Threshold Optimization in Multiple Binary Classifiers for Extreme Rare Events using Predicted Positive Data

Edgar Robles*, Fatima Zaidouni*, Aliki Mavromoustaki, Payam Refael
Univeristy of Costa Rica, San José, Costa Rica
University of Rochester, Rochester, NY, USA
Institute of Pure and Applied Mathematics, University of California Los Angeles, CA, USA
Google Inc. , Los Angeles, CA, USA
{edgar.roblesarias@ucr.ac.cr, fzaidoun@u.rochester.edu, aliki.mavromoustaki@gmail.com, payamiam@google.com}

Abstract

Binary classification is challenging when dealing with imbalanced data sets where the important class is made of extremely rare events, usually with a prevalence of around 0.1%. Such data sets are common in various real-world problems. Examples are medical diagnostics where the disease is considered a rare event, or multiple types of fraud detection where regular transactions are the most prevalent. The events are categorized as either predicted positive or predicted negative against a certain threshold. In large imbalanced data sets, it is expensive to verify, through human raters, all the predictions made by the classifier. As a result, only predicted positive events are rated. Additionally, in most industrial applications, it is useful to combine multiple classifiers to make a decision. We developed a machine learning pipeline which, combined with expert knowledge, decides on an optimal threshold. ROC curves are reformulated to true positive (TP) versus false positive (FP) curves. We propose two solutions to select an optimal threshold by maximizing the area under the curve (AUC): a graph-based approach and an analytic approach. The graph-based approach constructs a graph to select an optimal path in the threshold space and the analytic approach minimizes an energy function. Our results agree with the Google team's manual attempts to choose the operating point in their private data sets and binary classifiers while providing a rigorous mathematical framework. Our solutions built on the Google team's expert knowledge efforts which identified the marginal precisions used in our methods. To further evaluate the performance of our algorithm, we split 3 public data sets into training and testing sets and used them to train five different classifiers. The results show improvement of the f-1 measure by 1.5%, the precision was improved by an average of 5.1% and, the recall was reduced by 1.6% on average. Depending on the purpose of the classification, we show how to reverse this trade-off.

*Co-First authors: Both authors contributed equally to this work.

Introduction

Our study focuses on data sets with extreme rare events, where the minority class accounts for 0.01% to 0.19% of the population. Such data is very common in many applications including fraud detection where fraudsters constitute a small percentage of users, medical diagnostics, face detection, money laundering, and intrusion detection. In such data sets, the distribution of positive to negative classes is extremely imbalanced and it is typically more important to correctly detect the positive class (Aggarwal 2014) that corresponds to the rare events. For instance, one of the examples used in the result section is a public data set on credit card fraud detection where the positive class accounts for 0.17% of all transactions.

In the rest of this paper, we provide some background about binary classifiers and their performance, we summarize related work and describe the problem statement in detail. Then we propose solutions for single classifier thresholding and for an ensemble of classifiers where two methods are proposed; the analytic and graph approach. Finally, we explain two experiments to benchmark our results and evaluate the performance of our algorithm, one using public data sets, and the other one using private Google data sets.

Background

Binary Classifiers

A binary classifier is a model that classifies its input into two classes; a positive and negative class. In this paper, we call the classifier's input a document, and its output a score. The higher the score, the more the classifier is confident that the document is in the positive class and vice versa. In the threshold binary model, a threshold number is picked and compared to the output score of the classifier to decide which class is predicted. If the score is larger than the selected threshold, the document is classified in the positive class. If the score is lower than the threshold, the document is classified in the negative class.

Classifier performance

To evaluate the performance of the classifier, the documents are often presented to human raters to determine the true positives, the false positives, and the true and false negatives. A higher threshold will cause the predicted class to be-

come more precise on average while losing some recall and a lower threshold will cause the predicted class to become less precise on average while gaining some recall. Note that:

$$\text{Precision} = \frac{TP}{TP + FP}$$

and

$$\text{Recall} = \frac{TP}{TP + FN}$$

This results in a trade-off where a very low threshold might yield many false positives and a high threshold might yield many false negatives. This trade-off is usually visualized with a receiver operating characteristic (ROC) curve that plots the true positive rate (TPR), defined as

$$TPR = \frac{TP}{TP + FN},$$

versus the false positive rate (FPR), defined as

$$FPR = \frac{FP}{FN + TP},$$

as suggested in (Bradley 1997) and (Provost and Fawcett 2001). A random classifier would be located along the main diagonal (Fawcett 2006) and a good classifier would have points in the upper left corner. This is visualized in Figure 1. A widely used performance measurement is the area under the curve (AUC) of the ROC. An optimal threshold is needed to minimize the false positives and negatives by maximizing the AUC.

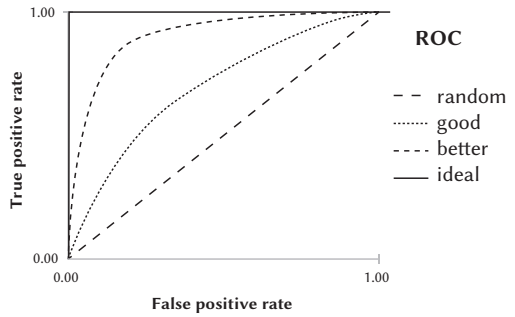

Figure 1: Typical ROC curve plotting TPR vs FPR. Random classifier produces a lines a long the diagonal, the closer the points to the upper left corner the better the classifier performance

Related Work

ROC and precision-recall analysis are widely used in imbalanced data set classification, as shown in various types of rare event data sets in (Kubat, Holte, and Matwin 1998), (Weiss 1999), and (Carvalho and Freitas 2002). However, choosing an optimal threshold also called "optimal operating point", is only well-understood for balanced data sets where there are approximately as many positives as negatives. A standard method is to take the intersection of the iso-performance tangent line in the ROC as the optimal cut-off (Hong 2009). The challenge arises when dealing with imbalanced data sets where rare events constitute the positive class.

The iso-performance line method maximizes the classification accuracy, which places more weight on the majority class, therefore overlooking the minority class in which we are interested the most. This method is therefore not applicable in our case, its pitfalls are discussed in detail in (Calabrese 2014) and (Fawcett and Provost 1996). Optimizing the threshold as a method to improve classification accuracy was explored in (Weiss 2004) and (Provost 2000) where rarity is explored and the challenges that it implies are investigated. In (Weiss and Provost 2003), the authors show that adjusting decision thresholds is an efficient way to deal with this bias. In (Calabrese 2014), the author suggests a quality measure called the cost-sensitive unweighted accuracy which uses the true rate as defined in (Hong 2009) as a quality criterion. It finds the optimal threshold that minimizes the difference between the simple mean of the Type 1 error, associated with false positives, and Type 2 error associated with false negatives. To correct for the imbalanced misclassification costs, the errors are weighted by their respective costs for which the ratio is usually known. An example using this method is found in (Roelens et al. 2018) for flood prevention data.

In the context of credit card assessment, an MCDM (Multiple Criteria Decision Making) was proposed in (Beulah Jeba Jaya .Y 2015) to select an optimal probability cut-off using hybrid data mining approaches.

Ensemble classifiers are common methods to combine multiple classifiers, each one looking at different features of a dataset. In (Breiman 2001), a method is shown where a group of poor classifiers can be harnessed together to obtain a way better performance. (Dietterich 2000) shows how different methods for voting in these ensembles of classifiers, from bayesian averaging to error-correcting, output coding, and boosting can often perform better than a single classifier.

Different thresholding methods were suitable for a range of applications, we identified few very important challenges, described below, that are faced when dealing with rare event data sets, which were not addressed before, in our knowledge.

Problem Description

In an extreme rare event problem, if one positive event happens in every ten thousand samples, one would have to label a million samples to detect 100 positive events. Like the data we are dealing with from Google, many datasets will lack information about the true and false negatives, because it is very expensive to have humans spend time rating such a large amount of data. Therefore, it makes sense that the only data classified by humans is the data that scored higher than a given score on the classifier i.e the positive class. Unfortunately, not knowing the true and false negatives, therefore, calls for analysis methods that do not require them.

The second important challenge to address occurs when using multiple classifiers to evaluate a single document. It often happens that we need to look at different aspects of

the document using different classifiers before making a decision. For instance, one classifier can be assigned images and another classifier can look for text. Each classifier will output a different score and after a threshold is chosen, it can be compared to each of the score and a decision can be made about whether the document belongs to the positive or negative class. The difficulty arises in finding a threshold that optimizes all the models jointly. This takes us from a single-dimensional optimization problem to a multi-dimensional one. In the following sections, we will explore the methods developed to address these challenges in the context of extreme rare event data sets.

Proposed Methods

For a single classifier

For the sake of completeness, we begin with the simple case of one classifier where we start defining the framework of our methods.

Let X_i be the independent and identically distributed (iid) random variable representing the score of positive documents. The number of true positives at a threshold t is the number of positive documents greater than the score t. Hence, for each threshold t we have TPR as:

$$\frac{1}{n}\sum_{i=1}^{n} I_{X_i \geq t},$$

where n is the total number of positive samples and $I_{X_i \geq t}$ is the indicator function. Similarly, let Y_i be the iid random variable of the score of negative documents, then FPR is given as:

$$\frac{1}{m}\sum_{i=1}^{m} I_{Y_i \geq t},$$

where m is the total number of negative samples. The ROC curve is then parametrized as

$$\left(\frac{1}{m}\sum_{i=1}^{m} I_{Y_i \geq t}, \frac{1}{n}\sum_{i=1}^{n} I_{X_i \geq t}\right), \quad t \in \mathbb{R},$$

and is expected to converge to $(P(Y_i \geq t), P(X_i \geq t))$. The trade-off optimization problem then becomes one of finding a threshold t that maximizes $P(X_i \geq t)$ and minimizes $P(Y_i \geq t)$. However, as mentioned in the introduction, we are addressing the situation where only the portion of the documents having scored higher than a certain threshold are rated. In other words, the distributions are left-censored, we only have $I_{X_i \geq t}, I_{Y_i \geq t}$ for $t \geq 0$ and we only know the total number of sample $m + n$ but do not have knowledge of either m or n. Hence, we are unable to plot an ROC curve.

Since we lack information about the true and false negatives, we use a reformulation of the ROC curve that plots TP instead of TPR and FP instead of FPR , this removes the normalization to probabilities that a normal ROC does by dividing by the positives or negatives. The reformulation is referred to as the positive-negative (PN) curve as defined in [8]. In our paper, we also refer to it as a partial ROC curve. The appearance is the same between the PN and ROC curve, but the chance-line (diagonal) in the ROC becomes the break-even line as known in information retrieval, a more detailed explanation is given in [8].

Mathematically,

$$\left(\sum_{i=1}^{m^*} I_{Y_i \geq t}, \sum_{i=1}^{n^*} I_{X_i \geq t}\right), \quad t \in \mathbb{R}_{\geq 0},$$

where m^* and n^* are the total number of positives and negatives, respectively, having scores greater than 0 for instance.

Given a classifier, a low threshold T can be set, the documents with a score higher than T can be sent to human raters to determine the TP and FP, then in the resulting data set, the threshold is raised to see how the TP and FP change. Note that as the threshold keep getting higher, we would have fewer data points in the sub-sample and therefore a wider confidence interval. Using the partial ROC curve implies that we can not use the rated data to simulate the TP and FP count had the model threshold been set even lower than T.

With the partial ROC curve, we can formulate our optimization problem as follows:

Given any point on the curve, if this point was the operating point and we wanted to get one additional true positive document, we can determine the cost that we need to pay in terms of a raw number of false positives.

In many situations, this formulation can relate to a business goal. For example, the positive class could be unwanted documents that a company needs to filter out without losing the wanted documents. Thus, they can provide a number to quantify how many good documents they are willing to give up in order to filter out one additional bad document. This can be answered by analyzing monetary costs for instance. Based on this, a target derivative $\frac{FP}{TP}$ can be chosen, again this represents the amount of false positives gained vs the amount of true positives gained at a certain threshold. The target derivative is based on expert knowledge, which was provided for us by Google in our application.

A partial ROC curve can be described as a parametric function

$$f(t) = (FP(t), TP(t)), \text{ where } f : T \to \mathbb{R}^2,$$

where T is the threshold space. A fit can be applied to find the points where the derivative has the desired value. In the result section, we present the fitting methods that worked best for typical partial ROC curves.

For an ensemble of classifiers

In this case, the PN curve (partial ROC) becomes an N-dimensional manifold, $\frac{dTP}{dFP}$ becomes a more complicated notion since a local direction must be picked for a unique derivative to exist.

In general, one can define the function as

$$f(\vec{t}) = (FP(\vec{t}), TP(\vec{t})); \; f : \hat{T} \to \mathbb{R}^2, \tag{1}$$

where $\hat{T}$ is the threshold space described by the cross product of all the threshold values for each classifier.

We can consider the derivative at a point to be:

$$\lim_{\Delta x \to 0} \frac{\Delta \mathrm{TP} \cdot u}{\Delta \mathrm{FP} \cdot u},$$

where u is a vector that describes the direction at which the derivative is being taken.

We describe a path

$$\phi(s) = (g(s), h(s)) \text{ for } 0 \leq s \leq 1$$

over $\hat{T}$ space, that is, a function $\phi : [0, 1] \to \hat{T}$.

We impose the following restrictions on ϕ for the resulting curve to behave like a partial ROC curve:

- $\phi(0) = (\inf T_1, \inf T_2, \ldots, \inf T_n)$
- $\phi(1) = (\sup T_1, \sup T_2, \ldots, \sup T_n)$
- $f(t)$ and $g(t)$ must be monotonous

The first two conditions are so that the curve starts and ends at the correct points, and the last point is so that the amount of true positives and false positives decrease with respect to our new "threshold".

To pick the best classifier, ϕ should maximize the area under the curve:

$$\mathrm{argmax}_{\phi} \int_{\phi([0,1])} TP \; dFP$$

We propose two algorithms to find ϕ, one is based on minimizing a function (the analytic approach) and the other one is based on building a graph (the graph approach).

Before we give a step by step example implementation for both methods, we introduce the following simple base algorithm for both methods to find the optimal operating point.

Algorithm 1 Base algorithm for finding the optimal operating point

Input: Set of points P representing $(TP, FP, T1, T2, ...)$, target derivative d
Output: Optimal operating point $(T1, T2, ...)$

1: $\mathcal{C} \leftarrow$ FindOptimalCurve(P)
2: Find point q where $\frac{d}{dt}\mathcal{C}(q) = d$
3: **return** q

In what follows, we explain the FindOptimalCurve function used in Algorithm 1 in the context of both the analytic and graph approach.

Analytic approach

The core of this approach is the usage of a family of functions for which we optimize the parameters in a way that maximizes the area under the corresponding PN curve. This approach estimates the integral yielding the area under the curve using the simple trapezoidal method. The algorithm implementation is summarized in Algorithm 2.

We describe the implementation of this algorithm in more detail using two thresholds, as an example, in the following steps:

Algorithm 2 Analytic implementation of FindOptimalCurve

Input: Set of points P representing $(TP, FP, T1, T2, ...)$, a family of curves f_p defined on $[0, 1] \to T$
Output: A curve $\mathcal{C}$

1: Fit a function $\hat{F}(T1, T2, ...)$ as the best fit for the points $(FP, T1, T2, ...)$
2: Fit a function $\hat{T}(T1, T2, ...)$ as the best fit for the points $(TP, T1, T2, ...)$
3: Let $g_p \leftarrow (\hat{T}, \hat{F}) \circ f_p$
4: $q \leftarrow \mathrm{argmax}_p AUC(g_p(t))dt$
5: **return** $\{(t, f_q(t)) \forall t \in [0, 1]\}$

1. Normalizing the thresholds
2. Plotting the three-dimensional plots (TP, T1, T2) and (FP, T1, T2) and fitting a surface through each one of them using an interpolation technique. We create an $n \times n$ grid on the threshold plane with each point on the grid corresponding to a pair of thresholds and a value of TP and FP. This step allows finding the estimators $\hat{\mathrm{TP}}$ and $\hat{\mathrm{FP}}$ at each point of the surface.
3. Turning the grid into a linear space; since the data is normalized at this point, the domain will be from 0 to 1. We can also specify the granularity i.e the size of the partitions in this domain. In the next steps, the granularity will also indicate the number of AUC samples that are calculated in the process.

   This step is the start of the parametrization used in this approach. If s denotes the parameter indicating the order of the partitions, then both thresholds are a function of s which is then used as an index in a separate parametric function to output the corresponding TP and FP values separately.
4. Defining an energy function as follows:

   $$E(\vec{p}) = \mathrm{AUC}(\hat{\mathrm{FP}} \circ \phi_{\vec{p}}, \hat{\mathrm{TP}} \circ \phi_{\vec{p}})$$

   where:

   $\phi_{\vec{p}}$ is a family of parametric functions defined from [0,1] to $T_1 \times T_2$ with $\vec{p}$ being the parameter. In our algorithm, we used a parametric function that is graphically close to the resulting discrete path and that passes through the bounds (0,0) and (1,1):

   $$\phi_{\vec{p}} = x^p \tag{2}$$

   AUC is a function that calculates the area under the curve using the trapezoidal formula, this is an estimation of the integral form of the calculation of the area under the curve described by: $\int_{\{(x, \phi_{\vec{p}}(x))\}} \mathrm{TP} \quad d\mathrm{FP}$
5. Minimizing the function below using the *Nelder-Mead* minimization function.

   $$-\log(\log(E))$$

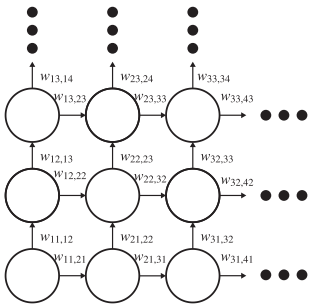

Figure 2: The graph used in this approach; each circle represents a node which is a 4-tuple from the data (T1, T2, TP, FP) and each arrow represents an edge.

Note that we made the previous function negative so we can use a minimization method to maximize the AUC. We also composed E with two logarithms in order to make it converge faster, otherwise, it might get stuck on local minima.

6. Creating a function that determines the curve in the TP-FP space that corresponds to the resulting path within the threshold space using the parameterization mentioned in Step 3.
7. Finding the point s where $\frac{dTP}{dFP}$ w.r.t. the parameterization s equals the target marginal precision, by estimating the derivative of $f \circ \phi$.
8. Finding the optimal operating point (T1,T2,TP,FP) given a target marginal precision ie. the values of t where $\phi(t) = s$. It can be interpolated at the value of s given by the closest marginal precision to the target.*Ridders* method can be used for root finding to get the best parameter s.

Graph-based approach

The goal of the graph-based approach is again to find the continuous path on the threshold space that maximizes the area under the curve in the TP-FP space, now with the constraint that every point is only allowed to move through its neighbors.

We defined the node, edge, and weight of the graph shown in Figure 2, for 2-dimensions, as follows:

- Node v: represents a pair of thresholds, having attributes of (T1,T2,TP,FP)
- Edge e: the connection between the nodes is established if and only if one of their thresholds is equal or their other threshold is consecutive in the discrete set of thresholds.
- Weight of edge $e = (v_1, v_2)$: this is equal to the negative of the area under the segments (trapezoid) $(\mathrm{FP}(v_1), \mathrm{TP}(v_1))$ and $(\mathrm{FP}(v_2), \mathrm{TP}(v_2))$ as shematized in Figure 3. Mathematically, the weight is defined as:

$$w(e) = -\frac{1}{2}(|\mathrm{FP}(v_1) - \mathrm{FP}(v_2)|)(\mathrm{TP}(v_1) + \mathrm{TP}(v_2)) \tag{3}$$

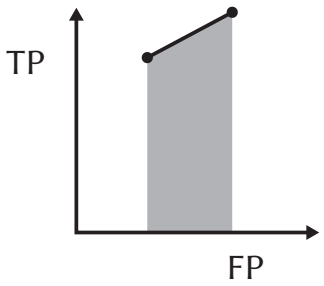

Figure 3: The trapezoid area under a single segment between two edges, the weight is defined to be the negative of the highlighted area

Then the total weight of edges in a path is the negative of the area under the curve swept out in the TP vs FP plot. In another words, by fixing end points, we can find a better TP vs FP curve by finding a path that has a minimal weight. So, it suffices to find the shortest path from v_0 to v_∞ ,with $(\mathrm{t1}(v_0), \mathrm{t2}(v_0)) = (0, 0)$, $(\mathrm{t1}(v_\infty), \mathrm{t2}(v_\infty)) = (\infty, \infty)$. Intuitively, it finds a shortest path from the left bottom corner to the right upper corner in the thresholds space $\mathrm{T}_1 \times \mathrm{T}_2$. The edge is directed towards up or right to avoid bent paths in the corresponding TP vs FP curve.

After we find the set of nodes in the shortest path, we connect the corresponding points and re-parametrize the path. Note that it would be continuous but not smooth.
The algorithm for this method is the following:

Algorithm 3 Graph based implementation of FindOptimalCurve

Input: Set of points P representing $(TP, FP, T1, T2, ...)$ sampled in a grid with respect to the thresholds
Output: A curve $\mathcal{C}$

1: Create a graph $\mathcal{G}$ connecting points to neighbors with a greater coordinate than them
2: Let the origin be A_0 and the point with the biggest coordinates be A_∞
3: $P \leftarrow$ FindOptimalPath(G, A_0, A_∞)
4: Let $f(t) : [0, 1] \rightarrow FP \times TP$ such that it passes through the path P
5: **return** $\{(t, f(t)) \forall t \in [0, 1]\}$

Below we explain this approach in detail, starting from step 3 of the analytic approach for a 2-dimensional example, since the first two steps are similar.

1. Transforming the threshold space into an $n \times n$ grid with each point on the grid corresponding to a pair of thresholds and a value of TP and FP.
2. Calculating the weight of the graph as the negative area under the segment formed by 2 points v_1, v_2 in the TP vs FP curve using equation 3
3. Determining the weight of the edge connecting the node of two consecutive points.
   Note that the predicted positives have to be non-negative,

hence the segment is above the x-axis. The weight is non-positive therefore a lower weight indicates a higher area under the segment.

4. Store the $(\mathrm{TP}, \mathrm{FP}, T_1, T_2)$ tuple in every node as extra information. The edge can then be defined as a 3-tuple of $(n_k, n_l, w_{k,l})$, where n_k and n_l are each node and $w_{k,l}$ is the negative area under the curve of each node.
   The nodes are connected if they are within the same neighborhood. A function can do so if the one of the thresholds is the same between two neighbours or the thresholds are consecutive to each other in the discrete set of thresholds.
   The path selected is therefore just a collection of nodes from the left bottom to right upper corner in thresholds space and the total weight of the path corresponds to the negative area under the curve swept out by it in the TP - FP space.

   Note that in order to compute the shortest path between the lower-left corner (source node) to the top right corner node, the *Bellman-Ford* algorithm can be implemented since it accepts edges with negative weights unlike the usual *Dijkstra*'s algorithm.
5. Connecting the nodes to obtain a continuous path in the threshold space and determining the curve in the TP-FP space that corresponds to the selected path. It does so using the same parametrization explained in step 3 of the analytic approach.
6. Getting the marginal precision, refer to step 7 of the analytic approach.
7. Determining the optimal operating point given a target marginal precision, refer to step 8 of the analytic approach.

Experiments and Results

Evaluating the performance using public datasets

To evaluate the performance of the method, we use rare event data sets where the amount of false negatives and true positives can be calculated, in order to compare the precision and recall of two standard classifiers with our methods on the resulting ensemble of classifiers. This was done using three rare event datasets[1][2][3] whose true and false positive and negative values are known.

Each dataset was cleaned by changing empty values for the average value in the column, turning categorical datasets into one-hot encoded vectors, and encoding cyclical values like days of the weeks into points in a unit circle. These datasets were then passed through a PCA procedure (Principal Component Analysis) to reduce the dimensionality. The data sets were then split into a training set T and a holdout testing set H, where T represents 75% of the data set and H represents the other 25%.

[1]https://www.kaggle.com/mlg-ulb/creditcardfraud

[2]https://www.kaggle.com/henriqueyamahata/bank-marketing

[3]https://www.kaggle.com/c/ieee-fraud-detection

Each classifier was trained using the first 20% of the components of the PCA. We need to simulate a joint optimization of two classifiers where each one of them makes decisions over a separate but related data set. Thus, one of the classifiers was assigned the remaining even components and a second classifier of the same type was assigned the remaining odd components.

The hyperparameters, for each classifier, were tuned by maximizing the average accuracy in a cross-validation test. A cross-validation test was done by splitting the training set T into 5 different cross-validation groups: $T_1, T_2, ...T_5$, where for each training set T_i, a classifier is trained using the set $\bigcup_{j \neq i} T_j$ and the score for the ith fold is denoted by s_i and is obtained by evaluating the accuracy of the classifier on T_i. The hyperparameters with the highest average value of $s_1, ..., s_5$ are picked.

The classifiers used for the testing were Linear SVMs, Random Forests, Decision Trees, k-nearest neighbors, XGBoosting and Naive Bayes.

We explored multiple classification techniques to determine whether our methods improve their performance. We relied on (Pedregosa et al. 2011) for a compilation of classification algorithms. In all classifiers, we estimated the hyperparameters through a grid search with cross-validation. For the k-nearest neighbors classifier, we tested all the values of k from 1 to $\sqrt{n}$ where n is the number of points in the training data, as well as the L_1 and L_2 metrics, the default behavior for classification, is to pick the class with the biggest number of nearest neighbors assigned to them. The decision tree classifier used the grid search coupled with the Gini and entropy criterion. We optimized the maximum depth parameter for all integer values between 1 and 500 as well as without setting a maximum depth, the default classification method is to follow the instructions for the tree. The random forest classifier also used the Gini and entropy criterion with the number of estimators between 50 and 500, with steps of 10 between them, the default behavior for classification is to let each decision tree in the forest vote, and the class with the most votes wins. For the extreme gradient booster (XGboost), we used a maximum depth from 1 to 20, a learning rate from 0.1 to 1, with steps of 0,1, and a number of estimators from 50 to 500 with a step of 10, the default classification behavior is the same as random forests. For the linear SVMs, we again used the grid-search with cross-validation to look for the hyperparameters between using L1 and L2 norm for the penalty, the default threshold used was 0. For SVMs, we used the hinge or squared hinge loss functions to train the classifier with a dual true or false and using all values of the regularization parameter λ from 0 to 1 with a step of 0.1, the default threshold is 0. We also used a Naive Bayes classifier with no optimized parameters, whose default classification behavior is to choose the class with the highest probability.

A grid was made using the result of the cross product of the thresholds of equally spaced points in each dimension. An input would be considered to have a true value if either of the scores produced by each classifier surpassed their own threshold. This means that we are taking the union of the classification. Using the scores outputted by the classifiers,

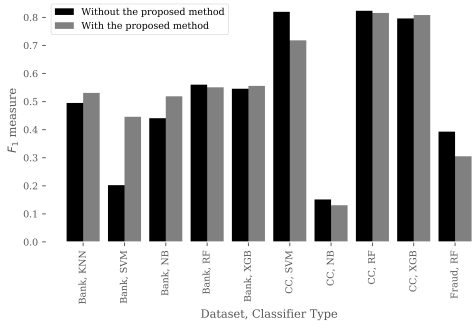

Figure 4: The F_1 measure of the best performing method to create the ensemble against the average F_1 measure of the two classifiers that compose it for each pair of dataset and classifier type. The classifier types are abbreviated as the following: KNN is K-nearest neighbors, SVM is a linear SVM, NB is Naive Bayes, RF is Random Forests and XGB is XGBoosting.

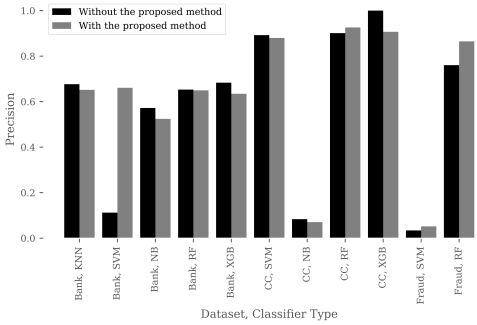

Figure 5: The precision of the best performing method to create the ensemble against the average F_1 measure of the two classifiers that compose it for each pair of dataset and classifier type. The classifier types are abbreviated as the following: KNN is K-nearest neighbors, SVM is a linear SVM, NB is Naive Bayes, RF is Random Forests and XGB is XGBoosting.

we calculated the amount of false positives and true positives for each pair of thresholds in the grid. This was the input for Algorithm 1. We then calculated the precision and recall from the testing set.

To evaluate their performance, the F_1 measure was used instead of the accuracy score. This latter is not suitable for rare event data sets where a difference in accuracy of 0.01% can make a huge difference in both the precision and recall. The F_1 measure is calculated by

$$F_1 = 2\frac{pr}{p+r},$$

where p is the precision and r is the recall.

Figures 4, 5 and 6 show the F_1 measure, precision, and recall, respectively, for each experiment. An experiment was defined as a combined pair of a dataset and a classifier type. When either the precision or recall were extremely low, the other quantity is "undefined". In this case, taking the union of the classifiers' decisions results in classifying every document as positive, which makes the ensemble classify every document as positive.

Overall, the precision was increased on average by 5.1% while the recall was reduced by an average of 1.6% and F_1 measure increased by an average of 1.5%.
Table 1 and Table 2 of the appendix show the results for all the simulated experiments with all the mentioned classifiers. Undefined values are showed as - on the tables. As the F_1 measure of the classifiers that do not use the proposed methods increases, the F_1 measure of the ensemble resulting from the proposed methods also increases. This trend is shown on figure 7. It also plots the identity line to show that the proposed methods improve the F_1 measure of all the points above this line.

Benchmark using Google data sets

The Google data contains 22 data sets each consisting of about 10,000 rows and 4 columns of data. The two first

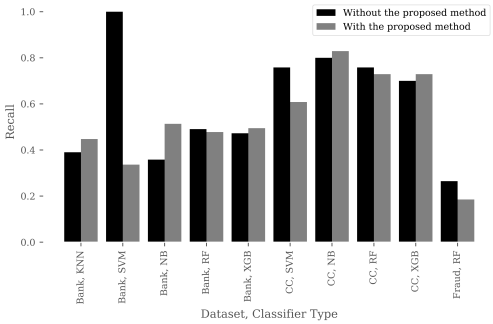

Figure 6: The recall of the best performing method to create the ensemble against the average F_1 measure of the two classifiers that compose it for each pair of dataset and classifier type. The classifier types are abbreviated as the following: KNN is K-nearest neighbors, SVM is a linear SVM, NB is Naive Bayes, RF is Random Forests and XGB is XGBoosting.

columns represent two thresholds of two different classifiers and the next two represent the number of true and false positives for each pair of thresholds as determined by human raters, assuming all documents over the threshold is considered as part of the positive class. These datasets, have a prevalence of the positive class of about 0.01% of the total number rows.

The result of running Algorithm 2 and 3 is shown in Figure 10 for an example data set. The figure represents the threshold paths obtained using both the analytic and graph approach, as well as the location of the operating point.

Using the parameterization from Equation 1, we can use the threshold paths to determine the optimal partial ROC curve that was selected by each of these algorithms.

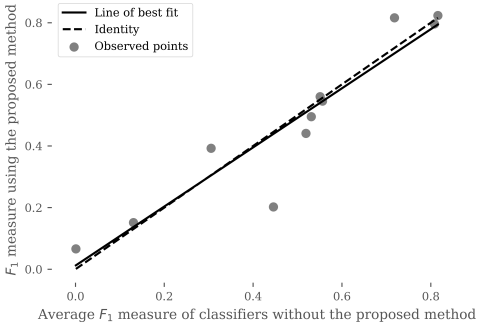

Figure 7: The average F_1 measure of the classifiers without the proposed method compared to the F_1 measure of the with the proposed method.

Next, we can determine the optimal operating point given a target marginal precision as determined by a business goal set by Google's expert knowledge efforts prior to this work. The result of running Algorithm 1 is shown for 3 different methods in Figure 8 in the threshold space and correspondingly in Figure 9 in TP vs FP space.

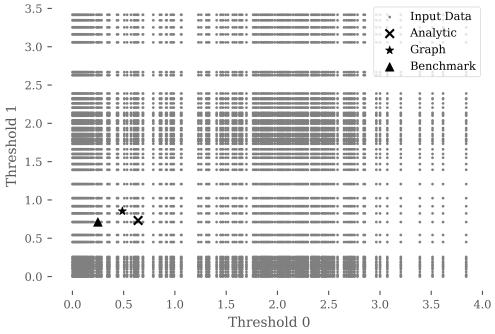

Figure 8: The operating point picked by each algorithm in the threshold space.

The analytic and graph methods mentioned in figures 8 and 9 are the proposed methods, while the benchmark is a fine-tuned operating point used by Google based on separating the false positives into small intervals called buckets, and picking the highest TP for each bucket, then adjusting this manually to fit the goals of the optimal operating point. While this latter is good as a benchmark for our algorithms, it can not be used as another optimization method since it lacks mathematical rigor and produces an erratic behavior in the threshold space which, without fine-tuning, would imply the non-existence or non-uniqueness of the derivative of the partial ROC curve needed to find the operating point.
Our methods make a good attempt at approximating the operating point selected by the benchmark method. These points are also plotted on their respective paths in Figure 10.

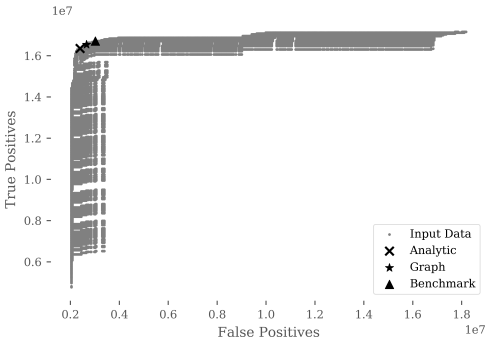

Figure 9: The operating point picked by each algorithm in the TP vs FP space.

Discussion

We used an or (union) in combining the decisions of the classifiers but it can be substituted by an and operation depending on whether the application prioritizes positive classification or negative classification. Our methods also assume that the ROC curve for both of the classifiers has a unique decreasing derivative which can be expected from a tuned classifier.
Both the analytic and graph methods were presented as although both useful, there are different disadvantages to each one of them. The analytic method is heavily dependent on the family of functions used to optimize over, and is prone to get stuck in an optimal minimum. On the other hand, it is uncertain whether the graph method is prone to overfitting to the data or not, however, unlike the analytic method, there is no way to mitigate it. A possible alteration to be done to mitigate overfitting with the graph method would be to hold a series of simulations where some connections between the nodes are deleted, and then picking the average path between the resulting set of paths.

Both algorithms scale reasonably well with respect to the amount of data. The graph algorithm scales at a rate of $O(n^k)$, where n is the number of rows and k is the number of classifiers. This is good when the number of dimensions is low, however, it scales exponentially with the number of classifiers in the ensemble. The analytic approach works the opposite way since it depends on an optimization problem, it does not scale very well with respect to the number of rows, however, it scales linearly with the number of dimensions. We compared the running time of the algorithms using a computer with an i7-6700 and 32GB of RAM for all the Google data sets combined. The analytic approach ran on average 10 times faster than the graph approach; for $n = 10,000$ and $k = 2$, the analytic method takes 60 seconds while the graph method takes 6 seconds in our conditions.

Finally, the experiments were done using an automated method to tune the classifiers, notably the grid-search with cross-validation in order to obtain a large number of samples across different accuracies. However, this could be improved since it does not reflect a real-world application of this pro-

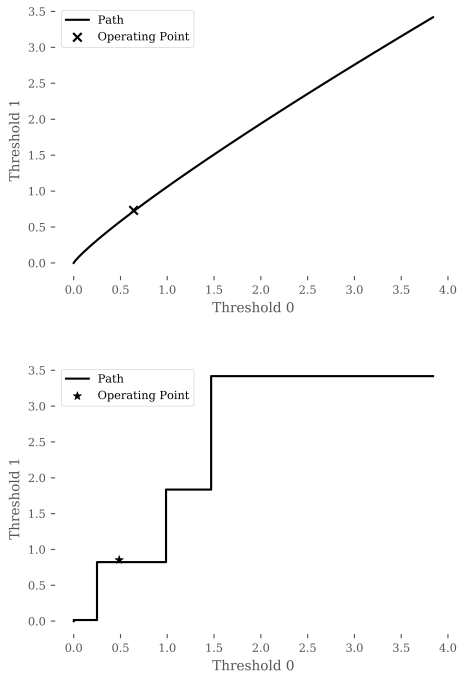

Figure 10: The path picked by the analytic (top) and graph method (bottom), using the function in equation 2 for the analytic method.

cess in which the hyperparameters would be estimated with the help of additional fine-tuning methods that depend on the specific application of the data set.

Conclusion

The proposed algorithm finds optimal thresholds for the output of binary classifiers used for rare event datasets. The method works by finding the best possible path in the TP - FP space, which is the one that maximizes the area under the curve of the projected partial ROC curve. This curve is then used to find the optimal operating point corresponding to a given target derivative. The latter represents the tradeoff of lost false positives to gained true positives which was obtained from the Google team's expert knowledge efforts prior to this work. Two optimization methods are proposed: an analytical method, which optimizes parameters of an energy function and a graph method, which performs the optimization through finding the path in a graph to maximize the AUC.

Our results were very close to the optimal operating points that were picked manually by the Google team. We were, therefore, able to automate their process of finding an optimal threshold while providing a mathematical framework and using their target derivatives(marginal precisions). Moreover, we compared the F_1 measure, precision, and recall of combining five classifiers and three datasets while either relying on our thresholding methods and taking the union of the decisions or using the classifiers individually with their default classification behavior. The result improved the precision while lowering the recall less significantly. This tradeoff could be reversed, if desired, by simply changing the ensemble decision between the individual classifiers from 'or' to 'and'. We discussed important limitations and possibilities for future work. Notably, optimizing different metrics other than AUC individually or in combinations would be worth exploring. Moreover, we optimized two binary classifiers jointly but the theoretical framework described in the methods section extends to N dimensions. Using this in practice will require optimizing over multiple parameters in the analytic approach and extending the graph to an N-dimensional lattice. Exploring this possibility through practical applications might open great avenues for the general optimization of ensemble classifiers.

Acknowledgements

We would like to thank our teammates Ching Pui Wan and Joanne Beckford, NSF grant DMS-0931852, Payam Refael and his team from Google, our academic mentor Aliki Mavromoustaki, and all of the IPAM staff at UCLA for making this work possible.

References

Aggarwal, C. C. 2014. *Data classification: algorithms and applications*. CRC Press, Taylor & Francis Group.

Beulah Jeba Jaya .Y, J. J. T. 2015. Multiple criteria decision making based credit risk prediction using optimal cut-off point approach. *International Journal of Applied Engineering Research* 10:20041–20054.

Bradley, A. P. 1997. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition* 30(7):1145–1159.

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

Calabrese, R. 2014. Optimal cut-off for rare events and unbalanced misclassification costs. *Journal of Applied Statistics* 41(8):1678–1693.

Carvalho, D. R., and Freitas, A. A. 2002. A genetic-algorithm for discovering small-disjunct rules in data mining. *Applied Soft Computing* 2(2):75–88.

Dietterich, T. G. 2000. Ensemble methods in machine learning. In *Multiple Classifier Systems*, 1–15. Berlin, Heidelberg: Springer Berlin Heidelberg.

Fawcett, T., and Provost, F. 1996. Combining data mining and machine learning for effective user profiling. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 8–13. AAAI Press.

Fawcett, T. 2006. An introduction to roc analysis. *Pattern recognition letters* 27(8):861–874.

Hong, C. S. 2009. Optimal threshold from roc and cap curves. *Communications in Statistics - Simulation and Computation* 38(10):2060–2072.

Kubat, M.; Holte, R. C.; and Matwin, S. 1998. Machine learning for the detection of oil spills in satellite radar images. *Machine learning* 30(2-3):195–215.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Provost, F., and Fawcett, T. 2001. Robust classification for imprecise environments. *Machine learning* 42(3):203–231.

Provost, F. 2000. Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI'2000 workshop on imbalanced data sets*, volume 68, 1–3. AAAI Press.

Roelens, J.; Rosier, I.; Dondeyne, S.; Van Orshoven, J.; and Diels, J. 2018. Extracting drainage networks and their connectivity using lidar data. *Hydrological processes* 32(8):1026–1037.

Weiss, G. M., and Provost, F. 2003. Learning when training data are costly: The effect of class distribution on tree induction. *Journal of artificial intelligence research* 19:315–354.

Weiss, G. M. 1999. Timeweaver: A genetic algorithm for identifying predictive patterns in sequences of events. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, 718–725. Morgan Kaufmann Publishers Inc.

Weiss, G. M. 2004. Mining with rarity: a unifying framework. *ACM Sigkdd Explorations Newsletter* 6(1):7–19.

Appendix

Table 1: The results of the experiment being performed on multiple datasets with different types of classifiers. The empty values in precision or F_1 measure occur when every data point is classified as a negative, and so precision is undefined and recall is 0.

Dataset	Ensemble	Classifier	Precision	Recall	F_1 measure
CreditCards	LinearSVM	Analytical	0.88	0.76	0.82
		Graph	0.897	0.76	0.82
		Linear SVM 1	0.88	0.66	0.75
		Linear SVM 2	0.88	0.56	0.68
	Random Forest	Analytical	0.90	0.76	0.82
		Graph	-	0	-
		Random Forest 1	0.94	0.73	0.82
		Random Forest 2	0.92	0.73	0.81
	Decision Tree	Analytical	-	0	-
		Graph	-	0	-
		Decision Tree 1	0.6696	0.63	0.65
		Decision Tree 2	0.7818	0.72	0.75
	k-nearest neighbors	Analytical	-	0	-
		Graph	-	0	-
		k-nearest neighbors	0.8191	0.6417	0.7196
		k-nearest neighbors	0.9342	0.5917	0.7245
	xgboost	Analytical	0.9231	0.7000	0.7962
		Graph	1.0000	0.0083	0.0165
		xgboost	0.9140	0.7083	0.7981
		xgboost	0.9000	0.7500	0.8182
	Naive Bayes	Analytical	0.0836	0.8000	0.1513
		Graph	-	0.0000	-
		Naive Bayes	0.0671	0.8417	0.1243
		Naive Bayes	0.0746	0.8167	0.1368
Bank	LinearSVM	Analytical	0.1125	1.0000	0.2022
		Graph	-	0.0000	-
		Linear SVM 1	0.6633	0.3368	0.4467
		Linear SVM 2	0.6582	0.3359	0.4448
	Random Forest	Analytical	0.6529	0.4905	0.5602
		Graph	-	0.0000	-
		Random Forest 1	0.6477	0.4827	0.5532
		Random Forest 2	0.6516	0.4732	0.5483
	Decision Tree	Analytical	-	0.0000	-
		Graph	-	0.0000	-
		Decision Tree 1	-	0.0000	-
		Decision Tree 2	-	0.0000	-
	k-nearest neighbors	Analytical	0.6766	0.3903	0.4951
		Graph	-	0.0000	-
		k-nearest neighbors	0.6528	0.4482	0.5315
		k-nearest neighbors	0.6508	0.4473	0.5302
	xgboost	Analytical	0.6458	0.4724	0.5456
		Graph	0.6832	0.3929	0.4989
		xgboost	0.6359	0.4870	0.5516
		xgboost	0.6333	0.5026	0.5604
	Naive Bayes	Analytical	0.5724	0.3584	0.4408
		Graph	0.5694	0.3083	0.4000
		Naive Bayes 1	0.5284	0.5147	0.5214
		Naive Bayes 2	0.5206	0.5121	0.5163

Table 2: Continuation of Table 1

Dataset	Ensemble	Classifier	Precision	Recall	F_1 measure
IEEEFraud	k-nearest neighbors	Analytical	-	0.0000	-
		Graph	-	0.0000	-
		k-nearest neighbors 1	0.6380	0.0897	0.1573
		k-nearest neighbors 2	0.6476	0.0942	0.1645
	Naive Bayes	Analytical	0.0342	1.0000	0.0662
		Graph	-	0.0000	-
		Naive Bayes 1	0.0873	0.1356	-
		Naive Bayes 2	0.0840	0.0954	0.0004
	LinearSVM	Analytical	0.0342	1.0000	0.0662
		Graph	0.0000	0.0000	-
		Linear SVM 1	0.0175	0.0002	0.0004
		Linear SVM 2	0.0870	0.0008	0.0016
	Decision Tree	Analytical	-	0.0000	-
		Graph	-	0.0000	-
		Decision Tree 1	-	0.0000	-
		Decision Tree 2	-	0.0000	-
	Random Forest	Analytical	0.7605	0.2647	0.3927
		Graph	-	0.0000	-
		Random Forest 1	0.8652	0.1855	0.3055
		Random Forest 2	0.8641	0.1851	0.3049