Omega[KL\left(p(y|\mathbf{x}), c_y(\mathbf{x})\right)] = \int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \log \frac{p(k|\mathbf{x})}{c_k(\mathbf{x})} d\mathbf{x} \tag{3.33}$$

$$= \sum_{k=1}^{K} p(k) \int_\Omega p(\mathbf{x}|k) \log \frac{p(k|\mathbf{x})}{c_k(\mathbf{x})} d\mathbf{x} \tag{3.34}$$

Parameterizing $\boldsymbol{\beta}$, as in 3.4.2

$$\boldsymbol{\beta} = \mathbf{u} + Z\mathbf{v}$$

and differentiating with respect to components of $\mathbf{v}$ then leads to a similar problem

$$\sum_{k=1}^{K} p(k) \int_\Omega p(\mathbf{x}|k) \frac{\mathbf{h}^k(\mathbf{x})}{c_k(\mathbf{x}; \boldsymbol{\beta})} d\mathbf{x} = \mathbf{1}$$

and through the same means, can be rewritten as a fixed point problem

$$\sum_{k=1}^{K} p(k) \int_\Omega p(\mathbf{x}|k) \frac{\beta_m h_m^k(\mathbf{x})}{c_k(\mathbf{x}; \boldsymbol{\beta})} d\mathbf{x} = \beta_m \quad m = 1, 2, \ldots, M$$

The same approach used to show (3.31) is a contraction can also be used to show that this mapping is a contraction and therefore this gives the update rule

$$\beta_m^{t+1} = \sum_{k=1}^{K} p(k) \int_\Omega p(\mathbf{x}|k) \frac{\beta_m^t h_m^k(\mathbf{x})}{c_k(\mathbf{x}; \boldsymbol{\beta}^t)} d\mathbf{x}$$

Since $p(\mathbf{x}|k)$ is not normally known, the integral can be estimated by taking the training instances as Monte Carlo quadrature points, giving

$$\beta_m^{t+1} = \sum_{k=1}^{K} p(k) \frac{1}{|X_k|} \sum_{\mathbf{x} \in X_k} \frac{\beta_m^t h_m^k(\mathbf{x})}{c_k(\mathbf{x}; \boldsymbol{\beta}^t)} \tag{3.35}$$

where X_k is the set of training instances with label k. Loosely using the relationship between the batch update rule (3.23) and the incremental update rule (3.24), this gives the following weighted update rule

$$\beta_m \leftarrow \beta_m + \frac{p(y)}{|X_y|} \beta_m \left(\frac{h_m^y(\mathbf{x})}{c_y(\mathbf{x}; \boldsymbol{\beta})} - 1 \right) \tag{3.36}$$

which suggests that

$$\eta_y = \frac{p(y)}{|X_y|} \tag{3.37}$$

This weighted update rule (3.36) is empirically demonstrated in section 6.1.4 where the number of negative samples greatly outnumbers the number of positive samples. Without this modification, the market would favor the participants that bet on negatives and never classify positive for any instance.

3.4.4 Case Study

We first investigate the behavior of three markets on a dataset in terms of training and test error as well as loss function. For that, we chose the `satimage` dataset from the UCI repository [7] since it has a supplied test set. The `satimage` dataset has a training set of size 4435 and a test set of size 2000.

The markets investigated are the constant market with both incremental and batch updates, given in eq. (3.24) and (3.23) respectively, the linear and aggressive markets with incremental updates given in (3.26). Observe that the η in eq. (3.24) is not divided by N (the number of observations) while the η in (3.23) is divided by N. Thus to obtain the same behavior the η in (3.24) should be the η from (3.23) divided by N. We used $\eta = 100/N$ for the incremental update and $\eta = 100$ for the batch update unless otherwise specified.

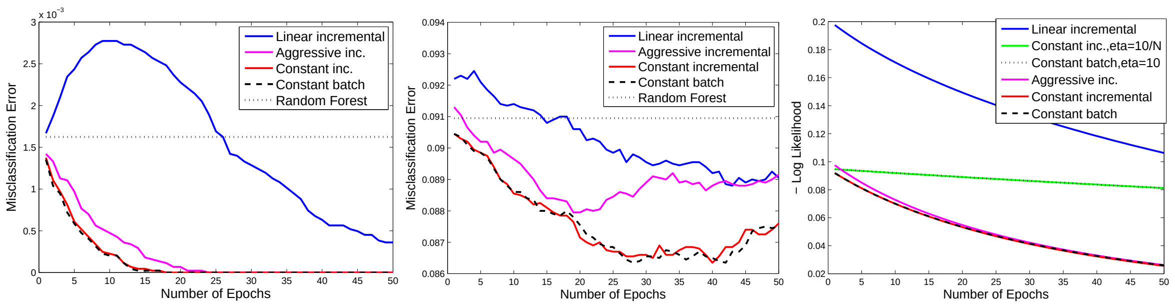


Figure 3.5: Experiments on the satimage dataset for the incremental and batch market updates. Left: The training error vs. number of epochs. Middle: The test error vs. number of epochs. Right: The negative log-likelihood function vs. number of training epochs. The learning rates are $\eta = 100/N$ for the incremental update and $\eta = 100$ for the batch update unless otherwise specified.

In Figure 3.5 are plotted the misclassification errors on the training and test sets and the negative log-likelihood function vs. the number of training epochs, averaged over 10 runs. From Figure 3.5 one could see that the incremental and batch updates perform similarly in terms of the likelihood function, training and test errors. However, the incremental update is preferred since it is requires less memory and can handle an arbitrarily large amount of training data. The aggressive and constant markets achieve similar values of the negative log likelihood and similar training errors, but the aggressive market seems to overfit more since the test error is larger than the constant incremental (p-value< 0.05). The linear market has worse values of the log-likelihood, training and test errors (p-value< 0.05).

3.5 Relation with Existing Supervised Learning Methods

Aside of general linear aggregation techniques, artificial prediction markets for classification can also mimic logistic regression and support vector machines. While not necessarily identical, it demonstrates the potential of prediction markets as a learning framework for classification tasks.

3.5.1 Constant Market

One specific and successful example of a classification market is the so-called Constant Market. In the constant market, betting functions are taken to be the classifiers themselves

$$\boldsymbol{\phi}(\mathbf{x}, \mathbf{c}) = \mathbf{h}(\mathbf{x}) \tag{3.38}$$

Participants bet entirely on experience and the price is completely ignored. This violates some of the properties previously described.

The equilibrium price in this market is given as a linear combination of classifiers

$$\mathbf{c}(\mathbf{x}) = \sum_{m=1}^{M} \beta_m \mathbf{h}_m(\mathbf{x}) \tag{3.39}$$

where $\sum_{m=1}^{M} \beta_m = 1$. This type of market is an example of linear aggregation which include methods such as Boosting [26] and Random Forest [9].

3.5.2 Logistic Regression

By taking the following betting functions

$$\phi_m^1(\mathbf{x}, 1-c) = (1-c)\left(\mathbf{x}_m^+ - \frac{1}{B}\ln(1-c)\right) \tag{3.40}$$

$$\phi_m^2(\mathbf{x}, c) = c\left(-x_m^- - \frac{1}{B}\ln c\right) \tag{3.41}$$

with $x^+ = xI(x > 0), x^- = xI(x < 0)$ where $I(\cdot)$ is the indicator function, and $B = \sum_{m=1}^{M} \beta_m$. The binary class equilibrium equations then become

$$\sum_{m=1}^{M} \beta_m c(1-c)\left(\mathbf{x}_m - \frac{1}{B}\ln(1-c) + \frac{1}{B}\ln(c)\right) = 0$$

and so $\ln \frac{1-c}{c} = \sum_{m=1}^{M} \beta_m \mathbf{x}_m$, which gives the logistic regression model

$$c = \frac{1}{1 + \exp(\sum_{m=1}^{M} \beta_m \mathbf{x}_m)}$$

This gives the update rule $\beta_m \leftarrow \beta_m - \eta\beta_m[(1-c)\mathbf{x}_m^+ + c\mathbf{x}_m^- - H(c)/B] + \eta\beta_m u_y(c)$, where $u_1(c) = \mathbf{x}_m^+ - \ln(1-c)/B,\ u_2(c) = -\mathbf{x}_m^- - \ln(c)/B$.

Writing $\mathbf{x}\boldsymbol{\beta} = \sum_{m=1}^{M} \beta_m \mathbf{x}_m$, the budget update can be rearranged to

$$\beta_m \leftarrow \beta_m - \eta\beta_m\left(\mathbf{x}_m - \frac{\mathbf{x}\boldsymbol{\beta}}{B}\right)\left(y - \frac{1}{1+\exp(\mathbf{x}\boldsymbol{\beta})}\right) \tag{3.42}$$

This equation resembles the standard per-observation update equation for online logistic regression:

$$\beta_m \leftarrow \beta_m - \eta\mathbf{x}_m\left(y - \frac{1}{1+\exp(\mathbf{x}\boldsymbol{\beta})}\right) \tag{3.43}$$

with two differences. The term $\mathbf{x}\boldsymbol{\beta}/B$ ensures the budgets always sum to B while the β_m ensures that $\beta_m \geq 0$.

The update from eq. (3.42), like eq. (3.43), tries to increase $|\mathbf{x}\boldsymbol{\beta}|$, but it does that subject to constraints that $\beta_m \geq 0,\ m = 1, \ldots, M$ and $\sum_{m=1}^{M} \beta_m = B$. Also observe that multiplying $\boldsymbol{\beta}$ by a constant does not change the decision line of the logistic regression.

3.5.3 Support Vector Machine

If each training instance $(\mathbf{x}_m, y_m),\ m = 1, 2, \ldots, M$ is taken to be associated with a participant in the market, with betting function $\phi_m(\mathbf{x})$ defined in terms of

$$u_m(\mathbf{x}) = \frac{\mathbf{x}_m^T \mathbf{x}}{\|\mathbf{x}_m\| \|\mathbf{x}\|}$$

and

$$\begin{aligned} \phi_m^{y_m}(\mathbf{x}) = u_m(\mathbf{x})^+ &= \begin{cases} u_m(\mathbf{x}) & u_m(\mathbf{x}) \geq 0 \\ 0 & \text{otherwise} \end{cases} \\ \phi_m^{2-y_m}(\mathbf{x}) = -u_m(\mathbf{x})^- &= \begin{cases} 0 & u_m(\mathbf{x}) \geq 0 \\ -u_m(\mathbf{x}) & \text{otherwise} \end{cases} \end{aligned} \tag{3.44}$$

This gives a constant market with two-class price equations

$$c = \frac{\sum_{m=1}^{M} \beta_m \phi_m^2(\mathbf{x})}{\sum_{m=1}^{M} \beta_m \left(\phi_m^1(\mathbf{x}) + \phi_m^2(\mathbf{x})\right)} = \frac{\sum_{m=1}^{M} \beta_m \left[y_m u_m(\mathbf{x}) - u_m(\mathbf{x})^-\right]}{\sum_{m=1}^{M} \beta_m |u_m(\mathbf{x})|}$$

since $\phi_m^2(\mathbf{x}) = y_m u_m(\mathbf{x}) - u_m(\mathbf{x})^-$ and $\phi_m^1 + \phi_m^2 = |u_m(\mathbf{x})|$. The decision rule $c > 0.5$ then becomes $\sum_{m=1}^{M} \beta_m \phi_m^2(\mathbf{x}) > \sum_{m=1}^{M} \beta_m \phi_m^1$ or $\sum_{m=1}^{M} \beta_m \left(\phi_m^2(\mathbf{x}) - \phi_m^1(\mathbf{x})\right) > 0$. Since $\phi_m^2(\mathbf{x}) - \phi_m^1(\mathbf{x}) = (2y_m - 2)u_m(\mathbf{x}) = (2y_m - 2)\frac{\mathbf{x}_m^T \mathbf{x}_m}{\|\mathbf{x}_m\|\|\mathbf{x}\|}$ (where $y_m \in \{1, 2\}$), we obtain something like the SVM

$$h(\mathbf{x}) = \operatorname{sgn}\left(\sum_{m=1}^{M} \alpha_m (2y_m - 3)\mathbf{x}_m^T \mathbf{x}\right)$$

where $\alpha_m = \beta_m / \|\mathbf{x}_m\|$. In this case, the budget update becomes

$$\beta_m \leftarrow \beta_m - \eta\beta_m |u_m(\mathbf{x})| + \eta\beta_m \frac{\phi_m^y(\mathbf{x})}{c_y}$$

The same reasoning holds for $u_m(\mathbf{x}) = K(\mathbf{x}_m, \mathbf{x})$ with the RBF kernel $K(\mathbf{x}_m, \mathbf{x}) = \exp(-\|\mathbf{x}_m - \mathbf{x}\|^2/\sigma^2)$. In figure 3.6 left, is shown an example of the decision boundary of a market trained online with an RBF kernel with $\sigma = 0.2$ on 1000 examples uniformly sampled in the $[-1, 1] \times [-1, 1]$ domain. In Figure 3.6 right, is shown the estimated probability $\hat{p}(y = 1|\mathbf{x})$.

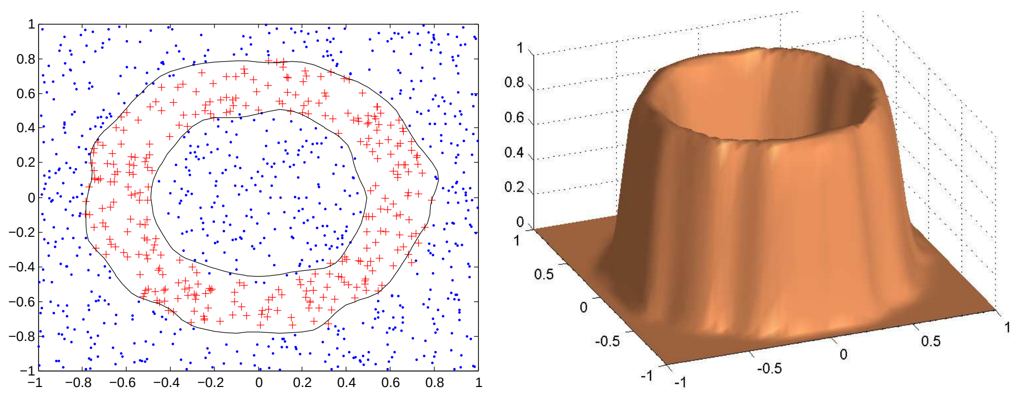


Figure 3.6: Left: 1000 training examples and learned decision boundary for an RBF kernel-based market from eq. (3.44) with $\sigma = 0.1$. Right: The estimated conditional probability function.

CHAPTER 4

PREDICTION MARKETS FOR REGRESSION

Described in the previous chapter, the classification market is defined by a *betting function* $\phi^k(\mathbf{x}, \mathbf{c})$ that describes the proportion of the budget β to allot for label k for a given instance $\mathbf{x}$ and trading prices for all labels $\mathbf{c}$. The equilibrium price $\mathbf{c}$ is defined such that the for any label, the sum of profits equaled the sum of losses

$$\sum_{m=1}^{M} \beta_m \frac{\phi_m^y(\mathbf{x}, \mathbf{c})}{c_y} = \sum_{m=1}^{M} \beta_m \sum_{k=1}^{K} \phi_m^k(\mathbf{x}, \mathbf{c}) \quad y = 1, 2, \ldots, K$$

This equilibrium system corresponds to the update rule for the classification market

$$\beta_m \leftarrow \beta_m - \beta_m \sum_{k=1}^{K} \phi_m^k(\mathbf{x}, \mathbf{c}) + \beta_m \frac{\phi_m^y(\mathbf{x}, \mathbf{c})}{c_y}$$

for $m = 1, 2, \ldots, M$. This is the profit. With a little reworking, the above equilibrium is equivalent to solving the following fixed point problem

$$c_k = \sum_{m=1}^{M} \beta_m \phi_m^k(\mathbf{x}, \mathbf{c}) \quad k = 1, 2, \ldots, K$$

The trading price $\mathbf{c}$ is considered to be an estimate of the conditional mass. In fact, [3] demonstrates that the classification market maximizes log likelihood.

4.1 Problem Setup

The extension of prediction markets to the regression problem proves to be counterintuitive. In classification, the goal is to predict the one correct label for a given instance. What can be said about regression? Assume, for the time being that the classification market framework generalizes. For the sake of consistency with probability notation $\phi(y|\mathbf{x}, c)$ will denote a betting functional that allots a proportion of the budget for response $y \in \mathbb{R}$. This implies that

$$\phi(y|\mathbf{x}, c) \geq 0 \tag{4.1}$$

$$\int_Y \phi(y|\mathbf{x}, c) dy \leq 1 \tag{4.2}$$

since no participant may bet more than the whole of their budget in this market. A curious consequence of this constraint is that it is possible for $\phi(y|\mathbf{x}, c) > 1$ for some y. Likewise, the trading prices for y are denoted as the price function $c(y|\mathbf{x})$. The trading price is a conditional density on the possible responses y. The prediction for y can be computed from, for example, expectation

$$y = \int_Y tc(t|\mathbf{x})dt \tag{4.3}$$

However, the price function can also model ambiguous responses. For example, points along a circle could result in a bimodal price function. An example of multi-modal price functions can be seen in figure 4.1 where a regression tree learns a conditional density on an Archimedes spiral.

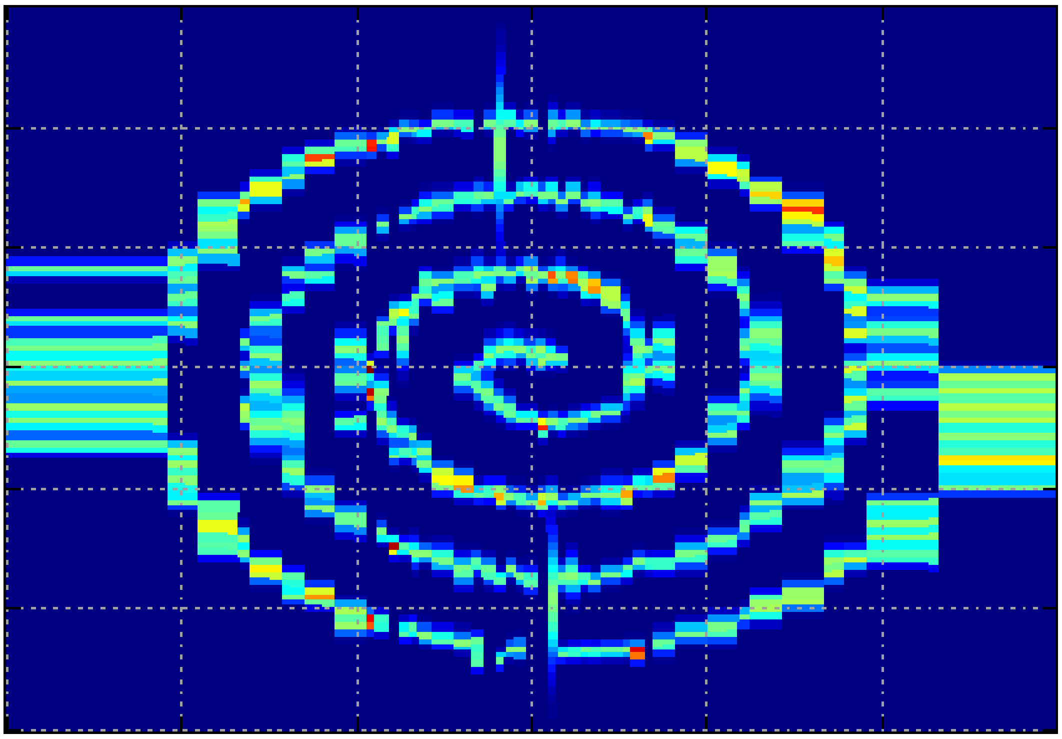


Figure 4.1: A conditional density of a *clustering* regression tree predicting multiple y values on an Archimedes spiral. The regression tree fits Gaussians to y values using EM. The splitting criteria is based on the average ℓ_2 residuals from the nearest cluster center. This illustrates how a Regression Market price function can be used to make predictions for more than just one y value. The distortion on the left and right sides correspond to the *default* leaf nodes used to make predictions beyond the training domain.

The equilibrium price function $c(y|\mathbf{x})$ receives similar treatment as the classification market. The objective is to find a $c(y|\mathbf{x})$ that gives conservation of budget. In other words, that the total winnings match the total losses. In the classification market, the winnings are determined on the one true class label $y = k$. However, in regression, predictions need only be accurate within some tolerance. Rewarding market participants based on their bet on the exact value of y may be too strict. This issue is resolved by introducing a *reward kernel* $K(t; y)$. The reward kernel is a density with a single mode centered about the ground truth y. The winnings are subsequently defined as

$$\text{winnings} = \int_Y K(t; y) \frac{\phi(t|\mathbf{x}, c)}{c(t|\mathbf{x})} dt \tag{4.4}$$

This has the effect of partially rewarding participants for nearby predictions. Likewise, the total expenditures for contracts are given as

$$\text{bet} = \int_Y \phi(t|\mathbf{x}, c) dt \tag{4.5}$$

These forms are similar to (3.11) (3.10) except with integrals instead of sums. This gives the general budget update in terms of profit

$$\beta_m \leftarrow \beta_m - \beta_m \int_Y \phi_m(t|\mathbf{x}, c) dt + \beta_m \int_Y K_y(t) \frac{\phi_m(t|\mathbf{x}, c)}{c(t|\mathbf{x})} dt$$

and can also be written like (3.12)

$$\beta_m \leftarrow \beta_m - \eta\beta_m \int_Y \phi_m(t|\mathbf{x}, c) dt + \eta\beta_m \int_Y K_y(t) \frac{\phi_m(t|\mathbf{x}, c)}{c(t|\mathbf{x})} dt \tag{4.6}$$

However, these rules assume $c(y|\mathbf{x})$ is already known. Similar to the classification market, the equilibrium price function $c(y|\mathbf{x})$ is defined such that gains match the losses

$$\sum_{m=1}^{M} \beta_m \int_Y K(t; y) \frac{\phi_m(t|\mathbf{x}, c)}{c(t|\mathbf{x})} dt = \sum_{m=1}^{M} \beta_m \int_Y \phi_m(t|\mathbf{x}, c) dt \tag{4.7}$$

At the time of this writing, there is no known general solution to (4.7). If $\phi_m(t|\mathbf{x}, c) = h_m(t|\mathbf{x})$ where $h_m(t|\mathbf{x})$ is a conditional density, then the solution is trivial and reduces to the Constant Regression Market. The general problem may be approximately solvable by considering an expansion of the price function of the form

$$c(t) = \sum_{i=1}^{N} \omega_i k_i(t)$$

where ω_i are weights and $k_i(t)$ are basis functions. Then the price function could be solved by solving for the weights ω_i. Another possibility is to consider solving for the price function value at discrete points t_i. In the update rule (4.13), these discrete points t_i would be the Hermite-Gauss nodal points.

4.1.1 Constant Market for Regression

For simplicity and the reported empirical performance of the *constant classification market*, the remainder of this chapter assumes $\phi(y|\mathbf{x}, c) = h(y|\mathbf{x})$ where $h(y|\mathbf{x})$ is a conditional density with mean $f(\mathbf{x})$. Here $f(\mathbf{x})$ is a regressor. This defines the *constant market* for regression with

$$c(y|\mathbf{x}) = \sum_{m=1}^{M} \beta_m h_m(y|\mathbf{x}) \tag{4.8}$$

$$y = \int_Y t c(t|\mathbf{x}) dt = \sum_{m=1}^{M} \beta_m f_m(\mathbf{x}) \tag{4.9}$$

The update rule is similar to that of the classification market in exception to the additional *reward kernel*

$$\beta_m \leftarrow \beta_m + \eta \beta_m \left(\int_Y K(t; y) \frac{h_m(t|\mathbf{x})}{c(t|\mathbf{x})} dt - 1 \right) \tag{4.10}$$

where η is the learning rate and also serves to prevent instanaeous bankruptcy (i.e. $\beta = 0$). The price function $c(y|\mathbf{x})$ satisfies (4.7) since

$$\begin{aligned} &\sum_{m=1}^{M} \left[\beta_m \int_Y K(t; \mathbf{x}) \frac{h_m(t|\mathbf{x})}{c(t|\mathbf{x})} dt - \beta_m \int_Y h_m(t|\mathbf{x}) dt \right] = \\ &\int_Y K(t; \mathbf{x}) \frac{\sum_{m=1}^{M} \beta_m h_m(t|\mathbf{x})}{c(t|\mathbf{x})} dt - 1 = \\ &\int_Y K(t; \mathbf{x}) dt - 1 = 0 \end{aligned}$$

The choice of $K(t; y)$ gives different update rules. We examine $K(t; y) = \delta(t - y)$ where $\delta(t)$ is the Dirac delta function and $K(t; y) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(t-y)^2}{2\sigma^2}}$

4.1.2 Delta Updates

When $K(t; y) = \delta(t - y)$ this gives an analogous update rule as the *classification market*

$$\beta_m \leftarrow \beta_m + \eta \beta_m \left(\frac{h_m(y|\mathbf{x})}{c(y|\mathbf{x})} - 1 \right) \tag{4.11}$$

Even though this reward kernel is exacting, it will be shown empirically to work relatively well.

4.1.3 Gaussian Updates

When $K(t; y) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(t-y)^2}{2\sigma^2}}$, this gives an update involving an integral

$$\beta_m \leftarrow \beta_m + \eta \beta_m \left(\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{-(t-y)^2}{2\sigma^2}} \frac{h_m(t|\mathbf{x})}{c(t|\mathbf{x})} dt - 1 \right) \tag{4.13}$$

Algorithm 5 Delta Budget Update $(\mathbf{x}, y, c)$

```
  **Input:** Training example $(\mathbf{x}, y)$, price function $c(y|\mathbf{x})$
  **for** $m = 1$ **to** $M$ **do**
    Update participant $m$'s budget as
        $$\beta_m \leftarrow (1-\eta)\beta_m + \eta \frac{\beta_m}{c(y|\mathbf{x})} h_m(y|\mathbf{x}) \quad (4.12)$$
  **end for**
```

One way to approximate this integral is with Hermite-Gauss quadrature [44]. A change of variables is required to apply the quadrature rule

$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{-(t-y)^2}{2\sigma^2}} \frac{h_m(t|\mathbf{x})}{c(t|\mathbf{x})} dt \tag{4.14}$$

$$= \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-t^2} \frac{h_m(y+\sqrt{2}\sigma t|\mathbf{x})}{c(y+\sqrt{2}\sigma t|\mathbf{x})} dt \tag{4.15}$$

$$\approx \frac{1}{\sqrt{\pi}} \sum_{i=1}^{n} \omega_i \frac{h_m(y+\sqrt{2}\sigma t_i|\mathbf{x})}{c(y+\sqrt{2}\sigma t_i|\mathbf{x})} \tag{4.16}$$

where ω_i, t_i are the n-point Hermite-Gauss weights and nodal points.

Intuitively, the choice of σ should reflect the noise variance of the training data (assuming Gaussian noise). If σ is too small, the market is more prone to overfitting. This σ can be chosen with cross validation by discretizing $\alpha \in (0, 1]$ and trying $\sigma = \alpha\sqrt{\frac{1}{N}\sum_{n=1}^{N} y_n^2}$ (assuming the noise has mean 0).

Algorithm 6 Gaussian Budget Update $(\mathbf{x}, y, c)$

```
  **Input:** Training example $(\mathbf{x}, y)$, price function $c(y|\mathbf{x})$
  **for** $m = 1$ **to** $M$ **do**
    Update participant $m$'s budget as
        $$\beta_m \leftarrow (1-\eta)\beta_m + \eta\beta_m \frac{1}{\sqrt{\pi}} \sum_{i=1}^{n} \omega_i \frac{h_m(y+\sqrt{2}\sigma t_i|\mathbf{x})}{c(y+\sqrt{2}\sigma t_i|\mathbf{x})} \quad (4.17)$$
  **end for**
```

4.1.4 Specialized Regression Markets

Introduced in [33], specialized markets are markets with participants which have local support in the feature space. This type of participant is assumed to perform relatively well in its domain. An example of a specialized market is a market with random tree leaves as participants. These types of markets have been demonstrated to be competitive with random forest. The specialized regression market of tree leaves is similar except that leaves are Gaussian instead of histograms. Each regression tree stores the sample mean $\bar{y}$ and variance σ^2 of instances that fall in each leaf.

4.2 Loss Function

Like the classification market, the regression market maximizes the log likelihood. The loss function is then the negative log likelihood, given by

$$\ell(\boldsymbol{\beta}) = -\sum_{(\mathbf{x},y)\in(X,Y)} \log(p(y|\mathbf{x})) \tag{4.18}$$

Similar approaches as in the previous chapter can be used to show this. Particularly, the contraction mapping approach in section 3.4.2 readily generalizes to the regression market. However, this minimizes the expected KL divergence given by

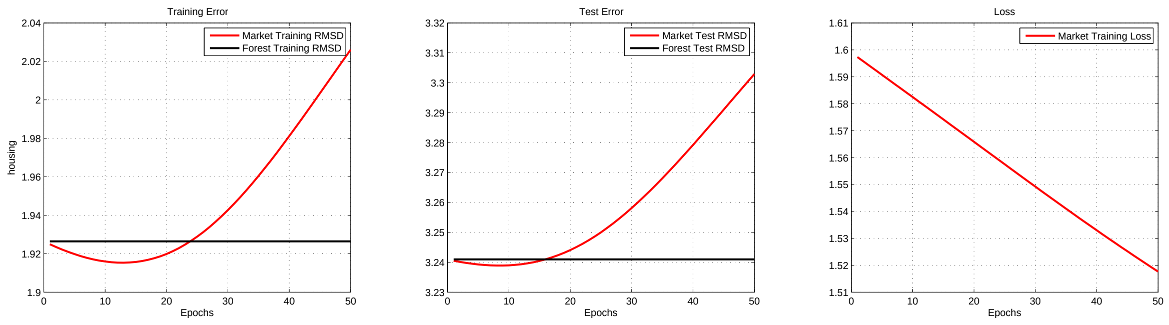
$$E_X[KL(p(y|\mathbf{x}), c(y|\mathbf{x};\boldsymbol{\beta}))] = \int_X p(\mathbf{x}) \int_Y p(y|\mathbf{x}) \log \frac{p(y|\mathbf{x})}{c(y|\mathbf{x};\boldsymbol{\beta})} dy d\mathbf{x} \tag{4.19}$$

In particular, this loss function suggests that the optimal reward kernel is the ground truth conditional $K(t) = p(t|\mathbf{x})$. With this interpretation, the reward kernel can serve as either a prior or as a means to regularize the regression market training.

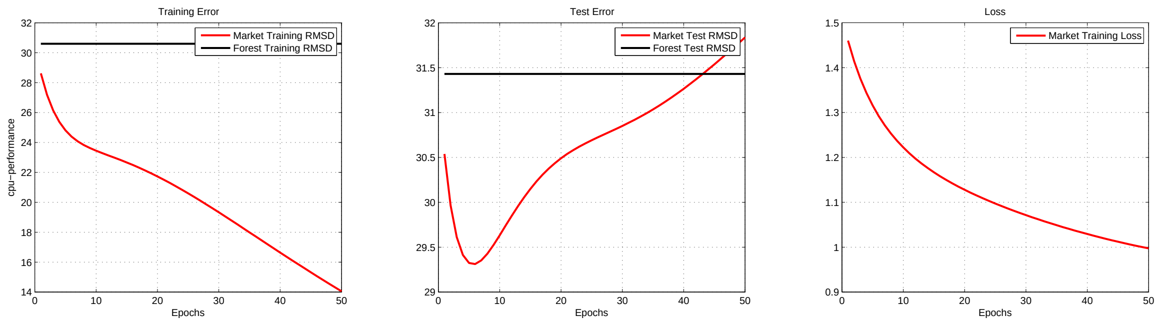
4.2.1 Case Study

To empirically demonstrate the loss function, we considered the evolution of the Regression Market over three data sets: housing, cpu-performance, californiahousing using the incremental update (4.11). The evolution of housing and cpu-performance was recorded over 50 epochs and averaged over 100 runs while the evolution for californiahousing was only recorded over 10 epochs because it overfits in relatively few epochs.

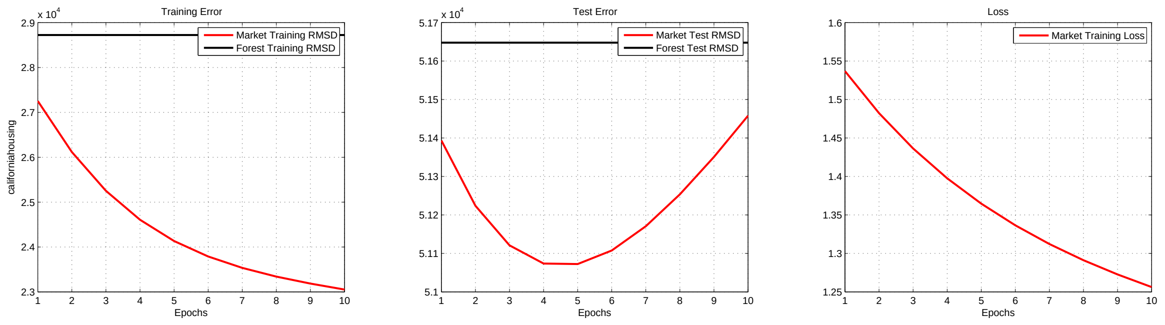
In all examples in figure 4.2, the Regression Market is maximizing the log likelihood. However, it is worth mentioning from the results, that maximizing the log likelihood does not necessarily imply that the training error decreases. The Regression Market is inferring the true unknown conditional density and not the regressor itself.



(a) Training error, test error, and negative log likelihood for the housing data set.



(b) Training error, test error, and negative log likelihood for the cpu-performance data set.



(c) Training error, test error, and negative log likelihood for the californiahousing data set.

Figure 4.2: Training error, test error and negative log likelihood for three data sets.

CHAPTER 5

PREDICTION MARKETS FOR DENSITY ESTIMATION

In real prediction markets, participants bid on the future outcome of an event. In the Classification Market, the event was an instance $\mathbf{x} \in X \subseteq \mathbb{R}^F$ and the outcome y was an element from a discrete and finite set $y \in Y = \{1, 2, \ldots, K\}$. This, in turn, generalized to regression only that the outcome y was an element from an uncountable set $y \in Y \subseteq \mathbb{R}$. In both cases, there was an analog of an *event* and *outcome*. However, depending on your perspective, density estimation either has no analog of an *event* or an *outcome*. Hence, it is not immediately clear how prediction markets solve the density estimation problem. However, the Regression Market update (4.6) provides a clue of how the prediction market can be made to solve the density estimation problem and ultimately how prediction markets work in general.

5.1 Problem Setup

As with classification and Regression Markets, the density estimation problem is solved by aggregating a given set of densities $\{h_m(\mathbf{x})\}_{m=1}^M$ that estimate the true distribution of a given set of instances $\{\mathbf{x}_n\}_{n=1}^N$ with $\mathbf{x}_n \in X \subseteq \mathbb{R}^F$. Each density has a corresponding budget β_m. The general objective is to compute the *price function* $c(\mathbf{x})$ with $c(\mathbf{x}) \geq 0$ and to update the budget β_m of each participant. The *price function* $c(\mathbf{x})$ has no intuitive interpretation as there is notion of *outcome* and therefore no *contract* to price. However, the price function is intended to estimate the true distribution and therefore we constrain $\int_X c(\mathbf{x}) d\mathbf{x} = 1$.

For reasons mentioned in previous paragraphs, the notion of *betting* does not readily apply. However, for now we suppose that *betting* generalizes to density aggregation. That is, these betting functions are defined analogously to (4.1) and (4.2)

$$\phi(\mathbf{x}, c) \geq 0 \tag{5.1}$$

$$\int_X \phi(\mathbf{x}, c) d\mathbf{x} \leq 1 \tag{5.2}$$

And as with the Regression Market, there are no intuitive properties for these functions and we only suppose the following betting function

$$\phi_{\text{constant}}(\mathbf{x}, c) = h(\mathbf{x}) \tag{5.3}$$

The notion of betting defines both the budget update and market equilibrium. Like the Classification and Regression Markets, the budget update is given in terms of the profit which is defined in terms of total spent and winnings. The total spent is a generalization of (4.5) and is given as

$$\text{bet} = \beta_m \int_X \phi_m(\mathbf{x}, c) d\mathbf{x} \tag{5.4}$$

Likewise, we suppose the winnings are a generalization of (4.4)

$$\text{winnings} = \beta_m \int_X K(\mathbf{x}) \frac{\phi_m(\mathbf{x}, c)}{c(\mathbf{x})} d\mathbf{x} \tag{5.5}$$

where $K(\mathbf{x})$ is similar to the *reward kernel* from the Regression Market update. This gives the general budget update rule as

$$\beta_m \leftarrow \beta_m - \eta\beta_m \int_X \phi_m(\mathbf{x}, c) d\mathbf{x} + \eta\beta_m \int_X K(\mathbf{x}) \frac{\phi_m(\mathbf{x}, c)}{c(\mathbf{x})} d\mathbf{x} \tag{5.6}$$

It can be shown that the derivation of the Classification Market equilibrium (3.16) generalizes for the Density Market and is given as

$$c(\mathbf{x}) = \sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x}, c) \tag{5.7}$$

Now assuming the prediction market really does generalize to density aggregation in this way, the *reward kernel* should distribute more winnings to those participants that best describe characteristics of the true distribution $p(\mathbf{x})$. In other words, if a participant approximately shares a common mode with $p(\mathbf{x})$, then the participant *ought* to win more than a participant that does not share a common mode with $p(\mathbf{x})$. Hence, it is *reasonable* to suppose that $K(\mathbf{x}) = p(\mathbf{x})$. Additionally, if the betting function is (5.3) then the update simplifies to

$$\beta_m \leftarrow (1-\eta)\beta_m + \eta\beta_m \int_X p(\mathbf{x}) \frac{h_m(\mathbf{x})}{c(\mathbf{x})} d\mathbf{x} \tag{5.8}$$

Now, since $\mathbf{x}_n \sim p(\mathbf{x})$ then the integral in (5.8) can be approximated with Monte Carlo quadrature, or

$$\int_X p(\mathbf{x}) \frac{h_m(\mathbf{x})}{c(\mathbf{x})} d\mathbf{x} \approx \frac{1}{N} \sum_{n=1}^{N} \frac{h_m(\mathbf{x}_n)}{c(\mathbf{x}_n)}$$

which gives a suspiciously similar update as the Classification Market (3.12) and Regression Market (4.11)

$$\beta_m \leftarrow (1-\eta)\beta_m + \eta\beta_m \frac{1}{N} \sum_{n=1}^{N} \frac{h_m(\mathbf{x}_n)}{c(\mathbf{x}_n)} \tag{5.9}$$

This update appears to be well approximated with the analog of the Classification and Regression Market updates

$$\beta_m \leftarrow (1-\eta)\beta_m + \eta\beta_m \frac{h_m(\mathbf{x})}{c(\mathbf{x})} \tag{5.10}$$

Algorithm 7 Density Market Budget Update $(\mathbf{x}, c)$

```
  **Input:** Training example $\mathbf{x}$, price function $c(\mathbf{x})$
  **for** $m = 1$ **to** $M$ **do**
    Update participant $m$'s budget as
        $$\beta_m \leftarrow (1-\eta)\beta_m + \eta\frac{\beta_m}{c(\mathbf{x})}h_m(\mathbf{x}) \tag{5.11}$$
  **end for**
```

5.2 Expectation-Maximization Algorithm

The constant Density Market can be related to the Expectation Maximization (EM) [30] algorithm in the context of mixture models. In this context, the objective is to associate instances $\mathbf{x}_n,\ n = 1, 2, \ldots, N$ to a single distribution from $p_m(\mathbf{x}),\ m = 1, 2, \ldots, M$ in the mixture. For each $\mathbf{x}_n$ we associate a $z_n \in \{1, 2, \ldots, M\}$ so that

$$\mathbf{x}_n \sim \sum_{m=1}^{M} I(z_n = m)p_m(\mathbf{x})$$

where $I(\cdot)$ denotes the indicator function. Then the density of this mixture model is given as

$$p(\mathbf{x}) = \sum_{m=1}^{M} \pi_m p_m(\mathbf{x})$$

where $\pi_m = \Pr(z = m)$. Since the true form of $p_m(\mathbf{x})$ is often not known, it is chosen to be some prior model with parameters θ_m, or $p_m(\mathbf{x}; \theta_m)$. A popular choice for $p_m(\mathbf{x})$ is a Gaussian with $\theta_m = (\mu_m, \Sigma_m)$ as the mean and covariance matrix respectively. To determine the weights π_m and parameters θ_m, the problem is posed as a maximum log likelihood problem given as

$$\max_{\pi,\theta} \sum_{n=1}^{N} \log \sum_{m=1}^{M} \pi_m p_m(\mathbf{x}_n; \theta_m)$$

However, this is difficult to maximize directly. If instead the log likelihood is written in terms of the latent variables z_n, then the problem becomes

$$\max_{\pi,\theta} \sum_{n=1}^{N} \log \sum_{m=1}^{M} I(z_n = m)p_m(\mathbf{x}_n; \theta_m)$$

And this has the effect of decoupling the problem so that

$$\sum_{n=1}^{N} \log \sum_{m=1}^{M} I(z_n = m) p_m(\mathbf{x}_n; \theta_m) = \sum_{m=1}^{M} \sum_{\mathbf{x} \in X_m} \log p_m(\mathbf{x}_m; \theta_m)$$

where $X_m = \{\mathbf{x}_n \ : \ z_n = m\}$. Now each set of parmeters θ_m can be optimized independently (e.g. with gradient descent, or prior knowledge), greatly simplifying the problem. However, the latent variables z_n are not actually known. Instead, the indicator $I(z_n = m)$ can be replaced by an estimate of $\Pr(z_n = m)$ where

$$\Pr(z_n = m) = \frac{\pi_m p_m(\mathbf{x}_n; \theta_m)}{\sum_{m=1}^{M} \pi_m p_m(\mathbf{x}_n; \theta_m)}$$

Thus, the algorithm proceeds iteratively in two steps, colloquially labeled the E-step and M-step respectively

1. The E-step
   Compute
   $$t_{m,n} = \Pr(z_n = m) = \frac{\pi_m p_m(\mathbf{x}_n; \theta_m)}{\sum_{m=1}^{M} \pi_m p_m(\mathbf{x}_n; \theta_m)}$$
2. The M-step
   Estimate π_m based on the latent variable probabilities
   $$\pi_m \leftarrow \frac{1}{N} \sum_{n=1}^{N} t_{m,n} \tag{5.12}$$
   and maximize over θ
   $$\theta_m \leftarrow \operatorname*{argmax}_{\theta} \sum_{n=1}^{N} \log t_{m,n} p_m(\mathbf{x}_n; \theta)$$
3. Repeat steps 1 and 2 until *convergence.*

The constant Density Market update rule is identical to part of the EM M-step update (5.12). Writing the Density Market update in terms of batch update (3.23) with $\eta = 1$ gives

$$\begin{aligned} \beta_m &\leftarrow \frac{1}{N} \sum_{n=1}^{N} \frac{\beta_m h_m(\mathbf{x}_n)}{c(\mathbf{x}_n)} \\ &= \frac{1}{N} \sum_{n=1}^{N} \frac{\beta_m h_m(\mathbf{x}_n)}{\sum_{m=1}^{M} \beta_m h_m(\mathbf{x}_n)} \\ &= \frac{1}{N} \sum_{n=1}^{N} t_{m,n} \end{aligned}$$

5.3 Loss Function

The Constant Density Market maximizes log likelihood. The loss function is then the negative log likelihood given by

$$\ell(\boldsymbol{\beta}) = -\sum_{\mathbf{x}\in X} \log c(\mathbf{x};\boldsymbol{\beta})$$

Since the constant density market is a particular form of the EM, proofs that EM maximizes log likelihood automatically apply to the constant density market. However, the contraction approach discussed in 3.4.2 readily generalizes to the constant density market.

CHAPTER 6

RESULTS

In this chapter, we consider a set of experiments for the three types of markets: the classification market, the regression market, and the density market. In the classification market, we consider benchmarks on probability estimation, random splits and cross validation. We first evaluate the probability estimation capabilities of the classification market when using three types of betting functions. We then consider and compare three types of betting functions used in the leaves of random trees to the raw random forest output for both our implementation of random forest and Implicit Online Learning and those results (when available) reported by Breiman. Similarly, for the regression task, we consider and compare two updates rules aggregating the leaves of regression trees to the raw regression forest output for both our implementation and those results reported by Breiman. We further investigate the learning power of the regression market using two update rules on shallow trees. For density estimation, we demonstrate preliminary results of the constant market fitting a Gaussian mixture model.

6.1 Classification Market

The Classification Market was first tested on synthetic data to evaluate its accuracy for probability estimation. The synthetic data was crafted in such a way so that it would exhibit 50 different levels of Bayes error. The market was trained on these data sets to determine how well it would approximate the probability for increasingly difficult problems (increasing Bayes error). Then the market was tested on 30 real data sets provided by the UCI machine learning repository [25]. Two experiments were carried out to compare the Classification Market with Random Forest and Implicit Online Learning. The first experiment was a comparison of the Classification Market with three betting strategies, our implementation of Random Forest and Breiman's Random Forest results [9] on randomly split sets. The second experiment was a comparison of the Classification Market with the constant betting strategy (3.12), our implementation of the Random Forest, and Implicit Online Learning on 10-fold cross validation sets. In both experiments, the Classification Market participants were branches of each of the random trees of our implementation of random forest. Each experiment was run on 100 samples and averaged. The comparisons are given in terms of misclassification rates and statistical significance for both mean misclassification rate and pair-wise misclassification rates ($\alpha < 0.01$).

6.1.1 Evaluation of the Probability Estimation and Classification Accuracy on Synthetic Data

We perform a series of experiments on synthetic datasets to evaluate the market's ability to predict class conditional probabilities $P(Y|\mathbf{x})$. The experiments are performed on 5000 binary datasets with 50 levels of Bayes error

$$E = \int \min\{p(\mathbf{x}, Y = 0), p(\mathbf{x}, Y = 1)\} d\mathbf{x},$$

ranging from 0.01 to 0.5 with equal increments. For each dataset, the two classes have equal frequency. Both $p(\mathbf{x}|Y = k), k = 0, 1$ are normal distributions $\mathcal{N}(\mu_k, \sigma^2 I)$, with $\mu_0 = 0, \sigma^2 = 1$ and μ_1 chosen in some random direction at such a distance to obtain the desired Bayes error.

For each of the 50 Bayes error levels, 100 datasets of size 200 were generated using the bisection method to find an appropriate μ_1 in a random direction. Training of the participant budgets is done with $\eta = 0.1$.

For each observation $\mathbf{x}$, the class conditional probability can be computed analytically using the Bayes rule

$$p^*(Y = 1|\mathbf{x}) = \frac{p(\mathbf{x}|Y = 1)p(Y = 1)}{p(\mathbf{x}, Y = 0) + p(\mathbf{x}, Y = 1)}$$

An estimation $\hat{p}(y = 1|\mathbf{x})$ obtained with one of the markets is compared to the true

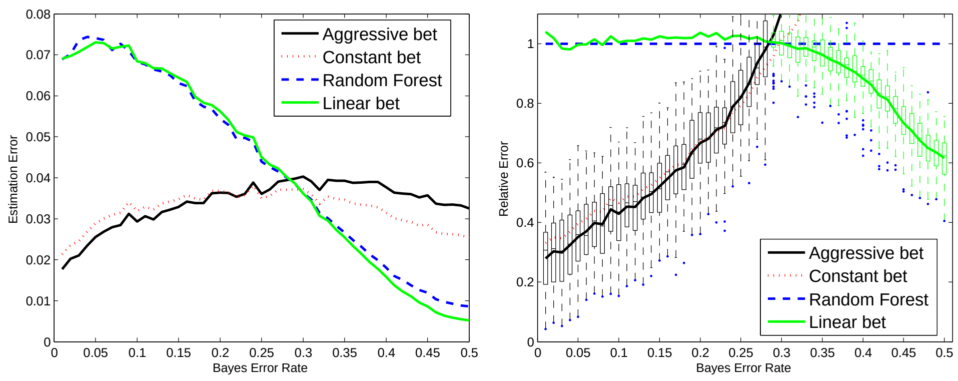


Figure 6.1: Left: Class probability estimation error vs problem difficulty for 5000 100D problems. Right: Probability estimation errors relative to random forest. The aggressive and linear betting are shown with box plots.

probability $p^*(Y = 1|\mathbf{x})$ using the L_2 norm

$$E(\hat{p}, p^*) = \int (\hat{p}(y = 1|\mathbf{x}) - p^*(y = 1|\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x}$$

where $p(\mathbf{x}) = p(\mathbf{x}, Y = 0) + p(\mathbf{x}, Y = 1)$.

In practice, this error is approximated using a sample of size 1000. The errors of the probability estimates obtained by the four markets are shown in Figure 6.1 for a 100D

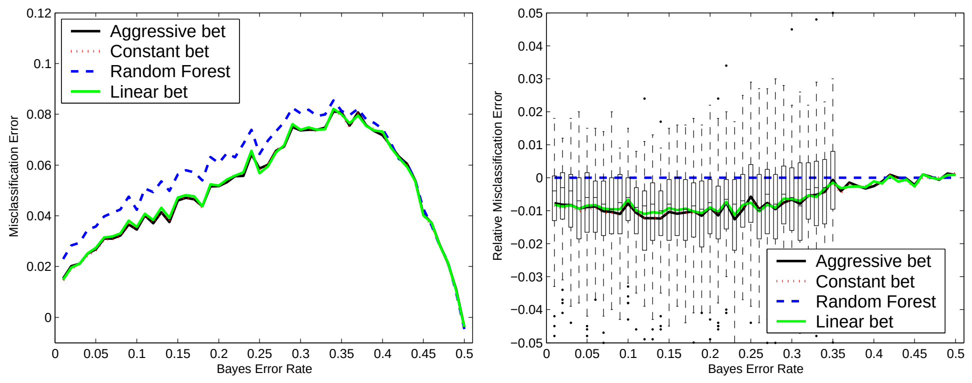


Figure 6.2: Left: Misclassification error minus Bayes error vs problem difficulty for 5000 100D problems. Right: Misclassification errors relative to random forest. The aggressive betting is shown with box plots.

problem setup. Also shown on the right are the errors relative to the random forest, obtained by dividing each error to the corresponding random forest error. As one could see, the aggressive and constant betting markets obtain significantly better (p-value < 0.01) probability estimators than the random forest, for Bayes errors up to 0.28. On the other hand, the linear betting market obtains probability estimators significantly better (p-value < 0.01) than the random forest for Bayes error from 0.34 to 0.5.

We also evaluated the misclassification errors of the four markets in predicting the correct class, for the same 5000 datasets. The difference between these misclassification errors and the Bayes error are shown in Figure 6.2, left. The difference between these misclassification errors and the random forest error are shown in Figure 6.2, right. We see that all markets with trained participants predict significantly better (p-value < 0.01) than random forest for Bayes errors up to 0.3, and behave similar to random forest for the remaining datasets.

6.1.2 Comparison with Random Forest on UCI Datasets

In this section we conduct an evaluation on 31 datasets from the UCI machine learning repository [7]. The optimal number of training epochs and η are meta-parameters that need to be chosen appropriately for each dataset. We observed experimentally that η can take any value up to a maximum that depends on the dataset. In these experiments we took $\eta = 10/N_{train}$. The best number of epochs was chosen by ten fold cross-validation.

In order to compare with the results in [9], the training and test sets were randomly subsampled from the available data, with 90% for training and 10% for testing. The exceptions are the `satimage`, `zipcode`, `hill-valley` and `poker`datasets with test sets of size $2000, 2007, 606, 10^6$ respectively. All results were averaged over 100 runs.

We present two random forest results. In the column named RFB are presented the random forest results from [9]where each tree node is split based on a random feature. In the column named RF we present the results of our own RF implementation with splits based on random features. The leaf nodes of the random trees from our RF implementation are used as specialized participants for all the markets evaluated.

Table 6.1: The misclassification errors for 31 datasets from the UC Irvine Repository are shown in percentages (%). The markets evaluated are our implementation of random forest (RF), and markets with Constant (CB), Linear (LB) and respectively Aggressive (AB) Betting. RFB contains the random forest results from [9].

Data	N_{train}	N_{test}	F	K	ADB	RFB	RF	CB	LB	AB
breast-cancer	683	–	9	2	3.2	2.9	2.7	2.7	2.7	2.7
sonar	208	–	60	2	15.6	15.9	18.1	17	17.4	17
vowel	990	–	10	11	4.1	3.4	4.2	3.6 ●	3.9 ●	3.4 ●
ecoli	336	–	7	8	14.8	12.8	14.5	14.3	14.4	14.3
german	1000	–	24	2	23.5	24.4	23.7	**23.3**	23.3	**23.3**
glass	214	–	9	6	22	20.6	22	21.9	21.9	21.8
image	2310	–	19	7	1.6	2.1	2.1	**1.8** ●	**1.8** ●	**1.8** ●
ionosphere	351	–	34	2	6.4	7.1	6.5	6.2	6.4	6.3
letter-recognition	20000	–	16	26	3.4	3.5	3.3	3.2 ●	3.2 ●	3.2 ●
liver-disorders	345	–	6	2	30.7	25.1	26.5	26.5	26.5	26.6
pima-diabetes	768	–	8	2	26.6	24.2	24.4	24.3	24.2	24.3
satimage	4435	2000	36	6	8.8	8.6	9.1	8.8 ●	8.9 ●	8.8 ●
vehicle	846	–	18	4	23.2	25.8	24.3	**23.6**	**24.2**	**23.6**
voting-records	232	–	16	2	4.8	4.1	4.1	4.1	4.1	4.1
zipcode	7291	2007	256	10	6.2	6.3	6.1	**6.2** †	**6.1**	**6.1**
abalone	4177	–	8	3	–	–	44.7	44.7	44.6	44.7
balance-scale	625	–	4	3	–	–	14	14.1	14.1	14.5 †
car	1728	–	6	4	–	–	2.5	0.9 ●	1.2 ●	0.9 ●
connect-4	67557	–	42	3	–	–	19.9	16.7 ●	16.9 ●	16.7 ●
cylinder-bands	277	–	33	2	–	–	22.5	22.7	22.5	22.5
hill-valley	606	606	100	2	–	–	45.1	44.4 ●	44.8 ●	44.5 ●
isolet	1559	–	617	26	–	–	7.6	7.4	7.5	7.4 ●
king-rk-vs-king	28056	–	6	18	–	–	21.6	11.0 ●	11.8 ●	11.0 ●
king-rk-vs-k-pawn	3196	–	36	2	–	–	1.2	0.4 ●	0.5 ●	0.4 ●
magic	19020	–	10	2	–	–	12.0	11.7 ●	11.8 ●	11.8 ●
madelon	2000	–	500	2	–	–	31.2	23 ●	23.1 ●	23 ●
musk	6598	–	166	2	–	–	2.2	1.1 ●	1.2 ●	1.1 ●
splice-junction-gene	3190	–	59	3	–	–	4.6	4.1 ●	4.2 ●	4.1 ●
SAheart	462	–	9	2	–	–	31.2	31.3	31.3	31.3
yeast	1484	–	8	10	–	–	37.8	37.9	37.9	37.7

The CB, LB and AB columns are the performances of the constant, linear and respectively aggressive markets on these datasets.

Significant mean differences ($\alpha < 0.01$) from RFB are shown with $+, -$ for when RFB is worse respectively better. Significant paired t-tests [20] ($\alpha < 0.01$) that compare the markets with our RF implementation are shown with $\bullet, \dagger$ for when RF is worse respectively better.

The constant, linear and aggressive markets significantly outperformed our RF implementation on 22, 19 respectively 22 datasets out of the 31 evaluated. They were not significantly outperformed by our RF implementation on any of the 31 datasets.

Compared to the RF results from [9] (RFB), CB, LB and AB significantly outperformed RFB on 6,5,6 datasets respectively, and were not significantly outperformed on any dataset.

6.1.3 Comparison with Implicit Online Learning

We implemented the implicit online learning [32] algorithm for classification with linear aggregation. The objective of implicit online learning is to minimize the loss $\ell(\beta)$ in a *conservative* way. The *conservativeness* of the update is determined by a Bregman divergence

$$D(\beta, \beta^t) = \phi(\beta) - \phi(\beta^t) - \langle \nabla\phi(\beta^t), \beta - \beta^t \rangle$$

where $\phi(\beta)$ are real-valued strictly convex functions. Rather than minimize the loss function itself, the function

$$f_t(\beta) = D(\beta, \beta^t) + \eta_t \ell(\beta)$$

is minimized instead. Here η_t is the learning rate. The Bregman divergence ensures that the optimal β is not *too far* from β^t. The algorithm for implicit online learning is as follows

$$\begin{aligned}\tilde{\beta}^{t+1} &= \underset{\beta \in \mathbb{R}^M}{\text{argmin}} f_t(\beta) \\ \beta^{t+1} &= \underset{\beta \in S}{\text{argmin}} D(\beta, \tilde{\beta}^{t+1})\end{aligned}$$

The first step solves the unconstrained version of the problem while the second step finds the *nearest* feasible solution to the unconstrained minimizer subject to the Bregman divergence.

For our problem we use

$$\ell(\beta) = -\log(c_y(\beta))$$

where $c_y(\beta)$ is the constant market equilibrium price for ground truth label y. We chose the squared Euclidean distance $D(\beta, \beta^t) = \|\beta - \beta^t\|_2^2$ as our Bregman divergence and learning rate $\eta_t = 1/\sqrt{t}$. To ensure that $c = \sum_{m=1}^M h_m \beta_m = H\beta$ is a valid probability vector, the feasible solution set is therefore $S = \{\beta \in [0,1]^M \ : \ \sum_{m=1}^M \beta_m = 1\}$. This gives the following update scheme

$$\begin{aligned}\tilde{\beta}^{t+1} &= \beta^t + \eta_t \frac{1}{p}(H^y)^T \\ \beta^{t+1} &= \underset{\beta \in S}{\text{argmin}} \left\{ \|\beta - \tilde{\beta}^{t+1}\|_2^2 \right\}\end{aligned}$$

where $H^y = \left(h_1^y,\ h_2^y,\ \ldots,\ h_M^y\right)$ is the vector of classifier outputs for the true label y, $q = H^y\beta^t,\ r = H^y(H^y)^T$ and $p = \frac{1}{2}\left(q + \sqrt{q^2 + 4\eta_t r}\right)$.

The results presented in Table 6.2 are obtained by 10 fold cross-validation. The cross-validation errors were averaged over 10 different permutations of the data in the cross-validation folds.

The results from CB online and implicit online are obtained in one epoch. The results from the CB offline and implicit offline columns are obtained in an off-line fashion using an appropriate number of epochs (up to 10) to obtain the smallest cross-validated error on a random permutation of the data that is different from the 10 permutations used to obtain the results.

The comparisons are done with paired t-tests and shown with $*$ and ‡ when the constant betting market is significantly ($\alpha < 0.01$) better or worse than the corresponding implicit online learning. We also performed a comparison with our RF implementation, and significant differences are shown with • and †.

Compared to RF, implicit online learning won 5-0, CB online won in 9-1 and CB offline won 12-0.

Compared to implicit online, which performed identical with implicit offline, both CB online and CB offline won 9-0.

The offline constant market performs best in many cases and is significantly better than Implicit Online Learning and random forest.

6.1.4 Comparison with Adaboost for Lymph Node Detection

Finally, we compared the linear aggregation capability of the artificial prediction market with adaboost for a lymph node detection problem. The system is setup as described in [4], namely a set of lymph node candidate positions (x, y, z) are obtained using a trained detector. Each candidate is segmented using gradient descent optimization and about 17000 features are extracted from the segmentation result. Using these features, adaboost constructed 32 weak classifiers. Each weak classifier is associated with one feature, splits the feature range into 64 bins and returns a predefined value (1 or -1), for each bin.

Thus, one can consider there are $M = 32 \times 64 = 2048$ specialized participants, each betting for one class (1 or -1) for any observation that falls in its domain. The participants are given budgets $\beta_{ij}, i = 1, .., 32, j = 1, .., 64$ where i is the feature index and j is the bin index. The participant budgets $\beta_{ij}, j = 1, ..., 64$ corresponding to the same feature i are initialized the same value β_i, namely the adaboost coefficient. For each bin, the return class 1 or -1 is the outcome for which the participant will bet its budget.

The constant betting market of the 2048 participants is initialized with these budgets and trained with the same training examples that were used to train the adaboost classifier.

The obtained constant market probability for an observation $\mathbf{x} = (x_1, ..., x_{32})$ is based on the bin indexes $\mathbf{b} = (b_1(x_1), ..., b_{32}(x_{32}))$:

$$p(y = 1|\mathbf{b}) = \frac{\sum_{i=1}^{32} \beta_{i,b_i} h_i(b_i)}{\sum_{i=1}^{32} \beta_{i,b_i}} \tag{6.1}$$

An important issue is that the number N_{pos} of positive examples is much smaller than the number N_{neg} of negatives. Similar to adaboost, the sum of the weights of the positive

Table 6.2: Testing misclassification rates of our implementation of Random Forest (RF), Implicit Online Learning [32], and Constant Betting (CB). ● indicates statistically significantly better than (RF), † indicates statistically significantly worse than (RF) and ∗ indicates statistically significantly better than Implicit Online/Offline Learning.

Dataset	N_{train}	N_{test}	F	K	RF	Implicit Online	CB Online	Implicit Offline	CB Offline
breast-cancer	683	–	9	2	3.1	3.1	3	3.1	3
sonar	208	–	60	2	15.1	15.2	15.3	15.1	14.6
vowel	990	–	10	11	3.2	3.2	3.2	3.2	2.9 ●∗
ecoli	336	–	7	8	13.7	13.7	13.6	13.7	13.6
german	1000	–	24	2	23.6	23.5	23.5	23.5	23.4
glass	214	–	9	6	21.4	21.4	21.3	21.4	21
image	2310	–	19	7	1.9	1.9	1.9	1.9	1.8 ●
ionosphere	351	–	34	2	6.4	6.5	6.5	6.5	6.5
letter-recognition	20000	–	16	26	3.3	3.3	3.3 ●∗	3.3	3.3
liver-disorders	345	–	6	2	26.4	26.4	26.4	26.4	26.4
pima-diabetes	768	–	8	2	23.2	23.2	23.2	23.2	23.2
satimage	4435	2000	36	6	8.8	8.8	8.8	8.8	8.7 ●
vehicle	846	–	18	4	24.8	24.7	24.9	24.7	24.9
voting-records	232	–	16	2	3.5	3.5	3.5	3.5	3.5
zipcode	7291	2007	256	10	6.1	6.1	6.2	6.1	6.2
abalone	4177	–	8	3	45.5	45.5	45.6 †	45.5	45.5
balance-scale	625	–	4	3	17.7	17.7	17.7	17.7	17.7
car	1728	–	6	4	2.3	2.3	1.8 ●∗	2.3	1.1 ●∗
connect-4	67557	–	42	3	19.9	19.9 ●	19.5 ●∗	19.9 ●	18.2 ●∗
cylinder-bands	277	–	33	2	21.4	21.3	21.2	21.3	20.8 ●
hill-valley	606	606	100	2	43.8	43.7	43.7	43.7	43.7
isolet	1559	–	617	26	6.9	6.9	6.9	6.9	6.9
king-rk-vs-king	28056	–	6	18	21.6	21.6 ●	19.6 ●∗	21.5 ●	15.7 ●∗
king-rk-vs-k-pawn	3196	–	36	2	1	1	0.7 ●∗	1	0.5 ●∗
magic	19020	–	10	2	11.9	11.9 ●	11.8 ●∗	11.9 ●	11.7 ●∗
madelon	2000	–	500	2	26.8	26.5 ●	25.6 ●∗	26.4 ●	21.6 ●∗
musk	6598	–	166	2	1.7	1.7 ●	1.6 ●∗	1.7 ●	1 ●∗
splice-junction-gene	3190	–	59	3	4.3	4.3	4.2 ●∗	4.3	4.1 ●∗
SAheart	462	–	9	2	31.5	31.5	31.6	31.5	31.6
yeast	1484	–	8	10	37.3	37.3	37.3	37.3	37.3

examples should be the same as the sum of weights of the negatives. To accomplish this in the market, we use the weighted update rule Eq. (3.27), with $w_{pos} = \frac{1}{N_{pos}}$ for each positive example and $w_{neg} = \frac{1}{N_{neg}}$ for each negative.

The adaboost classifier and the constant market were evaluated for a lymph node detection application on a dataset containing 54 CT scans of the pelvic and abdominal region, with a total of 569 lymph nodes, with six-fold cross-validation. The evaluation criterion is the same for all methods, as specified in [4]. A lymph node detection is considered correct if its center is inside a manual solid lymph node segmentation and is incorrect if it not inside any lymph node segmentation (solid or non-solid).

In Figure 6.3, left, is shown the training and testing detection rate at 3 false positives per volume (a clinically acceptable false positive rate) vs the number of training epochs. We see the detection rate increases to about 81% for epochs 6 to 16 epochs and then gradually decreases. In Figure 6.3, right, are shown the training and test ROC curves of adaboost and the constant market trained with 7 epochs. In this case the detection rate at 3 false

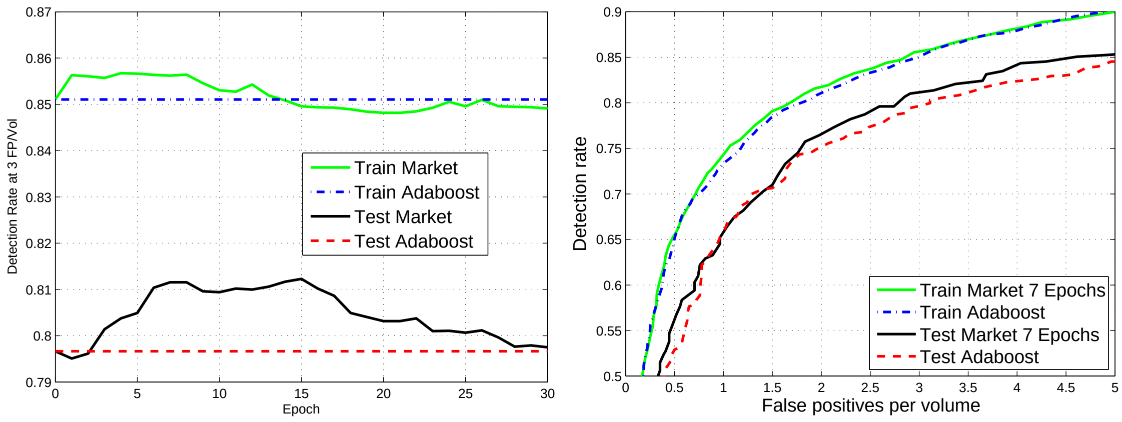


Figure 6.3: Left: Detection rate at 3 FP/vol vs. number of training epochs for a lymph node detection problem. Right: ROC curves for adaboost and the constant betting market with participants as the 2048 adaboost weak classifier bins. The results are obtained with six-fold cross-validation.

positives per volume improved from 79.6% for adaboost to 81.2% for the constant market. The p-value for this difference was 0.0276 based on paired t-test.

6.2 Regression Market

The Regression Market was tested on real and synthetic data sets provided by the UCI machine learning repository and LIAAD [49]. The experimental setup was similar to that of section 6.1. The Regression Market participants were branches of trained regression trees and compared with regression forest using the same regression trees. The regression tree branches themselves do not produce a probability estimate but an estimate for y and a local estimate of the variance σ^2. To make these compatible with the Regression Market, the estimate y and sample variance σ^2 were *wrapped* in a Gaussian density

$$p(y|\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-f(\mathbf{x}))^2}{2\sigma^2}}$$

See figure 6.4 for an example of Gaussians in tree leaves.

The regression tree branches participate in the Regression Market by way of a Gaussian. The Regression Market estimates for y were computed as an expected value of the equilibrium price

$$E_c[y] = \int_Y yc(y|\mathbf{x})dy = \sum_{m=1}^{M} \beta_m f_m(\mathbf{x}) \tag{6.2}$$

where $f_m(\mathbf{x})$ are each participant's estimate of the ground truth y.

We performed two types of experiments with both updates (4.11), (4.13) and compared with Breiman's original regression results [9] as well as additional data sets from UCI and LIAAD [49]. To be consistent with Breiman, nearly all experiments were conducted over

100 random splits where each split randomly sets aside 10% of the data set for testing. For abalone, only 10 random splits with 25% of the data set aside for testing were considered. Data sets with provided test sets were not randomly split. Instead, the forest and markets were trained 100 times on the entire training set and tested on the provided test set. These results vary due to the randomness of the regression forest.

All experiments were run on Windows 7 with 8GB of RAM and Core i7-2630QM processor (max 2.9GHz, 6MB L3 cache). On each training set 100 regression trees were trained. Each regression tree node considered 25 randomized features, each a linear combination of 2 random inputs. Each coefficient of the linear combination was uniformly picked from $[-1, 1]$. In our implementation, 1000 of these random features were generated in advance rather than at each node. The split criteria for each node is based on the weighted sample variance. The rule "don't split if the sample size is < 5" was enforced. Additionally, our implementation treats categoricals as numeric inputs which differs from Breiman's implementation. However, most data sets are comprised of numeric inputs.

Both market types were trained and evaluated over 50 epochs. Each epoch is one complete pass through the training set. The reported errors are those that minimize the MSE of the test set over the 50 epochs (averaged over the 100 runs).

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(f(\mathbf{x}_n) - y_n)^2 \tag{6.3}$$

The learning rate $\eta = \frac{10}{N_{\text{train}}}$ was used as in [3]. On the first run (random split or full training set), the parameter σ for the Gaussian Market reward kernel was estimated using 2-fold cross validation on the training set. This σ remained constant for the other 99 runs (9 runs for abalone). The prediction for y was computed with expectation

$$y = \int_Y tc(t|\mathbf{x})dt = \sum_{m=1}^{M}\beta_m f_m(\mathbf{x}) \tag{6.4}$$

In every result, significance is measured with significance level $\alpha = 0.01$ in two ways: pairwise t-test [20] and t-test on the means. The pairwise t-test was used to compare the 100 market runs with the 100 forest runs while the t-test on the means were compared with Breiman's reported results.

6.2.1 Comparison with Random Forest Regression

The first experiment considers aggregation of tree leaves of forests with fully grown trees on UCI and LIAAD data sets. The results of seven of the data sets are compared with Breiman's reported results. The missing data set Robot Arm is private.

From table 6.3 our RF doesn't perform identically with RFB. This can be attributed to the synthetic nature of some data sets such as friedman1, friedman2, and friedman3 and/or the fact that our implementation of regression forest does not treat categorical inputs the same way. Of the Breiman comparisons, only GM is legitimately significantly better than Breiman's results for friedman2. Out of all the data sets, DM is significantly better than RF for 12 data sets (in a pairwise sense) while GM is only significantly better than RF for 11 data sets. However, DM is significantly worse than RF for 3 data sets while GM is only significantly worse on 2 data sets. The significantly worse results can be attributed to overfitting and/or poorly tuned reward kernel in the case of GM.

Table 6.3: Table of MSE for forests and markets on UCI and LIAAD data sets. The F column is the number of inputs, Y is the range of regression, RFB is Breiman's reported error, RF is our forest implementation, DM is the Market with delta updates, and GM is the Market with Gaussian updates. Bullets/daggers represent pairwise significantly better/worse than RF while +/– represent significantly better/worse than RFB.

Data	N_{train}	N_{test}	F	Y	RFB	RF	DM	GM
abalone	4177	–	8	$[1.00, 29.00]$	4.600	4.571	4.571	4.571
friedman1	200	2000	10	$[4.30, 26.03]$	5.700	4.343+	4.335•+	4.193•+
friedman2	200	2000	4	$[-167.99, 1633.87]$	19600.0	19431.852	19232.482•	18369.546•+
friedman3	200	2000	4	$[0.13, 1.73]$	0.022	0.028–	0.028•–	0.026•–
housing	506	–	13	$[5.00, 50.00]$	10.200	10.471	10.130•	10.128•
ozone	330	–	8	$[1.00, 38.00]$	16.300	16.916	16.925	16.917
servo	167	–	4	$[0.13, 7.10]$	0.246	0.336	0.295	0.322
ailerons	7154	6596	40	$[-0.00, -0.00]$	–	2.814e-008	2.814e-008•	2.814e-008•
auto-mpg	392	–	7	$[9.00, 46.60]$	–	6.469	6.444	6.405•
auto-price	159	–	15	$[5118.00, 35056.00]$	–	3823550.43	3723413.430	3815863.98
bank	4500	3693	32	$[0.00, 0.67]$	–	7.238e-003	7.212e-003•	7.210e-003•
breast cancer	194	–	32	$[1.00, 125.00]$	–	1112.270	1112.509	1108.325
cartexample	40768	–	10	$[-12.69, 12.20]$	–	1.233	1.233†	1.232•
computeractivity	8192	–	21	$[0.00, 99.00]$	–	5.414	5.398•	5.414†
diabetes	43	–	2	$[3.00, 6.60]$	–	0.415	0.426†	0.415
elevators	8752	7847	18	$[0.01, 0.08]$	–	9.319e-006	9.288e-006•	9.225e-006•
forestfires	517	–	12	$[0.00, 1090.84]$	–	5834.819	5844.493†	5680.131•
kinematics	8192	–	8	$[0.04, 1.46]$	–	0.013	0.013•	0.013•
machine	209	–	6	$[6.00, 1150.00]$	–	3154.521	2991.798•	3042.336
poletelecomm	5000	10000	48	$[0.00, 100.00]$	–	29.813	28.855•	29.863†
pumadyn	4499	3693	32	$[-0.09, 0.09]$	–	9.237e-005	8.917e-005•	8.888e-005•
pyrimidines	74	–	27	$[0.10, 0.90]$	–	0.013	0.013	0.012
triazines	186	–	60	$[0.10, 0.90]$	–	0.015	0.015	0.015

The Regression Market is almost always significantly better than our implementation of regression forest. It is significantly worse on cart, forestfires, and pima. This may be due to too large a value of the learning rate η. Neither Regression Market nor our implementation of regression forest match Breiman's regression forest. This may be due to differences in our implementation and/or the fact that Breiman considers random linear combinations of two features while we consider $\sqrt{F}$ features.

6.2.2 Fast Regression using Shallow Trees

This experiment examined the aggregation capabilities of the regression market with shallow trees. In many problems, it is prohibitively expensive to train and even evaluate deep trees. In practice this is mitigated by enforcing a maximum tree depth. For example in [18] and [45] the regression trees were constrained to depth 7. However, this strict constraint on tree depth is prone to introduce leaves that do not generalize well due to prematurely halting tree growth. The specialized regression market of tree leaves can be used to weight the leaves. Poorly performing leaves will tend to have less weight thus improving the overall prediction accuracy.

In addition to the previously mentioned experiment details, regression trees were grown with a maximum depth of 10. Using the same depth 10 trees, MSE errors were computed for leaves no deeper than depth 5. Figure 6.5 serves as an example of how a depth 5 tree was evaluated from a depth 10 tree. Both depth 5 and depth 10 evaluations for training and test sets were recorded. The timings for the larger of the two sets were averaged over the 100 runs and used to compute the speedup. The markets were applied to the depth 5 leaves only. Since the market is just a linear aggregation of 100 leaves per instance, the reported speedup for forest is similar to the speedup of the market.

From table 6.4 it can be seen that the depth 5 forest is roughly twice the speed of the depth 10 forest. On diabetes, the small data set, features and forest likely fit in cache giving the strange 0.7 speedup. DM performs significantly better than RF on seven data sets (in a pairwise set) while DM only performs significantly better on six data sets. However, DM performs significantly worse on two data sets while GM performs significantly worse on one. No method legitimately performs significantly better than RFB since RF is already better than RFB on those two data sets. The significantly worse results can be attributed to overfitting and/or poorly tuned reward kernel in the case of GM.

Table 6.4: Table of MSE for depth 5 forests and markets on UCI and LIAAD data sets. The F column is the number of inputs, Y is the range of regression, RFB is Breiman's reported error (these errors are from fully grown trees), RF is our forest implementation, DM is the Market with delta updates, and GM is the Market with Gaussian updates, and Speedup is the speedup factor of a depth 5 tree versus a depth 10 tree for evaluation. Bullets/daggers represent pairwise significantly better/worse than RF while +/– represent significantly better/worse than RFB.

Data	N_{train}	N_{test}	F	Y	RFB	RF	DM	GM	Speedup
abalone	4177	–	8	$[1.00, 29.00]$	4.600	4.438	4.318•+	4.438	3.3
friedman1	200	2000	10	$[4.30, 26.03]$	5.700	5.076+	4.701•+	4.429•+	1.8
friedman2	200	2000	4	$[-167.99, 1633.87]$	19600.0	29343.562–	23200.438•–	21183.421•–	1.9
friedman3	200	2000	4	$[0.13, 1.73]$	0.022	0.034–	0.029•–	0.028•–	2.0
housing	506	–	13	$[5.00, 50.00]$	10.200	12.869–	12.056•–	11.947•–	2.2
ozone	330	–	8	$[1.00, 38.00]$	16.300	16.976	16.964	16.932	2.1
servo	167	–	4	$[0.13, 7.10]$	0.246	0.248	0.241	0.254	1.6
auto-mpg	392	–	7	$[9.00, 46.60]$	–	8.248	7.817•	7.750•	2.1
auto-price	159	–	15	$[5118.00, 35056.00]$	–	4699789.7	4524741.81	4431992.3	1.4
breast cancer	194	–	32	$[1.00, 125.00]$	–	1073.319	1071.820	1072.126	2.1
diabetes	43	–	2	$[3.00, 6.60]$	–	0.400	0.426†	0.393	0.7
forestfires	517	–	12	$[0.00, 1090.84]$	–	4945.630	5445.001†	5196.451†	2.2
machine	209	–	6	$[6.00, 1150.00]$	–	3137.001	3127.932	2930.506	1.8
triazines	186	–	60	$[0.10, 0.90]$	–	0.016	0.015•	0.015•	2.0

6.3 Density Market

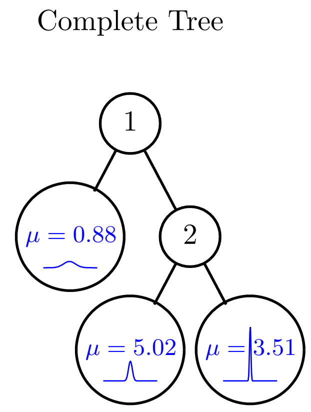
To test whether the Density Market can really fit distributions, we consider mixture models of Gaussians in both one and two dimensions. In one dimension, we generate a random mixture model and sample 1000 points with which to train the Density Market. In two dimensions, we consider EM *clustering* on a cloud of 1000 points describing a circle to infer Gaussian participants for the Density Market. Even though EM does not cluster this type of data very well, the objective was to generate participants for the Density Market.

6.3.1 Fitting 1D Gaussians

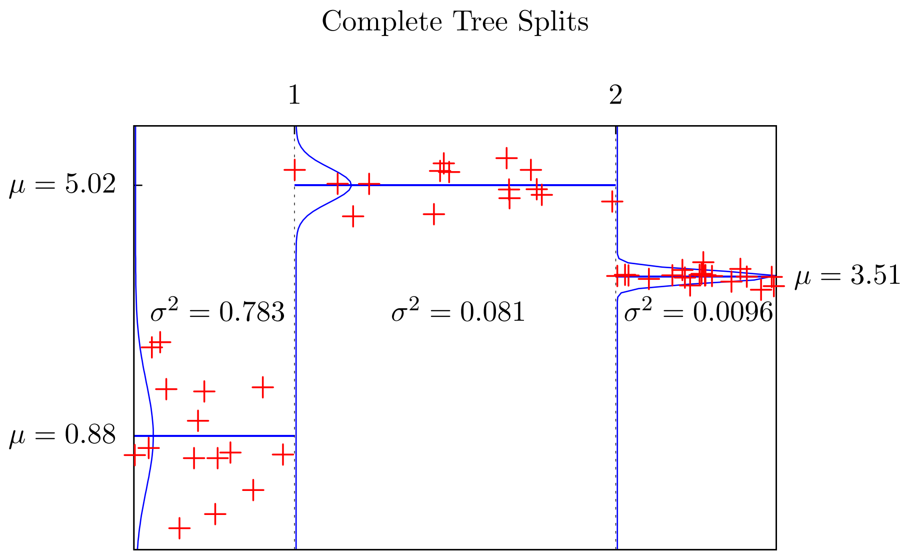
In this experiment, we considered four randomized mixture models composed of 10 Gaussians with randomized mean and variance. For each mixture, we considered fitting the true mixture model with 100 Gaussians including the 10 mixture Gaussians, 5 Gaussians of which 5 are from the 10 mixture Gaussians, 100 randomized Gaussians, and 5 randomized Gaussians. The Density Market was trained on a sample of size 1000 over four epochs. The evolution of the market over the four epochs in all four cases have been plotted in figure 6.6. The Density Market converges in just a few epochs. The performance depends on the how well the participants approximate the ground truth constituents. For example in 6.6(b), the participants are the ground truth constituents and the Density Market can fit the mixture relatively well.

6.3.2 Fitting 2D Gaussians

In this experiment, we considered 2D Gaussians inferred by EM clustering on a cloud of points describing a circle. We repeatedly inferred 10 Gaussians initialized randomly for a total of 100 Gaussians. We then trained a Density Market with these 100 Gaussians to describe the distribution of points on the circle. Figure 6.7 illustrate the data points, cluster centers and resulting trained Density Market. The Gaussians inferred by EM will not necessarily fit the points well since points sampled along a circle do not behave like points sampled from Gaussian distributions. However, the Density Market can be used to weed out the poorly fit Gaussians. The budget configuration shown in figure 6.7(b) illustrates that a large proportion of the participants have gone bankrupt (i.e. $\beta_m = 0$).



(a)



(b)

Figure 6.4: These figures demonstrate specialized Gaussian participants in a regression tree. The numbered nodes in figure (a) correspond to the region splits in figure (b). Each leaf stores the mean y value and estimated variance σ^2 for its partition and use these as the Gaussian parameters.

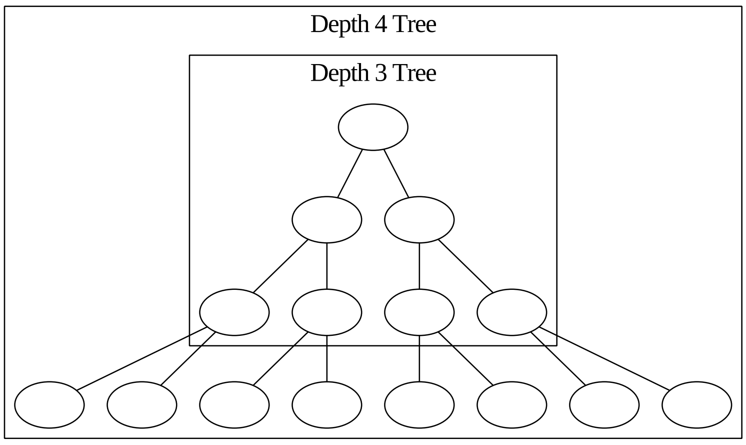
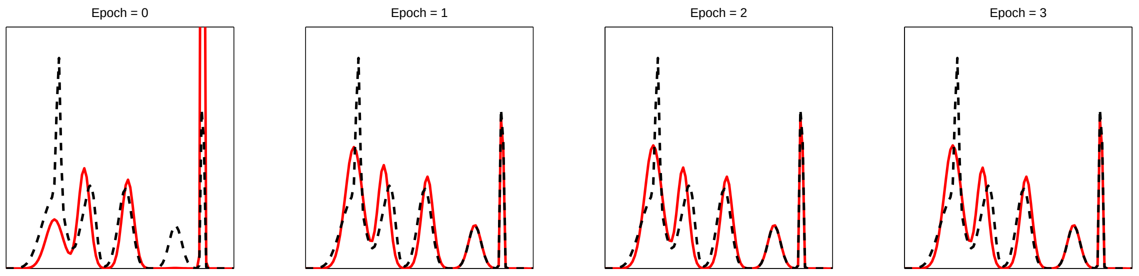
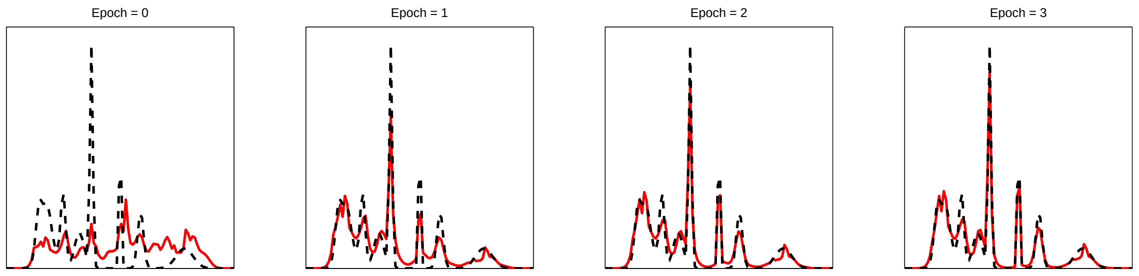


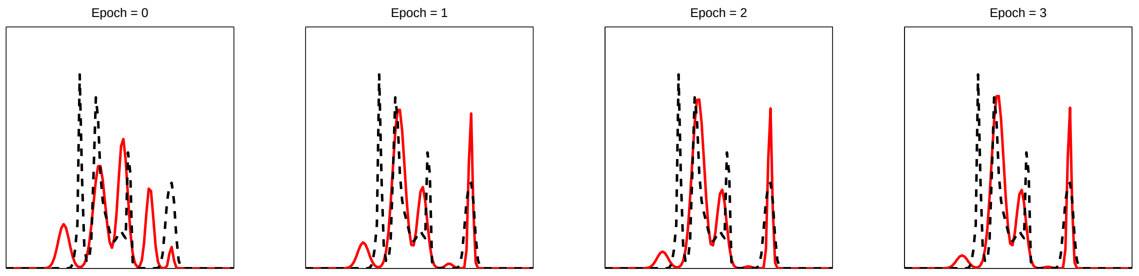
Figure 6.5: Examples of tree depths. A depth 3 tree may be evaluated from a depth 4 tree by considering only the depth 3 subtree. This serves as an example of how a depth 5 tree was evaluated from a depth 10 tree for comparison in the aggregation of shallow regression tree leaves.



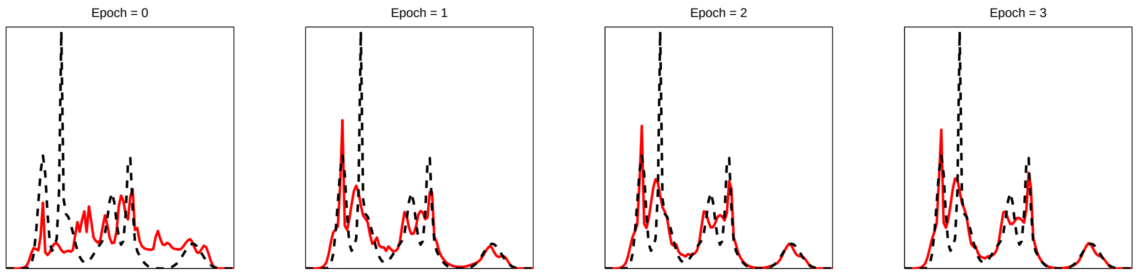
(a) Density Market evolution with 5 Gaussians, all of which are 5 of the true Gaussians fitting a mixture of 10 Gaussians.



(b) Density Market evolution with 100 Gaussians with the 10 true Gaussians fitting a mixture of 10 Gaussians.

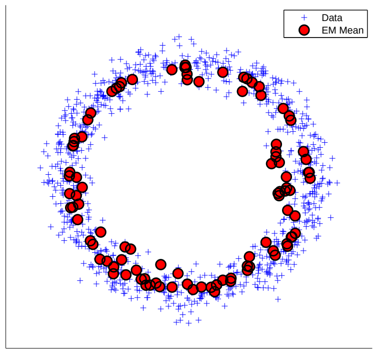


(c) Density Market evolution with 5 randomized Gaussians fitting a mixture of 10 Gaussians.

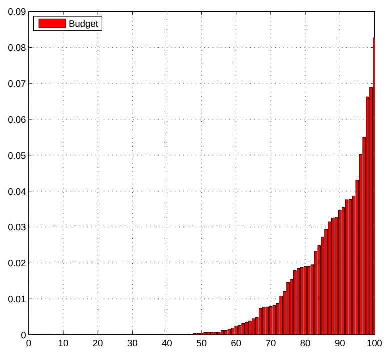


(d) Density Market evolution with 100 randomized Gaussians fitting a mixture of 10 Gaussians.

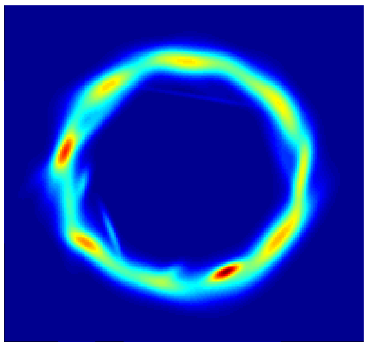
Figure 6.6: These figures illustrate the Density Market fitting Gaussians (red) to a set of data points sampled from the ground truth (black dashes).



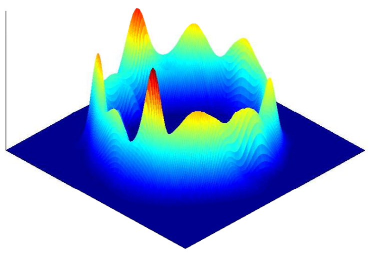
(a) The circle data with corresponding inferred EM Gaussian means.



(b) The sorted budget configuration of the trained Density Market.



(c) An intensity plot of the trained Density Market viewed from above.



(d) A 3D view of the trained Density Market.

Figure 6.7: These figures illustrate the Density Market fitting 2D Gaussians inferred by EM to points sampled along a circle as well as the resulting budgets sorted (β). Many poorly fit Gaussians are weeded out by the market.

CHAPTER 7

PROSPECTIVE IDEAS

This chapter covers a collection of ideas for further development and applications of the prediction market. These were very briefly explored but not in any great detail.

7.1 Market Transform

At the time of this writing, all work on Articial Prediction Markets have assumed discrete and finite participants, indexing budgets and betting functions with the integers β_m and ϕ_m for $m = 1, 2, \ldots, M$. However, suppose that there were uncountably many participants? Suppose a family of betting functionals were indexed over a parameter space $\theta \in \Theta$ with budget $\beta(\theta)$ and participants $\phi(\cdot; \theta)$. It is tempting to suppose that the bets and winnings generalize. Suppose, for example, the case of the regression market, then

$$\begin{aligned}\text{Bet} &= \beta(\theta) \int_Y \phi(t|\mathbf{x}, c; \theta) dt \\ \text{Winnings} &= \beta(\theta) \int_Y p(t|\mathbf{x}) \frac{\phi(t|\mathbf{x}, c; \theta)}{c(t|\mathbf{x}; \beta)} dt\end{aligned}$$

where $p(t|\mathbf{x})$ is the ground truth and reward kernel. Therefore, this would produce the following update rule

$$\beta(\theta) \leftarrow \beta(\theta) - \beta(\theta) \int_Y \phi(t|\mathbf{x}, c; \theta) dt + \beta(\theta) \int_Y p(t|\mathbf{x}) \frac{\phi(t|\mathbf{x}, c; \theta)}{c(t|\mathbf{x}; \beta)} dt$$

Likewise, the equilibrium price functional $c(y|\mathbf{x}; \beta)$ would be computed such that winnings and losses matched, or

$$\int_\Theta \beta(\theta) \int_Y p(t|\mathbf{x}) \frac{\phi(t|\mathbf{x}, c; \theta)}{c(t|\mathbf{x}; \beta)} dt d\theta = \int_\Theta \beta(\theta) \int_Y \phi(t|\mathbf{x}, c; \theta) dt d\theta$$

However, there is no clear solution to these functional equations even in the case of the regression and density markets. However, if $\phi(t|\mathbf{x}, c; \theta) = h(t|\mathbf{x}; \theta)$, then solving the equilibrium equation is trivial and reduces to

$$\int_\Theta \left(\beta(\theta) \int_Y p(t|\mathbf{x}) \frac{h(t|\mathbf{x}; \theta)}{c(t|\mathbf{x}; \beta)} dt - \beta(\theta) \int_Y h(t|\mathbf{x}; \theta) dt \right) d\theta = 0$$

If the price functional is assumed to be similarly defined as in the classification, regression and density markets

$$c(t|\mathbf{x};\beta) = \int_\Theta \beta(\theta) h(t|\mathbf{x};\theta) d\theta$$

at least almost everywhere, then it is trivial to show that budget is conserved and that this is really an equilibrium price functional

$$\begin{aligned}
&\int_\Theta \left(\beta(\theta)\int_Y p(t|\mathbf{x})\frac{h(t|\mathbf{x};\theta)}{c(t|\mathbf{x};\beta)}dt - \beta(\theta)\int_Y h(t|\mathbf{x};\theta)dt\right)d\theta \\
&= \int_\Theta\int_Y \left(\beta(\theta)p(t|\mathbf{x})\frac{h(t|\mathbf{x};\theta)}{c(t|\mathbf{x};\beta)} - \beta(\theta)h(t|\mathbf{x};\theta)dt\right)dtd\theta \\
&= \int_Y\int_\Theta \left(\beta(\theta)p(t|\mathbf{x})\frac{h(t|\mathbf{x};\theta)}{c(t|\mathbf{x};\beta)} - \beta(\theta)h(t|\mathbf{x};\theta)dt\right)d\theta dt \\
&= \int_Y \left(p(t|\mathbf{x})\int_\Theta \frac{\beta(\theta)h(t|\mathbf{x};\theta)}{c(t|\mathbf{x};\beta)}d\theta - c(t|\mathbf{x};\beta)\right)dt \\
&= \int_Y p(t|\mathbf{x})dt - 1 = 0
\end{aligned}$$

The uniqueness of the price functional is not known under any assumption (such as smoothness). However, this market defines a transform between the ground truth and budget function. First observe that successive budget updates can be written as

$$\begin{aligned}
\beta^{i+1}(\theta) &= \beta^i(\theta)\int_Y p(t|\mathbf{x})\frac{h(t|\mathbf{x};\theta)}{c(t|\mathbf{x};\beta^i)}dt \\
&= \beta^{i-1}(\theta)\int_Y p(t|\mathbf{x})\frac{h(t|\mathbf{x};\theta)}{c(t|\mathbf{x};\beta^{i-1})}dt\int_Y p(t|\mathbf{x})\frac{h(t|\mathbf{x};\theta)}{c(t|\mathbf{x};\beta^i)}dt \\
&= \beta^0(\theta)\prod_{j=0}^{i}\int_Y p(t|\mathbf{x})\frac{h(t|\mathbf{x};\theta)}{c(t|\mathbf{x};\beta^j)}dt
\end{aligned}$$

where $\beta^0(\theta)$ is some chosen initial budget function. Assuming that there is a unique budget function $\beta^*(\theta)$ such that $p(t|\mathbf{x}) = c(t|\mathbf{x};\beta^*)$, then as $i \to \infty$ we would expect $\beta^i \to \beta^*$. Then this defines the transform as

$$\beta^*(\theta) = \beta^0(\theta)\prod_{i=0}^{\infty}\int_Y p(t|\mathbf{x})\frac{h(t|\mathbf{x};\theta)}{c(t|\mathbf{x};\beta^i)}dt \tag{7.1}$$

$$p(t|\mathbf{x}) = \int_\Theta \beta^*(\theta)h(t|\mathbf{x};\theta)d\theta \tag{7.2}$$

The optimal budget function $\beta^*(\theta)$ is expected to be a density with modes at the optimal constituent parameters. This might lead to an alternative method to EM to infer mixture model weights and parameters.

7.2 Clustering Market

If the Market Transform described in 7.1 really does work, then one possible application of this market is clustering. Writing (7.1) and (7.2) in terms of density aggregation gives

$$\beta^*(\theta) = \beta^0(\theta) \prod_{i=0}^{\infty} \int_X p(\mathbf{x}) \frac{h(\mathbf{x};\theta)}{c(\mathbf{x};\beta^i)} d\mathbf{x} \tag{7.3}$$

$$p(\mathbf{x}) = \int_\Theta \beta^*(\theta) h(\mathbf{x};\theta) d\theta \tag{7.4}$$

where the price functional is defined as

$$c(\mathbf{x};\beta) = \int_\Theta \beta(\theta) h(\mathbf{x};\theta) d\theta \tag{7.5}$$

It is suspected that the optimal $\beta^*(\theta)$ that gives $p(\mathbf{x}) = c(\mathbf{x};\beta^*)$ has modes around parameters θ that best describe the clusters. That is

$$\nabla_\theta \beta^*(\theta) = \mathbf{0} \tag{7.6}$$

For all solutions $\theta_m,\ m = 1, 2, \dots, M$ to (7.6), we suppose the ideal mixture model is then given as

$$c(\mathbf{x};\beta^*) = \frac{1}{n} \sum_{m=1}^{M} \beta^*(\theta_m) h(\mathbf{x};\theta_m)$$

where $n = \sum_{m=1}^{M} \beta(\theta_m)$ and θ_m are parameters that describe the clusters (e.g. like μ, Σ in EM clustering with Gaussians). This looks surprisingly like the density market equilibrium price function (5.7). However, it is not computationally feasible to directly compute (7.3). One possible solution is to simultaneously update and optimize the budget function. The incremental budget function update for densities is given as

$$\beta^{t+1}(\theta) = \beta^t(\theta) \int_X p(\mathbf{x}) \frac{h(\mathbf{x};\theta)}{c(\mathbf{x};\beta^t)} d\mathbf{x}$$

The gradient with respect to θ is then given as

$$\nabla_\theta \beta^{t+1}(\theta) = \nabla_\theta \beta^t(\theta) \int_X p(\mathbf{x}) \frac{h(\mathbf{x};\theta)}{c(\mathbf{x};\beta^t)} d\mathbf{x} + \beta^t(\theta) \int_X p(\mathbf{x}) \frac{\nabla_\theta h(\mathbf{x};\theta)}{c(\mathbf{x};\beta^t)} d\mathbf{x}$$

The objective then, is to find a set of distinct local optima $\theta^*_m,\ m = 1, 2, \dots, M$ for (7.2). This can be accomplished through, for example, gradient ascent

$$\theta_m^{t+1} = \theta_m^t + \epsilon \nabla_\theta \beta^{t+1}(\theta_m^t) \quad m = 1, 2, \dots, M$$

If ϵ is sufficiently small, we might avoid recomputing the gradient of the budget function on the new θ_m^{t+1} by noting that

$$\nabla_\theta \beta^{t+1}(\theta_m^{t+1}) \approx \nabla_\theta \beta^{t+1}(\theta_m^t)$$

which follows from the Taylor expansion of $\beta^{t+1}(\theta)$ about θ_m^t. This gives the following update scheme

$$\beta_m^{t+1} = \beta_m^t \int_X p(\mathbf{x}) \frac{h(\mathbf{x};\theta_m^t)}{c(\mathbf{x};\beta^t)} d\mathbf{x} \tag{7.7}$$

$$\nabla_\theta \beta_m^{t+1} = \nabla_\theta \beta_m^t \int_X p(\mathbf{x}) \frac{h(\mathbf{x};\theta_m^t)}{c(\mathbf{x};\beta_m^t)} d\mathbf{x} + \beta_m^t \int_X p(\mathbf{x}) \frac{\nabla_\theta h(\mathbf{x};\theta_m^t)}{c(\mathbf{x};\beta_m^t)} d\mathbf{x} \tag{7.8}$$

$$\theta_m^{t+1} = \theta_m^t + \epsilon \nabla_\theta \beta_m^{t+1} \tag{7.9}$$

where $\beta_m^t = \beta^t(\theta_m^t)$ and we may arbitrarily choose θ_m^0 and $\beta_m^0 = \frac{1}{M}$, $\nabla_\theta \beta_m^0 = \mathbf{0}$, $m = 1, 2, \ldots, M$. The price function $c(\mathbf{x};\beta^t)$ might be approximated with 5.7

$$c(\mathbf{x};\beta^t) = \sum_{m=1}^{M} \beta_m^t h(\mathbf{x};\theta_m^t)$$

The integrals in the above scheme can be estimated with Monte Carlo quadrature on the data points $\mathbf{x}_n$, giving

$$\beta_m^{t+1} = \beta_m^t \frac{1}{N} \sum_{n=1}^{N} \frac{h(\mathbf{x}_n;\theta_m^t)}{c(\mathbf{x}_n;\beta^t)} \tag{7.10}$$

$$\nabla_\theta \beta_m^{t+1} = \nabla_\theta \beta_m^t \frac{1}{N} \sum_{n=1}^{N} \frac{h(\mathbf{x}_n;\theta_m^t)}{c(\mathbf{x}_n;\beta_m^t)} + \beta_m^t \frac{1}{N} \sum_{n=1}^{N} \frac{\nabla_\theta h(\mathbf{x}_n;\theta_m^t)}{c(\mathbf{x}_n;\beta_m^t)} \tag{7.11}$$

$$\theta_m^{t+1} = \theta_m^t + \epsilon \nabla_\theta \beta_m^{t+1} \tag{7.12}$$

This idea is further justified from empirical observations from section 6.3. Even when budgets are randomly initialized, just one market update can make *dramatic* changes to the budgets. Each new θ_m^t can be thought of as resulting in a new market with new participants. Thus, the next budget update should *dramatically* change to reflect the best participants.

7.3 Object Detection

Recent works [27][45][18] have examined the regression forest for object detection with promising findings. Since the Classification and Regression Markets have been extensively applied to aggregating the leaves of decision trees, this section explores the possibilities and issues when aggregating regression forests for object detection.

7.3.1 Problem Setup

Given a set of images I_n, $n = 1, 2, \ldots, N$ of arbitrary dimensions (could be 2D or 3D) and a set of corresponding annotated positions $\mathbf{y}_n$, the objective is to learn a regressor $f(I(\mathbf{x}))$ that predicts an offset vector $\mathbf{v}$ such that $\mathbf{y} \approx \mathbf{x} + \mathbf{v}$. Since the images I_n can vary in dimension and field of view, the regressor cannot possibly predict the object position directly. The training instances for the regressor are pairs of patches and offsets $(I(\mathbf{x}), \mathbf{y} - \mathbf{x})$ where $\mathbf{y}$ is the annoted object position for image I. This gives the training set as

$$\{(I_n(\mathbf{x}), \mathbf{y}_n - \mathbf{x} \ : \ \forall \mathbf{x} \in I_n, \ n = 1, 2, \ldots, N\}$$

where $\mathbf{x} \in I_n$ denotes all pixel/voxel positions in image I_n.

7.3.2 Regression Forest for Object Detection

The regression forest for object detection is similar to the regression forest described in chapter 2 except that the prediction is multidimensional. While the training and evaluation of this type of regression tree is similar, the splitting criteria is slightly different than the conventional regression forest. The splitting criteria used in [45] amounts to component-wise variance reduction

$$\mathrm{Var}(V) = \mathrm{E}[V^T V] - \mathrm{E}[V]^T \mathrm{E}[V]$$

where V denotes a matrix with the offset vectors $\mathbf{v}_n(\mathbf{x}) = \mathbf{y}_n - \mathbf{x}$ as the rows. This gives the splitting criteria as

$$\ell(V; t, f) = \frac{|V_{I(\mathbf{x})_f \le t}|}{|V|} \mathrm{Var}(V_{I(\mathbf{x})_f \le t}) + \frac{|V_{I(\mathbf{x})_f > t}|}{|V|} \mathrm{Var}(V_{I(\mathbf{x})_f > t}) \tag{7.13}$$

In [18], the offsets were modeled as Gaussians and the regression tree optimized the information gain (2.4) with the differential entropy of the Gaussians as the purity function

$$H_{\text{differential entropy}}(\mu, \Sigma) = \frac{1}{2} \log \Big((2\pi e)^d |\Sigma| \Big)$$

where μ is the mean of the offset vectors and Σ is the covariance matrix.

7.3.3 Hough Forest

The Hough Forest [27] solves a slightly different problem. In addition to the offset vectors $\mathbf{v}_n(\mathbf{x})$, it also considers the foreground/background of an image patch. In addition to the setup described in section 7.3.1, the Hough Forest also introduces an annotated bounding box in each image I_n centered around the object position $\mathbf{y}_n$. Thus, a training instance is a tuple $(I(\mathbf{x}), \mathbf{y} - \mathbf{x}, z)$ where z is the foreground/background label. A training instance is labeled foreground if $\mathbf{x}$ falls inside the bounding box and background otherwise. For any background position $\mathbf{x}$, the offset vector $\mathbf{y} - \mathbf{x}$ is ommitted from the training instance. In this sense, a Hough Forest is both a regressor and a classifier.

Hough trees employ randomly alternating split criteria to solve both the regression and classification problem. It either uses (7.13) for the regression task, or for the classification task (2.4) with the entropy purity function H_{entropy}.

When evaluated, a Hough Forest produces a probability score that the patch $I(\mathbf{x})$ is foreground and if any foreground example fell in one of the leaves, it predicts an average offset $\mathbf{v}$ to the object position. Thus, the Hough Forest is evaluated at every position in an image I and places a weighted vote at the predicted position $\mathbf{x} + \mathbf{v}$ with the weight as the foreground probability score. This gives a voting map which requires some post processing, such as non-maximal suppression or mean-shift, to extract the final predicted location due to some possible outlier predictions. See figure 7.1 for an example of the offset predictions and voting map. To deal with scale and aspect ratio variation, the Hough Forest is evaluated on a pyramid of images. For each scaled image, the Hough Forest predicts an aspect ratio adjusted offset, or

$$\mathbf{y} = \mathbf{x} + r\mathbf{v}$$

where r is the aspect ratio scalar. On the Weizmann horse data set [8], the scales varied from $s \in \{0.7,\ 0.6,\ 0.5,\ 0.4,\ 0.3\}$ and the aspect ratios varied from $r \in \{0.5,\ 0.75,\ 1.0,\ 1.25,\ 1.5\}$.

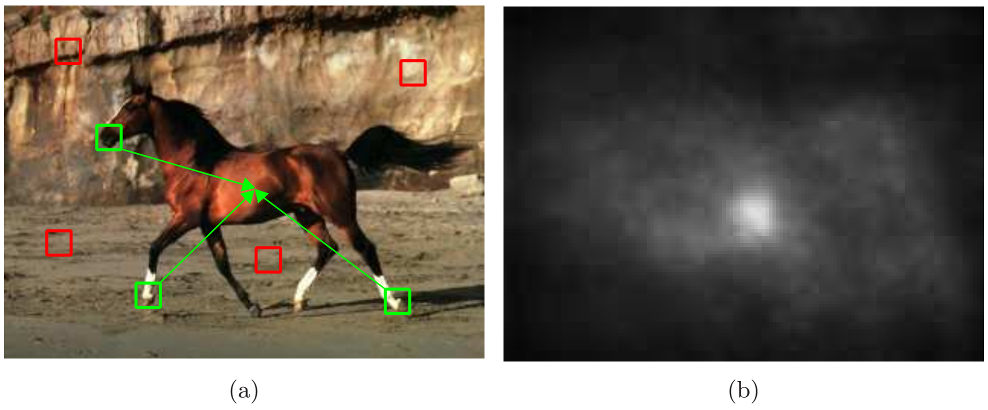


Figure 7.1: Example of Hough Forest evaluation. Figure (a) illustrates how Hough Forest predicts foreground (green) and background (red). The foreground patches predict offsets while the background patches do not. Figure (b) shows the resulting voting map on the image. The horse center prediction is well localized, although with some noisy predictions far away.

7.3.4 Hough Market

Briefly explored, the Hough Market is a type of density market that aggregates the leaves of Hough trees in Hough Forest. The objective is to minimize the KL divergence between the resulting voting map and a Gaussian centered about the the object center y.

$$\mathrm{KL}(N(\mathbf{x},\mathbf{y},\sigma), c(\mathbf{x}|I)) = \sum_{\mathbf{x}\in I} N(\mathbf{x},\mathbf{y},\sigma)\log\left(\frac{N(\mathbf{x},\mathbf{y},\sigma)}{c(\mathbf{x}|I)}\right)$$

where $N(\mathbf{x},\mathbf{y},\sigma)$ is the Gaussian density with mean $\mathbf{y}$ and standard deviation σ and $c(\mathbf{x}|I)$ is the voting map. The Hough Market attempts to exploit specialization and updates the estimated probability scores stored in the leaves to weed out poorly performing Hough tree leaves. See example 7.2 for an illustration of the process.

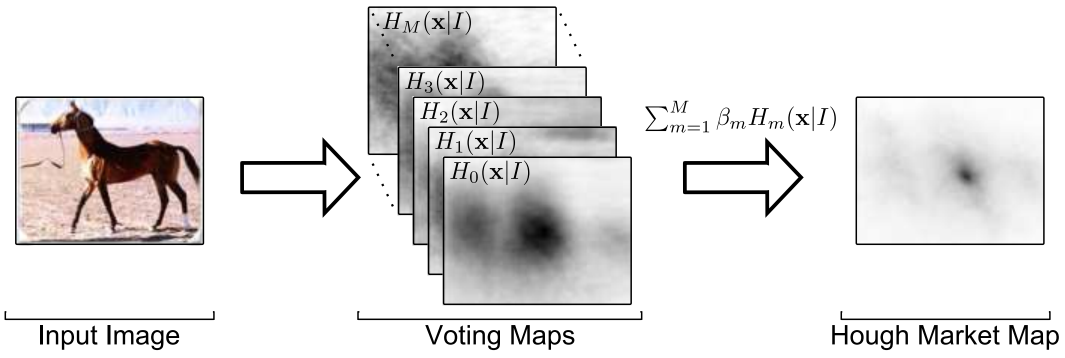


Figure 7.2: Example of aggregation of Hough tree leaves on a horse image.

Training. The training in a Hough Market proceeds by first recording a voting map for each individual leaf H_m. If a leaf is a foreground leaf, then the voting map for the leaf is computed as described by section 7.3.3. If the leaf is negative, then the voting map is computed by adding $\frac{1}{|I_m|}$, where $|I_m| = \text{width} \times \text{height}$, at each pixel position to indicate uncertainty at every position. Then the equilibrium price is computed as

$$c(\mathbf{x}|I) = \frac{1}{Z} \sum_{m=1}^{M} \beta_m H_m(\mathbf{x}|I) \tag{7.14}$$

where $Z = \sum_{\mathbf{x} \in I} c(\mathbf{x}|I)$ to normalize it to sum to 1. Since the *ground truth* density $N(\mathbf{x}, \mathbf{y}, \sigma)$ is known, we may directly approximate 5.6 with a Riemann sum

$$\beta_m \leftarrow \beta_m + \eta \beta_m \sum_{\mathbf{x} \in I} \left[N(\mathbf{x}, \mathbf{y}, \sigma) \frac{H_m(\mathbf{x}|I)/Z}{c(\mathbf{x}|I; \boldsymbol{\beta})} - \frac{H_m(\mathbf{x}|I)}{Z} \right] \tag{7.15}$$

Here the normalization factor Z is needed on the betting functions since they do not necessarily sum to ≤ 1. The η here is used to weight the update on positives and negatives and taken to be

$$\eta = \begin{cases} 0.05\eta_{\max} & \text{positive} \\ 0.5\eta_{\max} & \text{negative} \end{cases} \tag{7.16}$$

where $\eta_{\max}$ describes the maximum value of η so that at least one participant goes bankrupt (i.e. $\beta_m = 0$).

A market was trained on the Weizmann horse data set as described above for each individual scale-ratio pair (s, r), giving a total of 25 markets.

Evaluation. During evaluation, the equilibrium price is instead computed without normalization since the normalized values are miniscule, thus

$$c(\mathbf{x}|I) = \sum_{m=1}^{M} \beta_m H_m(\mathbf{x}|I) \tag{7.17}$$

This also allows a fair comparison with Hough Forest since an untrained market will give identical voting maps as the Hough Forest. Once the voting map is computed for both Hough Forest and Hough Market, non-maximal suppression is used to post process the voting maps giving a few detections per image. A template box is then positioned and scaled based on the corresponding scale-ratio pair (s, r) to give the final bounding box detection.

The threshold for the non-maximal suppression is determined by the ROC curve (at 10% false alarm rate). The ROC curve is based on thresholds on the voting map weights with true and false positive defined as

$$\text{true positive} = \frac{|T \cap G|}{|T \cup G|} > 0.5$$

$$\text{false positive} = \frac{|T \cap G|}{|T \cup G|} \leq 0.5$$

where T is the template box and G is the ground truth annotation. Negative images are given 1 true negative if there are no detections and positive images are given 1 false negative if there are no detections. For the Hough Market, the AUC is computed at each epoch and then the epoch with the largest AUC is used for detection with the threshold picked the same way. Some example detections for Hough Forest and Hough Market can be seen in figures 7.4 and 7.5 respectively. The thresholds for the examples were computed on the unseen ROC curves given in figure 7.3.

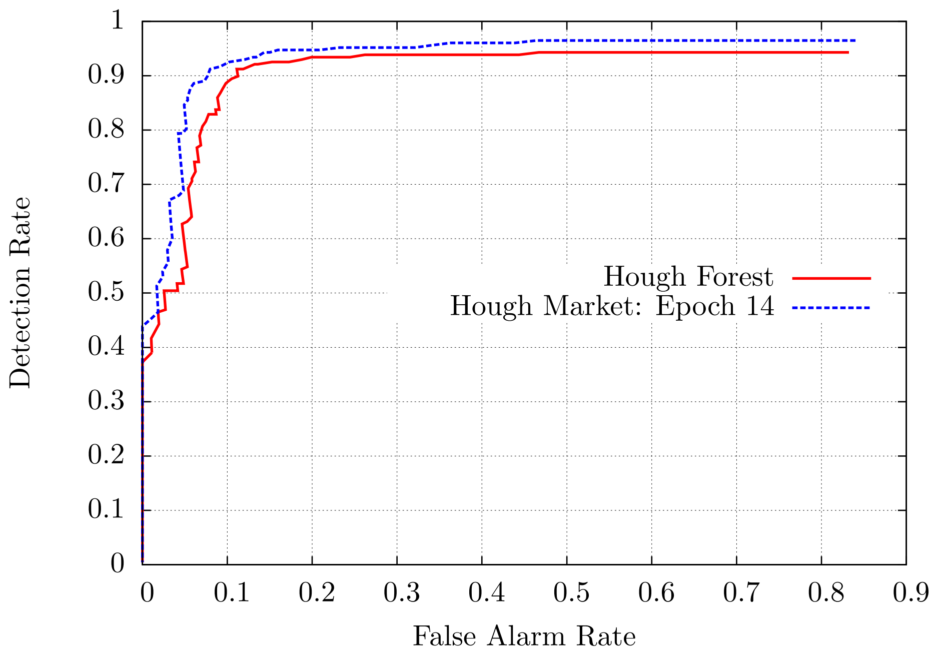


Figure 7.3: ROC curves for horse detection on the Weizmann test set.

Issues. While the Hough Market does reduce the average KL divergence on the training images, this does not necessarily imply an improvement in the ROC curve. Qualitatively, compared to the Hough Forest, the Hough Market would produce better localized peaks in the voting maps at the cost of more far away noisy predictions. The Hough Market was observed to increase the area under the unseen ROC curve (AUC), but the AUC for the training ROC curve would strangely decrease which is concerning.

7.4 Betting Function Learning

At the time of this writing, one limitation of the Artificial Prediction Market has been the inability to infer the participants or their parameters. Other aggregation methods such as boosting [52] train their own constituent classifiers. The Classification and Regression Markets have largely relied on decision tree leaves,as trained by decision trees, for the specialized classifier. Indeed, the original objective of Artificial Prediction Markets has

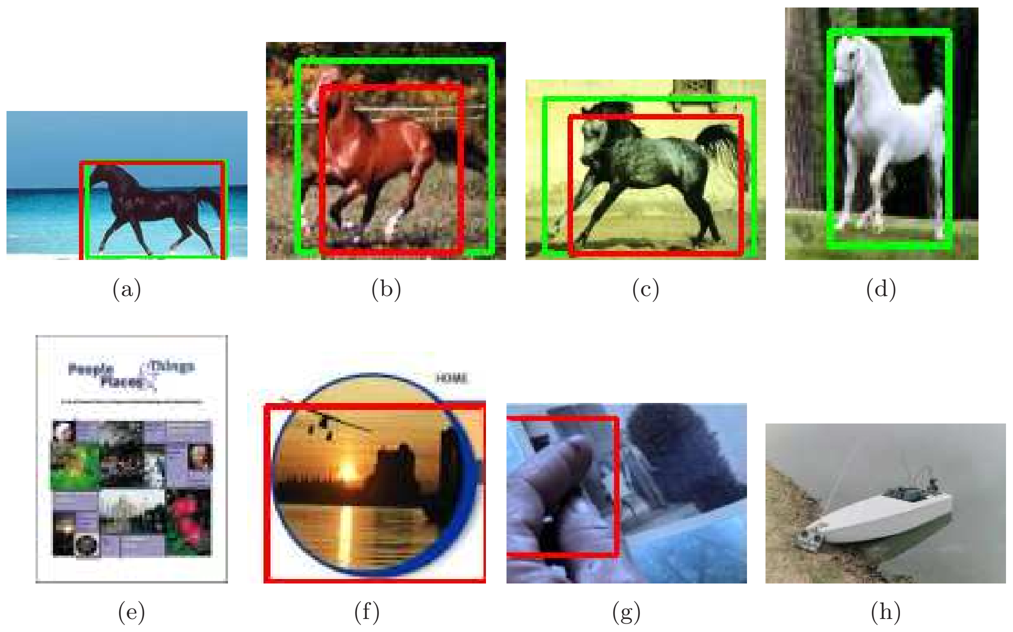


Figure 7.4: Example detections of the Hough Forest. The green box is the ground truth while the red box is the detection. The first row, or (a)(b)(c)(d), are detections on positive images while the second row, or (e)(f)(g)(h), are the detections on negative images.

always been purely aggregation of generic models. This section briefly explores inferring betting functions.

7.4.1 Market Prices and Auto Context

The trading prices of real prediction markets provide some insight on the outcome of an event. If the efficient market hypothesis, in any of its forms, is true, then the trading price reflects the fusion of some or all available information, public or private. There have been works that examined the issue of interpreting the trading prices of prediction markets. In this sense, contract trading prices could be useful *features* in learning problems.

Using probabilities as features has been explored in [51] with promising results in computer vision tasks. In [51], a classifier is first trained on image features to produce a map of the classification probabilities, then another classifier is trained using either image features or the classification map as features. This process can be repeated several times. While a single instance in an Artificial Prediction Market only produces a single equilibrium price instead of a map of prices, this bares some resemblance to AutoContext since betting functions are functions of the price. In that sense, the prices are features for betting functions.

7.4.2 Online Random Trees

One potential online learning method for betting function learning is the Online Random Tree [22]. This would also serve as an ideal method for comparison with the results

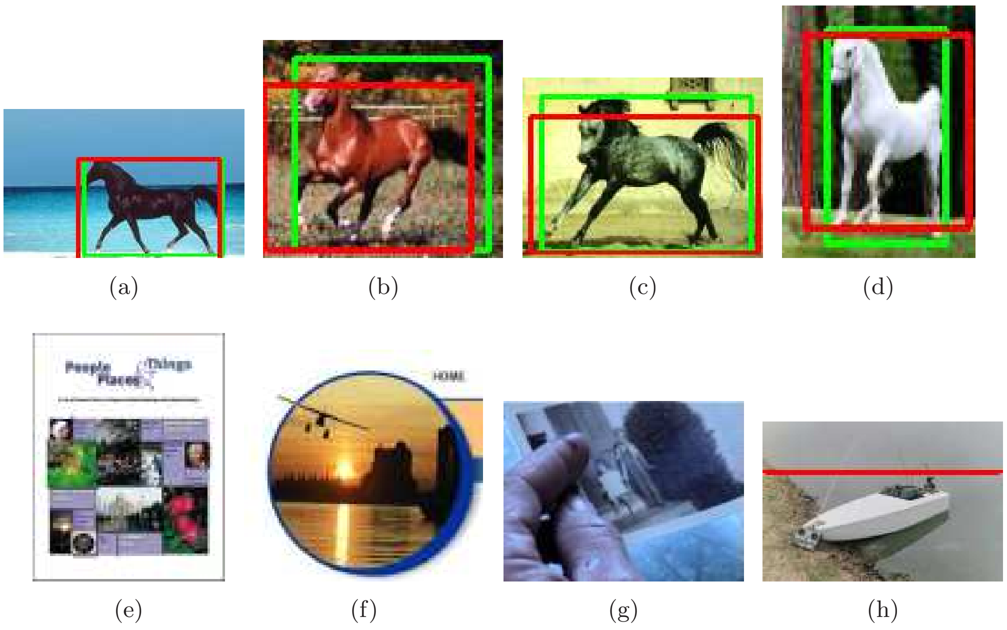


Figure 7.5: Example detections of the Hough Market. The green box is the ground truth while the red box is the detection. The first row, or (a)(b)(c)(d), are detections on positive images while the second row, or (e)(f)(g)(h), are the detections on negative images. While the Hough Market can eliminate some of the false positives and false negatives, it can also introduce them as in (h).

reported in this work and in [34][33][3] as these consider the *batch* random forest. Unlike the training algorithm described in section 2.2, the Online Random Tree described in [22] incrementally considers the correlation between features either in a forward fashion (*CorrFS*) or a backward fashion (*CorrBE*).

7.4.3 AutoMarket

With a suitable online learning method such as Online Random Tree, the market and its participants can be trained incrementally. This involves first estimating the equilibrium prices $c_k,\ k = 1, 2, \ldots, K$, computing the budget update, and then updating the participants with the instance features augmented with the price.

For a given training instance $(\mathbf{x}, y)$, the equilibrium price is first computed by solving the fixed point equation

$$c_k = \frac{1}{n}\sum_{m=1}^{M} \beta_m \phi_m^k(\mathbf{x}, \mathbf{c}) \quad k = 1, 2, \ldots, K \tag{7.18}$$

where $n = \sum_{m=1}^{M} \beta_m \sum_{k=1}^{K} \phi_m(\mathbf{x}, \mathbf{c})$ is a normalizer. The solution must be approximated with something like the Mann Iteration [37] as in algorithm 4. Then the budgets are updated

$$\beta_m \leftarrow \beta_m - \eta\beta_m \sum_{k=1}^{K} \phi_m^k(\mathbf{x}, \mathbf{c}) + \eta\beta_m \frac{\phi_m^y(\mathbf{x}, \mathbf{c})}{c_y} \tag{7.19}$$

where $\eta \leq 1$ is intended to prevent instant bankruptcies (i.e. $\beta_m = 0$). And lastly the feature vector $\mathbf{x}$ is augmented with the prices $[\mathbf{x}|\mathbf{c}]$ and the training instance $([\mathbf{x}|\mathbf{c}], y)$ is used to update each participant $\phi_m,\ m = 1, 2, \ldots, M$.

One special consideration is that the online method give equal weight to the features $\mathbf{x}$ and $\mathbf{c}$ (if this applies). This is to ensure that features from either $\mathbf{x}$ or $\mathbf{c}$ are selected fairly as the dimension of $\mathbf{x}$ is likely to exceed K, the number of class labels and the dimension of $\mathbf{c}$.

CHAPTER 8

CONCLUSION

This work presents a theory for artificial prediction markets for the purpose of supervised learning of class conditional probability estimators, real value conditional probability estimators for regression, and density estimators. The artificial prediction market is a novel online learning algorithm that can be easily implemented classification, regression and density estimation applications. Linear aggregation, logistic regression as well as certain kernel methods can be viewed as particular instances of the artificial prediction markets. Inspired from real life, specialized classifiers that only bet on subsets of the instance space were introduced. Experimental comparisons on real and synthetic data show that the prediction market usually outperforms random forest, regression forest, adaboost and implicit online learning in prediction accuracy.

The artificial prediction market shows the following promising features:

1. It can be updated online with minimal computational cost when a new observation $(\mathbf{x}, y)$ is presented (or just $\mathbf{x}$ in the case of density estimation).

2. It has a simple form of the update iteration that can be easily implemented.

3. For multi-class classification it can fuse information from all types of binary or multi-class classifiers: e.g. trained one-vs-all, many-vs-many, multi-class decision tree, etc.

4. It can obtain meaningful probability estimates when only a subset of the market participants are involved for a particular instance $\mathbf{x} \in X$. This feature is useful for learning on manifolds [6, 21, 46] , where the location on the manifold decides which market participants should be involved. For example, in face detection, different face part classifiers (eyes, mouth, ears, nose, hair, etc) can be involved in the market, depending on the orientation of the head hypothesis being evaluated.

5. Because of their betting functions, the specialized market participants can decide for which instances they bet and how much. This is another way to combine classifiers, regressors, and density estimators different from other approaches such as boosting, kernel methods, EM, where all classifiers, regressors, and density estimators participate in estimating the conditional probability for each observation.

6. Because of the general nature of the framework, the market can potentially aggregate heterogenous models. Other aggregation approaches such as boosting, kernel methods,

and EM work with specific forms of models at a time.

Future work includes finding explicit bounds for the generalization error based on the number of training examples. Another item of future work is finding other generic types specialized participants that are not leaves of random or adaboost trees. For example, by clustering the instances $\mathbf{x} \in \Omega$, one could find regions of the instance space Ω where simple models (e.g. logistic regression, linear regression, Gaussians, etc) can be used as specialized market participants for that region. In the case of regression, hinge functions from *Multivariate Adaptive Regression Splines* can be used as specialized participants in place of regression tree leaves. Lastly, we have briefly explored betting function learning in the online random tree framework. This especially deserves more attention since even the participants are online learning methods. This can potentially be extended to MARS as well in the regression task.

APPENDIX A

PROOFS

Proof of Theorem 3.1.1. From eq. (3.12), the total budget $\sum_{m=1}^{M} \beta_m$ is conserved if and only if

$$\sum_{m=1}^{M} \sum_{k=1}^{K} \beta_m \phi_m^k(\mathbf{x}, \mathbf{c}) = \sum_{m=1}^{M} \beta_m \phi_m^y(\mathbf{x}, \mathbf{c})/c_y \tag{A.1}$$

Denoting $n = \sum_{m=1}^{M} \sum_{k=1}^{K} \beta_m \phi_m^k(\mathbf{x}, \mathbf{c})$, and since the above equation must hold for all y, we obtain that eq. (3.18) is a necessary condition and also $c_k \neq 0, k = 1, ..., K$, which means $c_k > 0, k = 1, ..., K$. Reciprocally, if $c_k > 0$ and eq. (3.18) hold for all k, dividing by c_k we obtain eq. (A.1).

□

Proof of Remark 3.1.2. Since the total budget is conserved and is positive, there exists a $\beta_m > 0$, therefore $\sum_{m=1}^{M} \beta_m \phi_m^k(\mathbf{x}, 0) > 0$, which implies $\lim_{c_k \to 0} f_k(c_k) = \infty$. From the fact that $f_k(c_k)$ is continuous and strictly decreasing, with $\lim_{c_k \to 0} f_k(c_k) = \infty$ and $\lim_{c_k \to 1} f_k(c_k) = 0$, it implies that for every $n > 0$ there exists a unique c_k that satisfies $f_k(c_k) = n$. □

Proof of Theorem 3.1.3. From Remark 3.1.2 we get that for every $n \geq n_k, n > 0$ there is a unique $c_k(n)$ such that $f_k(c_k(n)) = n$. Moreover, following the proof of Remark 3.1.2 we see that $c_k(n)$ is continuous and strictly decreasing on (n_k, ∞), with $\lim_{n \to \infty} c_k(n) = 0$. If $\max_k n_k > 0$, take $n^* = \max_k n_k$. There exists $k \in \{1, ..., K\}$ such that $n_k = n^*$, so $c_k(n^*) = 1$, therefore $\sum_{j=1}^{K} c_j(n^*) \geq 1$.

If $\max_k n_k = 0$ then $n_k = 0, k = 1, ..., K$ which means $\phi_m^k(\mathbf{x}, 1) = 0, k = 1, ..., K$ for all m with $\beta_m > 0$. Let $a_m^k = \min\{c | \phi_m^k(\mathbf{x}, c) = 0\}$. We have $a_m^k > 0$ for all k since $\phi_m^k(\mathbf{x}, 0) > 0$. Thus $\lim_{n \to 0_+} c_k(n) = \max_m a_m^k \geq a_1^k$, where we assumed that $\phi_1(\mathbf{x}, \mathbf{c})$ satisfies Assumption 1. But from Assumption 1 there exists k such that $a_1^k = 1$. Thus $\lim_{n \to 0_+} \sum_{k=1}^{K} c_k(n) \geq \sum_{k=1}^{K} a_1^k > 1$ so there exists n^* such that $\sum_{k=1}^{K} c_k(n^*) \geq 1$.

Either way, since $\sum_{k=1}^{K} c_k(n)$ is continuous, strictly decreasing, and since $\sum_{k=1}^{K} c_k(n^*) \geq 1$ and $\lim_{n \to \infty} \sum_{k=1}^{K} c_k(n) = 0$, there exists a unique $n > 0$ such that $\sum_{k=1}^{K} c_k(n) = 1$. For this n, from Theorem 3.1.1 follows that the total budget is conserved for the price $\mathbf{c} = (c_1(n), ..., c_K(n))$. Uniqueness follows from the uniqueness of $c_k(n)$ and the uniqueness of n. □

Proof of Theorem 3.4.1. For the current parameters $\gamma = (\gamma_1, ..., \gamma_M) = (\sqrt{\beta_1}, ..., \sqrt{\beta_m})$ and an observation $(\mathbf{x}_i, y_i)$, we have the market price for label y_i:

$$c_{y_i}(\mathbf{x}_i) = \sum_{m=1}^{M} \gamma_m^2 \phi_m^{y_i}(\mathbf{x}_i) / (\sum_{m=1}^{M} \sum_{k=1}^{K} \gamma_m^2 \phi_m^k(\mathbf{x}_i)) \tag{A.2}$$

So the log-likelihood is

$$L(\gamma) = \frac{1}{N} \sum_{i=1}^{N} \log c_{y_i}(\mathbf{x}_i) = \frac{1}{N} \sum_{i=1}^{N} \log \sum_{m=1}^{M} \gamma_m^2 \phi_m^{y_i}(\mathbf{x}_i) - \frac{1}{N} \sum_{i=1}^{N} \log \sum_{m=1}^{M} \sum_{k=1}^{K} \gamma_m^2 \phi_m^k(\mathbf{x}_i) \tag{A.3}$$

We obtain the gradient components:

$$\frac{\partial L(\gamma)}{\partial \gamma_j} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\gamma_j \phi_j^{y_i}(\mathbf{x}_i)}{\sum_{m=1}^{M} \gamma_m^2 \phi_m^{y_i}(\mathbf{x}_i)} - \frac{\gamma_j \sum_{k=1}^{K} \phi_j^k(\mathbf{x}_i)}{\sum_{m=1}^{M} \sum_{k=1}^{K} \gamma_m^2 \phi_m^k(\mathbf{x}_i)} \right) \tag{A.4}$$

Then from (A.2) we have $\sum_{m=1}^{M} \gamma_m^2 \phi_m^{y_i}(\mathbf{x}_i) = B(\mathbf{x}_i) c_{y_i}(\mathbf{x}_i)$. Hence (A.4) becomes

$$\frac{\partial L(\gamma)}{\partial \gamma_j} = \frac{\gamma_j}{N} \sum_{i=1}^{N} \frac{1}{B(\mathbf{x}_i)} \left(\frac{\phi_j^{y_i}(\mathbf{x}_i)}{c_{y_i}(\mathbf{x}_i)} - \sum_{k=1}^{K} \phi_j^k(\mathbf{x}_i) \right).$$

Write $u_j = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{B(\mathbf{x}_i)} \left(\frac{\phi_j^{y_i}(\mathbf{x}_i)}{c_{y_i}(\mathbf{x}_i)} - \sum_{k=1}^{K} \phi_j^k(\mathbf{x}_i) \right)$, then $\frac{\partial L(\gamma)}{\partial \gamma_j} = \gamma_j u_j$. The batch update (3.23) is $\beta_j \leftarrow \beta_j + \eta \beta_j u_j$. By taking the square root we get the update in γ

$$\gamma_j \leftarrow \gamma_j \sqrt{1 + \eta u_j} = \gamma_j + \gamma_j(\sqrt{1 + \eta u_j} - 1) = \gamma_j + \gamma_j \frac{\eta u_j}{\sqrt{1 + \eta u_j} + 1} = \gamma_j'.$$

We can write the Taylor expansion:

$$L(\gamma') = L(\gamma) + (\gamma' - \gamma)^T \nabla L(\gamma) + \frac{1}{2} (\gamma' - \gamma)^T H(L)(\zeta)(\gamma' - \gamma)$$

so

$$L(\gamma') = L(\gamma) + \sum_{j=1}^{M} \gamma_j u_j \frac{\eta \gamma_j u_j}{\sqrt{1 + \eta u_j} + 1} + \eta^2 A(\eta) = L(\gamma) + \eta \sum_{j=1}^{M} \frac{\gamma_j^2 u_j^2}{\sqrt{1 + \eta u_j} + 1} + \eta^2 A(\eta)$$

where $|A(\eta)|$ is bounded in a neighborhood of 0.

Now assume that $\nabla L(\gamma) \neq 0$, thus $\gamma_j u_j \neq 0$ for some j. Then $\sum_{j=1}^{M} \frac{\gamma_j^2 u_j^2}{\sqrt{1+\eta u_j}+1} > 0$ hence $L(\gamma') > L(\gamma)$ for any η small enough.

Thus as long as $\nabla L(\gamma) \neq 0$ the batch update (3.23) with any η sufficiently small will increase the likelihood function.

The batch update (3.23) can be split into N per-observation updates of the form (3.24). □

Proof of Theorem 3.4.3. The Hessian matrix $\ell(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta} \in \mathbb{R}_{\geq 0}^M \backslash \{\mathbf{0}\}$ is defined as

$$H = \int_{\Omega} p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{\mathbf{h}^k(\mathbf{x}) \mathbf{h}^k(\mathbf{x})^T}{c_k(\mathbf{x}; \boldsymbol{\beta})^2} d\mathbf{x}$$

Suppose $\exists \mathbf{v} \in \mathbb{R}^M,\ \mathbf{v} \neq \mathbf{0}$ such that $\mathbf{v}^T H \mathbf{v} = 0$, then

$$\int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{\mathbf{v}^T \mathbf{h}^k(\mathbf{x}) \mathbf{h}^k(\mathbf{x})^T \mathbf{v}}{c_k(\mathbf{x};\boldsymbol{\beta})^2} d\mathbf{x} = 0$$

But since H is symmetric, it is *at least* semi-positive definite and so this will only integrate to 0 if $\mathbf{h}^k(\mathbf{x})^T \mathbf{v} = 0,\ k = 1, 2, \ldots, K$ almost everywhere. However, since the market participants $\mathbf{h}(\mathbf{x})$ is a vector of classifiers with linear dependence only on a zero measure set, then $\mathbf{h}^k(\mathbf{x})^T \mathbf{v} = 0 \implies \mathbf{v} = \mathbf{0}$. This is a contradiction and H is strictly positive definite. □

Proof of Theorem 3.4.4. Differentiating $g_m(\boldsymbol{\beta})$ gives

$$\frac{\partial g_m}{\partial \beta_n} = \beta_m \frac{\partial f_m}{\partial \beta_n} \quad m \neq n$$
$$\frac{\partial g_m}{\partial \beta_m} = f_m + \beta_m \frac{\partial f_m}{\partial \beta_m}$$

where the derivatives of f_m are given as

$$\frac{\partial f_m}{\partial \beta_n} = -\int_\Omega p(\mathbf{x}) \sum_{k=1}^{K} p(k|\mathbf{x}) \frac{h_m^k(\mathbf{x}) h_n^k(\mathbf{x})}{c_k(\mathbf{x};\boldsymbol{\beta})} d\mathbf{x}$$

This gives $J_\mathbf{g}$ in the convenient form

$$J_\mathbf{g} = \text{diag}(\mathbf{f}) + \text{diag}(\boldsymbol{\beta}) J_\mathbf{f}$$

Evaluated at $\boldsymbol{\beta}^*$ gives

$$J_\mathbf{g}(\boldsymbol{\beta}^*) = I + \text{diag}(\boldsymbol{\beta}^*) J_\mathbf{f}(\boldsymbol{\beta}^*)$$

since $f(\boldsymbol{\beta}^*) = \mathbf{1}$ from (3.30).

To show $J_\mathbf{g}(\boldsymbol{\beta}^*)$ has eigenvalues with magnitude less than 1, we first show that all eigenvalues $\lambda_{\boldsymbol{\beta}\mathbf{f}}$ of $\text{diag}(\boldsymbol{\beta}^*) J_\mathbf{f}(\boldsymbol{\beta}^*)$ are bounded $-1 \leq \lambda_{\boldsymbol{\beta}\mathbf{f}} < 0$.

First note that $\mathbf{f}(\boldsymbol{\beta}) = -\nabla_{\boldsymbol{\beta}} KL(p(\mathbf{x}), c(\mathbf{x}, \boldsymbol{\beta}))$ and so $J_\mathbf{f} = -H_{KL}$ where H_{KL} is the Hessian matrix of (3.28) and since $KL(p(\mathbf{x}), c(\mathbf{x}, \boldsymbol{\beta}))$ is strictly convex in $\boldsymbol{\beta}$ then $-H_{KL}$ is negative definite and thus has strictly negative eigenvalues. Now since the eigenvalues of $\text{diag}(\boldsymbol{\beta}^*) J_\mathbf{f}(\boldsymbol{\beta}^*)$ are equivalent to $\text{diag}(\boldsymbol{\beta}^*)^{\frac{1}{2}} J_\mathbf{f}(\boldsymbol{\beta}^*) \text{diag}(\boldsymbol{\beta}^*)^{\frac{1}{2}}$ which is a symmetric matrix, it's obvious that $\text{diag}(\boldsymbol{\beta}^*) J_\mathbf{f}(\boldsymbol{\beta}^*)$ remains negative definite and thus has strictly real negative eigenvalues. Therefore $\lambda_{\boldsymbol{\beta}\mathbf{f}} < 0$.

Now denote the matrix $A = \text{diag}(\boldsymbol{\beta}^*) J_\mathbf{f}(\boldsymbol{\beta}^*)$ given as

$$A_{mn} = \beta_m^* \frac{\partial f_m}{\partial \beta_n} = \beta_m^* \frac{\partial f_n}{\partial \beta_m}$$

Then by Gershgorin's circle theorem, the eigenvalues $\lambda_{\boldsymbol{\beta}\mathbf{f}}$ lie in the union of complex disks each defined as

$$|\lambda_{\boldsymbol{\beta}\mathbf{f}} - A_{nn}| \leq \sum_{m \neq n} |A_{mn}| \quad n = 1, 2, \ldots, M$$

However since the eigenvalues are real negative values, we may instead write

$$\begin{aligned}
-\sum_{m\neq n}|A_{mn}| &\leq \lambda_{\boldsymbol{\beta}\mathbf{f}} + A_{nn} \leq \sum_{m\neq n}|A_{mn}| \\
\sum_{m\neq n}\beta_m^*\frac{\partial f_n}{\partial \beta_m} &\leq \lambda_{\boldsymbol{\beta}\mathbf{f}} - \beta_n^*\frac{\partial f_n}{\partial \beta_n} < -\sum_{m\neq n}\beta_m^*\frac{\partial f_n}{\partial \beta_m} \\
\sum_{m=1}^{M}\beta_m^*\frac{\partial f_n}{\partial \beta_m} &\leq \lambda_{\boldsymbol{\beta}\mathbf{f}} < 0
\end{aligned}$$

Expanding the lower bound gives

$$\begin{aligned}
\sum_{m=1}^{M}\beta_m^*\frac{\partial f_n}{\beta_m} &= -\sum_{m=1}^{M}\beta_m^*\int_\Omega p(\mathbf{x})\sum_{k=1}^{K}p(k|\mathbf{x})\frac{h_m^k(\mathbf{x})h_n^k(\mathbf{x})}{c_k(\mathbf{x};\boldsymbol{\beta}^*)^2}d\mathbf{x} \\
&= -\int_\Omega p(\mathbf{x})\sum_{k=1}^{K}p(k|\mathbf{x})\frac{h_n^k(\mathbf{x})}{c_k(\mathbf{x};\boldsymbol{\beta}^*)}\sum_{m=1}^{M}\frac{\beta_m^* h_m^k(\mathbf{x})}{c_k(\mathbf{x};\boldsymbol{\beta}^*)}d\mathbf{x} \\
&= -\int_\Omega p(\mathbf{x})\sum_{k=1}^{K}p(k|\mathbf{x})\frac{h_n^k(\mathbf{x})}{c_k(\mathbf{x};\boldsymbol{\beta}^*)}d\mathbf{x} = -f_n(\boldsymbol{\beta}^*) = -1
\end{aligned}$$

Therefore

$$-1 \leq \lambda_{\boldsymbol{\beta}\mathbf{f}} < 0 \qquad \Longrightarrow J_{\mathbf{g}}(\boldsymbol{\beta}^*) = I + Q\Lambda_{\boldsymbol{\beta}\mathbf{f}}Q^{-1} = Q(I+\Lambda_{\boldsymbol{\beta}\mathbf{f}})Q^{-1} = Q\Lambda_{\mathbf{g}}Q^{-1}$$

giving the bounds for $\lambda_{\mathbf{g}}$

$$0 \leq \lambda_{\mathbf{g}} = 1 + \lambda_{\boldsymbol{\beta}\mathbf{f}} < 1$$

Therefore $\boldsymbol{\beta}^*$ is a sink that solves $\mathbf{f}(\boldsymbol{\beta}) = \mathbf{1} \implies \nabla_{\mathbf{v}} KL(p(\mathbf{x}), c(\mathbf{x};\boldsymbol{\beta})) = \mathbf{0}$. $\square$

BIBLIOGRAPHY

[1] Storkey Amos. Machine learning markets. *Journal of Machine Learning Research*, 2011.

[2] K. J. Arrow, R. Forsythe, M. Gorham, R. Hahn, R. Hanson, J. O. Ledyard, S. Levmore, R. Litan, P. Milgrom, and F. D. Nelson. The promise of prediction markets. *Science*, 320(5878):877, 2008.

[3] A. Barbu and N. Lay. An introduction to artificial prediction markets for classification. *Journal of Machine Learning Research*, 13:2177–2204, 2012.

[4] A. Barbu, M. Suehling, X. Xu, D. Liu, S. Zhou, and D. Comaniciu. Automatic detection and segmentation of lymph nodes from ct data. *IEEE Trans. on Medical Imaging*, 31(2):240–250, 2012.

[5] S. Basu. Investment performance of common stocks in relation to their price-earnings ratios: A test of the efficient market hypothesis. *The Journal of Finance*, 32(3):663–682, 1977.

[6] M. Belkin and P. Niyogi. Semi-supervised learning on Riemannian manifolds. *Machine Learning*, 56(1):209–239, 2004.

[7] C. Blake and CJ Merz. UCI repository of machine learning databases [http://www. ics. uci. edu/ mlearn/MLRepository. html], Department of Information and Computer Science. *University of California, Irvine, CA*, 1998.

[8] Eran Borenstein, Eitan Sharon, and Shimon Ullman. Combining top-down and bottom-up segmentation. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*, pages 46–46. IEEE, 2004.

[9] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[10] L. Breiman, J. H. Friedman, R. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, California, 1984.

[11] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[12] F. Bunea and A. Nobel. Sequential procedures for aggregating arbitrary estimators of a conditional mean. *IEEE Transactions on Information Theory*, 54(4):1725–1734, 2008.

[13] Y. Chen, J. Abernethy, and J.W. Vaughan. An optimization-based framework for automated market-making. *Proceedings of the EC*, 11:5–9, 2011.

[14] Y. Chen and J.W. Vaughan. A new understanding of prediction markets via no-regret learning. In *Proceedings of the 11th ACM conference on Electronic commerce*, pages 189–198. ACM, 2010.

[15] Yileng Chen and Jennifer Wortman Vaughan. A new understanding of prediction markets via no-regret learning. In *In the Eleventh ACM Conference on Electronic Commerce (EC 2010)*, 2010.

[16] C. Chow. On optimum recognition error and reject tradeoff. *IEEE Trans. on Information Theory*, 16(1):41–46, 1970.

[17] B. Cowgill, J. Wolfers, and E. Zitzewitz. Using prediction markets to track information flows: Evidence from Google. *Dartmouth College*, 2008.

[18] Antonio Criminisi, Jamie Shotton, Duncan Robertson, Konukoglu, and Ender. Regression forests for efficient anatomy detection and localization in ct studies. In Bjoern Menze, Georg Langs, Zhuowen Tu, and Antonio Criminisi, editors, *Medical Computer Vision. Recognition Techniques and Applications in Medical Imaging*, volume 6533 of *Lecture Notes in Computer Science*, pages 106–117. Springer Berlin / Heidelberg, 2011.

[19] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[20] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:30, 2006.

[21] A. Elgammal and C.S. Lee. Inferring 3d body pose from silhouettes using activity manifold learning. In *CVPR*, 2004.

[22] Osman Hassab Elgawi. Online random forests based on corrfs and corrbe. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–7. IEEE, 2008.

[23] E.F. Fama. Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, pages 383–417, 1970.

[24] C. Ferri, P. Flach, and J. Hernández-Orallo. Delegating classifiers. In *International Conference in Machine Learning*, 2004.

[25] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[26] J.H. Friedman and B.E. Popescu. Predictive learning via rule ensembles. *Annals of Applied Statistics*, 2(3):916–954, 2008.

[27] Juergen Gall and Victor Lempitsky. Class-specific hough forests for object detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1022–1029. IEEE, 2009.

[28] S. Gjerstad and M.C. Hall. Risk aversion, beliefs, and prediction market equilibrium. *Economic Science Laboratory, University of Arizona*, 2005.

[29] Jonathan L. Gross and Jay Yellen. *Graph Theory and its Applications.* Chapman & Hall/CRC, Boca Raton, Florida, 2006.

[30] Trevor J.. Hastie, Robert John Tibshirani, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer, 2009.

[31] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. Technical report, DTIC Document, 1996.

[32] B. Kulis and P.L. Bartlett. Implicit Online Learning. In *ICML*, 2010.

[33] N. Lay and A. Barbu. Supervised Aggregation of Classifiers using Artificial Prediction Markets. In *ICML*, 2010.

[34] Nathan Lay. Supervised aggregation of classifers using artificial prediction markets. Master's thesis, The Florida State University, Tallahassee, Florida, November 2009.

[35] B.G. Malkiel. The efficient market hypothesis and its critics. *The Journal of Economic Perspectives*, 17(1):59–82, 2003.

[36] O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1 & 18, 1990.

[37] W. Robert Mann. Mean Value Methods in Iteration. *Proc. Amer. Math. Soc.*, 4:506–510, 1953.

[38] C.F. Manski. Interpreting the predictions of prediction markets. *Economics Letters*, 91(3):425–429, 2006.

[39] Tom M. Mitchell. *Machine Learning.* WCB/McGrow-Hill, Boston, MA, 1997.

[40] J. Perols, K. Chari, and M. Agrawal. Information Market-Based Decision Fusion. *Management Science*, 55(5):827–842, 2009.

[41] C.R. Plott, J. Wit, and W.C. Yang. Parimutuel betting markets as information aggregation devices: Experimental results. *Economic Theory*, 22(2):311–351, 2003.

[42] P.M. Polgreen, F.D. Nelson, and G.R. Neumann. Use of prediction markets to forecast infectious disease activity. *Clinical Infectious Diseases*, 44(2):272–279, 2006.

[43] C. Polk, R. Hanson, J. Ledyard, and T. Ishikida. The policy analysis market: an electronic commerce application of a combinatorial information market. In *Proceedings of the 4th ACM conference on Electronic commerce*, pages 272–273. ACM New York, NY, USA, 2003.

[44] W.H. Press. *Numerical recipes: the art of scientific computing.* Cambridge University Press, 2007.

[45] P Kohli R Girshick, J Shotton and A Criminisi. Efficient Regression of General-Activity Human Poses from Depth Images. In *Proceedings of the 13th International Conference on Computer Vision*, 2011.

[46] L.K. Saul and S.T. Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *The Journal of Machine Learning Research*, 4:119–155, 2003.

[47] R.E. Schapire. The boosting approach to machine learning: An overview. *LECTURE NOTES IN STATISTICS-NEW YORK-SPRINGER VERLAG-*, pages 149–172, 2003.

[48] A. Storkey, J. Millin, and K. Geras. Isoelastic agents and wealth updates in machine learning markets. *ICML*, 2012.

[49] Luis Torgo. Regression data sets, 2010.

[50] F. Tortorella. Reducing the classification cost of support vector classifiers through an ROC-based reject rule. *Pattern Analysis & Applications*, 7(2):128–143, 2004.

[51] Zhuowen Tu. Auto-context and its application to high-level vision tasks. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[52] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[53] J. Wolfers and E. Zitzewitz. Prediction markets. *Journal of Economic Perspectives*, pages 107–126, 2004.

BIOGRAPHICAL SKETCH

Nathan Lay was born and raised in Boca Raton, FL where he grew passionate interest in mathematics and computer science in his later high school years. He attended Florida State University in 2003 and pursued a major in pure mathematics and a minor in computer science and earned a bachelors degree in pure mathematics cum laude in 2007. In his senior year of undergraduate studies, he briefly worked with Mark Sussman, Yousuff Hussaini, his former student Edwin Jimenez, and former postdoctoral research associate Svetlana Poroseva on fluid dynamics, Monte Carlo techniques, uncertainty quantification of hurricane models, and survivability of power systems. His work contributed to four corresponding papers. Following his dual passion, he entered the newly formed scientific computing graduate program at Florida State University pursuant of a masters degree where he met his current adviser Adrian Barbu. With similar interests and under the guidance and support of his adviser, Nathan worked on face detection and later produced a novel aggregation technique based on prediction markets. He earned a masters degree in 2009 and a Ph.D. in 2013 with his current adviser in the areas of machine learning and biomedical imaging. He was hired by Siemens Corporate Research first as an intern in June 2011 and later as a Research Scientist in May 2012. He works primarily on machine learning and computer vision tasks in medical imaging.

Nathan's research interests include machine learning, computer vision, and mathematics. His hobbies intertwine with his research interest but also include fishing, scuba diving, computers, table tennis, rock climbing and miscellaneous mathematics.