AUC Maximization on Imbalanced Data with Random Forests and Stochastic Gradient Descent Methods

Kenneth R. DeVoe
Department of Mathematics
University at Albany, SUNY
Albany, NY 12222
kdevoe@albany.edu

May 11, 2021

Abstract

In machine learning applications, it's common for a dataset to contain many more instances from one class than another. Common examples include disease and fraud detection. For these datasets, common metrics for evaluating model performance, such as the error rate, are no longer useful. This paper seeks to compare several techniques for classifying unbalanced data, measuring model performance using the area under the precision-recall curve (AUC). Specifically, we compare the performance and run time of a random forest classifier (with and without oversampling and cost-sensitive approaches) with stochastic gradient descent algorithms meant to fit a linear model to the data, choosing the coefficients that maximize the AUC. After evaluating model performance on several smaller datasets, we then attempt to construct the best classifer on a large, highly imbalanced dataset of (non)fradulent credit card transactions.

1 Introduction

In supervised machine learning, a binary classifier takes a sample of labelled 'training data'

$$S = \{(\mathbf{x}_1, y_1), ...(\mathbf{x}_M, y_M)\}, y_i = \pm 1$$

and attempts to build a model to predict y_i based on the features $\mathbf{x}_i$. Model performance can then be evaluated by making predictions on a new set of unlabelled 'testing data',

$$S' = \{\mathbf{x}_{M+1}, ..., \mathbf{x}_n\}$$

and comparing the results to the true class labels, $Y' = \{y_{M+1}, ...y_n\}$.

In many applications, datasets contain imbalances where more instances belong to one class (the majority class) than the other (the minority class). This is common in disease, fraud, and rare event detection. In these applications, common metrics like the error rate are poor measures of model performance. Depending on the severity of the class imbalance, good error rates could easily be

achieved by having a model assume all instances belong to the majority class. For data sets with an imbalanced class ratio a better performance metric is the area under the precision-recall curve (AUC), defined as the probability that a randomly chosen positive instance is assigned a higher decision value than a randomly chosen negative instance.

In this paper, we'll consider the problem of constructing machine learning models with the intention of maximizing the AUC. We'll first consider a random forest classifier, then attempt to improve the performance of this classifier by oversampling the minority class with the SMOTE algorithm [1], as well as using a cost-sensitive approach with the MetaCost algorithm [2].

We'll then consider the problem of fitting a linear model to our data, while choosing the coefficients with the intention of maximizing the AUC. Two different stochastic gradient algorithms (SPAUC [3] and ADAM [4]) will be used to solve this problem.

2 Problem Definition and Algorithms

This section contains details on the algorithms used throughout this paper.

2.1 Trees and Forests

Classification Trees: An overview of decision tree algorithms can be found in [5]. For the purposes of this paper, we used a decision tree classifier with node impurity measured by the cross-entropy.

Random Forests[1]: A thorough description of random forest classifiers can be found in [5]. Random forests are built by building a collection of decorrelated decision trees and aggregating their predictions. We start drawing by B samples with replacement from the training data each of size N. We then perform the following sequence for every terminal node of the tree until the minimum node size is achieved.

1. Randomly select m features
2. Select the best variable and split point among the chosen features.
3. Split the node.

The procedure then gives us a collection of B decision trees, T_b. The random forest classifier then makes class predictions based on the majority prediction of the B decision trees.

2.2 SMOTE

Synthetic Minority Oversampling Technique (SMOTE) is an algorithm that produces synthetic data for the minority class in the training data to rectify the class imbalance. The algorithm works

[1]For the purposes of this project, we used the random forest classifier available in Python's sklearn package.

by selecting the k nearest neighbors of a minority class instance, $\mathbf{x}_0$, then randomly adds an new minority class instance on the line between $\mathbf{x}_0$ and each of it's neighbors. Depending on the severity of the class imbalance, not all neighbors may be used (one or two may be chosen at random) and we loop through the minority class instances until the class imbalance is rectified. A more detailed overview of the algorithm and its variations can be found in [1].

2.3 MetaCost

When working with imbalanced data, a cost sensitive approach involves imposing different costs on misclassification. These costs are generally represented by a matrix C, where $C(i, j)$ is the cost of predicting an instance belonging to class j belongs to class i. In order to use the cost matrix, our model needs to be able to predict class probabilities for each instance it's trained on. Common choices for costs are $C(i, i) = 0$ (there is no cost for a correct prediction) and $C(i, j) = \frac{P(i)}{P(j)}$.

MetaCost [2] is a cost-sensitive algorithm that can be implemented on any classifier. In the event the classifier is incapable of predicting class probabilities, these probabilities can be approximated empirically by running the classifier on repeated bootstrap resamples of the training data. The algorithm works as follows:

1. **Inputs**:
   - S Training Data
   - L classifier
   - C cost matrix
   - m number of resamples
   - n number of examples in each resample.
   - p True if L can produce class probabilities.
   - q if True, all resamples are used for each example, regardless of whether the example is in the resampling.
2. **Algorithm:**

   For i =1,...,m:
   S_i is a resampling of S (with replacement) of size n
   M_i is the model made by fitting L to S_i.

   For x in S:
   For all classes j:
   $P(j|x) = \text{mean}(P(j|x, M_i)$
   If p is true then $P(j|x, M_i)$ is produced by the model. Otherwise $P(j|x, M_i) = 1$ if the M_i predicts x is in class j and is zero otherwise.
   If q is true all M_i are used. If q is false M_i is only used if $x \in S_i$.
   x's new class is $\text{argmin}_i \sum_j P(j|x)C(i, j)$.
   M is the model produced by applying L to S with the new training labels.

   return(M)

For our applications, we'll set $C(i,i) = 0$ (there is no cost of making a correct prediction). For binary classification we'll choose different values for the ratio of $C(1,2)$ to $C(2,1)$, choosing the ratio that maximizes the AUC via a grid search.

Since random forest classifiers can predict class probabilities, we can keep p as true in the algorithm. Experiments showed setting q equal to true as well produced minor, yet statistically insignificant improvements in model performance.

Moreover, as we've already shown the random forest classifier is produced from building decision trees on bootstrap resamples of the training data, it's not necessary to perform the bootstrapping again in the MetaCost algorithm.

2.4 AUC Maximization on a Linear Predictive Model

The theory outlined here follows and is necessary not only to understand the SPAUC algorithm in 2.4.1, but also our application of the ADAM algorithm in 2.4.2 to the problem of online AUC maximization. The approach described here is a summary of [3]. More details and motivations can be found by referring to that source.

Consider the problem of fitting a linear decision function

$$h_{\mathbf{w}}(x) = \mathbf{w}^T x \tag{1}$$

to the training data while choosing $\mathbf{w}$ with the intention of maximizing the AUC, given by

$$AUC(\mathbf{w}) = \mathbb{E}(\mathbb{I}_{\mathbf{w}^T x > \mathbf{w}^T x'} | y = 1, y' = -1) \tag{2}$$

or equivalently

$$\min_{\mathbf{w}} f(\mathbf{w}) + \Omega(\mathbf{w}) \tag{3}$$

where $\Omega(\mathbf{w})$ is a convex regularizer and $f(\mathbf{w})$ is given by[2]

$$\begin{aligned} p(1-p)(1 - AUC(\mathbf{w})) \leq f(\mathbf{w}) &= p(1-p)\mathbb{E}(1 - \mathbf{w}^T(x - x')^2 \mathbb{I}_{[y=1,y'=-1]}) \\ &= \mathbb{E}(1 - \mathbf{w}^T(x - x')^2 \mathbb{I}_{[y=1,y'=-1]}) \end{aligned} \tag{4}$$

Here multiplying by $p(1-p)$ gives a more convenient formulation of the problem by removing the conditional expectation in (2).

We can now formulate (3) as a saddle point problem

$$\min_{\mathbf{w},a,b} \max_{\alpha} \mathbb{E}_z(F(\mathbf{w}, a, b, \alpha; z)) + \Omega(\mathbf{w}) \tag{5}$$

where

$$\begin{aligned} F(\mathbf{w}, a, b, \alpha; z) =& p(1-p) + (1-p)(\mathbf{w}^T x - a)^2 \mathbb{I}_{y=1} + p(w^T x - b)^2 \mathbb{I}_{y=-1} \\ &+ 2(1+\alpha)\mathbf{w}^T x (p\mathbb{I}_{y=-1} - (1-p)\mathbb{I}_{y=1}) - p(1-p)\alpha^2 \end{aligned} \tag{6}$$

This formulation allows the application of stochastic gradient descent methods on the primal variables and dual variable α. Closed form solutions of a, b, and α exist in terms of $\mathbf{w}$.

[2]Here, the square loss is used

2.4.1 Online AUC Maximization with SPAUC

Here we continue by summarizing the construction of the Stochastic Proximal AUC (SPAUC) algorithm in [3]. Building off the saddle point formulation in (6), let

$$\begin{aligned}\tilde{F}(\mathbf{w};z) =& p(1-p) + (1-p)(\mathbf{w}^T(x - \mathbb{E}[x'|y'=1]))^2\mathbb{I}_{y=1} \\ &+ p(\mathbf{w}^T(x - \mathbb{E}[x'|y'=-1]))^2\mathbb{I}_{y=-1} + 2p(1-p)\mathbf{w}^T(\mathbb{E}[x'|y'=-1] \\ &- \mathbb{E}[x'|y'=1]) + p(1-p)(\mathbf{w}^T\mathbb{E}[x'|y'=-1] - \mathbb{E}[x'|y'=1]))^2\end{aligned} \tag{7}$$

Importantly it can be shown that for any $\mathbf{w}$

$$\begin{aligned}\mathbb{E}(\tilde{F}(\mathbf{w};z) + \Omega(\mathbf{w})) &= f(\mathbf{w}) + \Omega(\mathbf{w}) \\ \mathbb{E}(\tilde{F}'(\mathbf{w};z) + \Omega'(\mathbf{w})) &= \nabla f(\mathbf{w}) + \Omega'(\mathbf{w})\end{aligned} \tag{8}$$

so that we now have unbiased estimators for the function and it's gradient we wish to minimize. We can now provide an overview of the SPAUC algorithm, which is built in the context of online streaming data. New data can be continuously streamed into the model with $\mathbf{w}$updated accordingly. To begin, we can approximate p and the conditional expectations in (7) at any time, t as follows:

$$\begin{aligned}p_t &= \frac{\sum_{i=0}^{t-1}\mathbb{I}_{[y_i=1]}}{t} \\ u_t &= \frac{\sum_{i=0}^{t-1}x_i\mathbb{I}_{[y_i=1]}}{\sum_{i=0}^{t-1}\mathbb{I}_{[y_i=1]}} \\ v_t &= \frac{\sum_{i=0}^{t-1}x_i\mathbb{I}_{[y_i=-1]}}{\sum_{i=0}^{t-1}\mathbb{I}_{[y_i=-1]}}\end{aligned} \tag{9}$$

and use these to build the following realizations of $\tilde{F}(\mathbf{w};z)$ and $\tilde{F}'(\mathbf{w};z)$ at time t.

$$\begin{aligned}\tilde{F}_t(\mathbf{w};z) =& (1-p_t)(\mathbf{w}^T(x-u_t))^2\mathbb{I}_{y=1} + p_t(\mathbf{w}^T(x-v_t))^2\mathbb{I}_{y=-1} \\ &+ 2p_t(1-p_t)\mathbf{w}^T(v_t-u_t) + p_t(1-p_t)(w^T(v_t-u_t))^2 + p_t(1-p_t)\end{aligned} \tag{10}$$

$$\begin{aligned}\tilde{F}_t'(\mathbf{w};z) =& 2(1-p_t)(x-u_t)(x-u_t)^T\mathbf{w}\mathbb{I}_{y=1} + 2p_t(x-v_t)(x-v_t)^T\mathbf{w}\mathbb{I}_{y=-1} \\ &+ 2p_t(1-p_t)(v_t-u_t) + 2p_t(1-p_t)(v_t-u_t)(v_t-u_t)^T\mathbf{w}\end{aligned} \tag{11}$$

The SPAUC algorithm now works as follows:

1. **Inputs**:
   - T : number of iterations
   - η_t: sequence of step sizes
   - Ω: convex regularizer
   - $\mathbf{w}_1$: initial value of $\mathbf{w}$(typically 0)
2. **Algorithm**:

   Initialize $n_+ = n_- = s_+ = s_- = 0$.

For $t = 1, ..., T$:
$n_+ = n_+ + \mathbb{I}_{y_t=1}$
$n_- = n_- + \mathbb{I}_{y_t=-1}$
$s_+ = s_+ + x_t \mathbb{I}_{y_t=1}$
$s_- = s_- + x_t \mathbb{I}_{y_t=-1}$
$u_t = \frac{s_+}{n_+}$
$v_t = \frac{s_-}{n_-}$

Calculate the gradient, $\tilde{F}'_t(\mathbf{w}_t; z_t)$ using (11), choosing z_t randomly, then find w_{t+1} according to

$$\mathbf{w}_{t+1} = \text{argmin}_{\mathbf{w}} \eta_t < \mathbf{w} - \mathbf{w}_t, \hat{F}'_t(\mathbf{w}_t; z_t) > + \eta_t \Omega(\mathbf{w}) + \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 \tag{12}$$

Using $\Omega(\mathbf{w}) = \lambda\|\mathbf{w}\|^2$, (12) can be solved analytically as

$$\mathbf{w}_{t+1} = \frac{\mathbf{w}_t - \eta_t \hat{F}'_t(\mathbf{w}_t; z_t)}{1 + 2\lambda\eta_t} \tag{13}$$

The questions still remains as to how to select the sequence of step sizes.

Self Bounding Property[3]: Suppose our regularizer satisfies

$$\|\Omega'(\mathbf{w})\|_2^2 \leq A_1 \Omega(\mathbf{w}) + A_2 \tag{14}$$

and define $C_1 = \max(A_1, 16k^2)$ where $k = \max\{1, \|x\|_2\}$. Here x is drawn from the feature space of the data.

Fast convergence of SPAUC with high probability[4] can be achieved setting

$$\begin{aligned} \eta_1 &= \frac{1}{2C_1} \\ \eta_t &= \eta_1 t^{-\theta} \text{ for } \theta > \frac{1}{2} \end{aligned} \tag{15}$$

2.4.2 ADAM

Starting with a stochastic objective function, $f(\mathbf{w})$, with realization $f_t(\mathbf{w})$ at time t, ADAM [4] is a stochastic gradient descent algorithm that minimizes $\mathbb{E}(f(\mathbf{w}))$ with respect to the parameter $\mathbf{w}$. Referring to (8), we can use ADAM with the objective function $\tilde{F}(\mathbf{w}; z) + \Omega(\mathbf{w})$ to solve the minimization problem in (3). The realizations at time t are given in (10) and (11), but the regularizer and its derivative now need to be included in these realizations. The algorithm works as follows.

1. **Inputs**:
    - α : step size (default .001)

[3] for $\Omega(\mathbf{w}) = \lambda\|\mathbf{w}\|^2$ the self bounding property holds with $A_1 = 4\lambda$ and $A_2 = 0$.

[4] Specfically, SPAUC converges with probability $1 - \delta$ in $O(T^{\theta-1} \log^{\frac{3}{2}} \frac{T}{\delta})$

- β_1, β_2: decay rates for moment estimates (defaults are 0.9 and 0.999)
- ϵ: (default 10^{-8})
- $f(\mathbf{w})$: stochastic objective function
- $\mathbf{w}_0$: initial value of $\mathbf{w}$(typically 0)
- T: number of iterations

2. **Algorithm**:

   Initialize $m_0 = v_0 = t = 0$.

   For $t = 0, ..., T$:
   $t = t + 1$
   $g_t = \nabla f_t(\mathbf{w}_{t-1})$
   $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
   $v_t = \beta_w \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
   $m_t = m_t/(1 - \beta_1^t)$
   $v_t = v_t/(1 - \beta_2^t)$
   $\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \cdot m_t/(\sqrt{v_t} + \epsilon)$

   return($\mathbf{w}_T$)

3 Experimental Evaluation

This section contains information on the datasets studied as well as the experimental methods and results.

3.1 Datasets

The following datasets were studied using the algorithms in section 2.

Data Set	# of instances	# of features	Class Imbalance
Kyphosis	81	3	0.21
Loan Data	9578	13	0.16
German	1000	24	0.30
Diabetes	768	8	0.35

The German and Diabetes datasets were obtained from the UCI repository, but the class imbalance for these datasets is not terribly severe. The were chosen as they were smaller datasets that were studied in [3], that we could attempt to reproduce results on. Descriptions of these datasets can be found in the UCI repository.

The Kyphosis[5] and Loan Data[6] sets were obtained from Kaggle and Lendingclub.com, respectively, and have more severe class imbalances. The author had some experience running classifiers on these datasets and wanted to continue studying them using the approaches in section 2. The Kyphosis dataset contains kyphosis as the target feature (an abnormal curvature of the spine) while the loan data contains information on borrowers, with the target feature being whether or not they default on their loan.

3.2 Experimental Methodology

3.2.1 Experimental Goals

The goals of this project and the experiments are listed below.

1. Code the MetaCost, SPAUC, and ADAM algorithms from scratch in Python, and upload to Github. The random forest and SMOTE algorithms were carried out using the sklearn library in Python.
2. Compare the AUC on testing data of the random forest classifier (with and without MetaCost and SMOTE) to SPAUC and ADAM on the four datasets in section 3.1
3. Compare the run time of the aforementioned models on the datasets in 3.1
4. Infer the best model(s) to use on our large, highly imbalanced dataset of (non)fraudulent credit card transactions.

3.2.2 Program Outline:

The following program was carried out on each dataset.

1. **Random Forest**:
    - Split data into 70% training and 30% testing while preserving the class imbalance in each set. The number of splits depended on the size of the dataset and is reported in 3.3.
    - Fit a random forest classifier on the training data using 100 trees, entropy as the splitting criterion, and splitting until all nodes are pure (all instances in each node are of the same class). Record the time needed to fit the classifier.
    - Use the fitted random forest classifier to predict the probability each instance in the testing data belongs to class 1.
    - Use these probabilities to compute the AUC. Report the mean and standard deviation of the AUC across all splits of the dataset into a training and test set. Do the same for the run time.

[5]data can be found at https://www.kaggle.com/abbasit/kyphosis-dataset
[6]data can be found at https://www.lendingclub.com/investing/peer-to-peer

2. **Random Forest with SMOTE**:

   - The program followed here was the same as for a regular random forest classifier, with the following exception. After splitting the data in training and test sets, we rectified the class imbalance on the training data using the SMOTE algorithm with 5 neighbors.

3. **Random Forest with MetaCost**:

   - First it's necessary to perform cross-validation on the cost matrix. We used a 2x2 cost matrix with $C(1,1) = C(2,2) = 0$, $C(1,2) = r$, and $C(2,1) = 1$. We then split the dataset into 70% and 30% testing several times (the number of splits depended on the size of the dataset) and fit a random forest classifier using the MetaCost algorithm ($m = 1$, $n = \#$ of instances in dataset, $p = q =$ True), on the training data selecting r from the sequence $\{0.1n\}_{n=1}^{100}$. We then ran each of these random forest classifiers on the testing data and stored the r that produced the highest AUC score. Finally, the optimal r was decided by averaging the best value of r across all splits of the dataset into training and test data.
   - We then followed the same approach as in item 1, but using the MetaCost algorithm with the aforementioned parameters and optimal value of r in the cost matrix.

4. **SPAUC**:

   - Data preprocessing was necessary to obtain good results with SPAUC. All features were standardized with mean 0 and standard deviation 1.
   - The SPAUC algorithm was used with a convex regularizer of $\Omega(\mathbf{w}) = \lambda\|\mathbf{w}\|^2$ and step size described in (15). This requires cross-validation of λ and θ. λ was chosen from $\{0, 10^n : n = -3, ...4\}$ while θ was chosen from the set $\{0.5, .75, 1\}$.
   - Cross-validation was performed as follows. The dataset was split into 70% training and 30% testing a number of times (anywhere from 3 to 50 depending on dataset size). On each split we ran the SPAUC algorithm, allowing it to run through the dataset several times. After each pass through the dataset we recorded the AUC score of the model and stopped once the improvement to the AUC score was less than .001. We then recorded the AUC score after applying our fitted model to the test data. This was done for each possible combination of λ and θ mentioned above. The test AUC was then averaged over all splits of the data and combinations of λ and θ, and the best combination was selected.
   - After selecting the best λ and θ, we again split the dataset in 70% training and 30% testing several times and ran the SPAUC algorithm on the training data. After each pass through the training data, we recorded the AUC score of our model on the training data, stopping once the improvement was less than .001, then recorded the run time. We then applied our model to the test data and recorded the AUC. The mean and standard deviation of the run times and testing AUC were then recorded.

5. **ADAM**:

   - Our program with the ADAM algorithm was nearly identical to the approach used for SPAUC. We used the default parameters for the model listed in section 2.4.2 with one notable exception. The default step size of $\alpha = .001$ was too large so the algorithm would not converge as it kept missing the maximum AUC. Instead we used the same step size in (15) which required cross-validation of θ in addition to λ.

4 Critical Analysis of Experimental Results

Here we discuss the results of program described in section 3.2.2.

4.1 AUC Score on Test Data

The table below contains information on the AUC scores produced by each model on each dataset. The scores in bold represent the statistically significant best scores on each dataset using a Welch t-test with $\alpha = .05$.[7] The tuples report the following information:

(Mean AUC Score on Test Data, Standard Deviation of AUC Score, # of Train, Test, Splits)

Data	RF	RF w/ SMOTE	RF w/ MetaCost	SPAUC	ADAM
Kyphosis	(.81, .084, 150)	(**.845**, .078, 150)	(.811, .089,100)	(.826, .070, 150)	(**.830**, .085, 150)
Loan Data	(.652,.010,30)	(.631,.011,30)	(.661,.010,10)	(**.670**,.014,10)	(**.669**,.014,10)
German	(**.792**, .021, 30)	(**.783**, .027, 30)	(**.783**, .024, 30)	(**.794,** .02,10)	(**.786**, .021, 10)
Diabetes	(**.827**, .027,30)	(**.815**, .024,30)	(**.821**,.020, 30)	(**.83**, .023, 10)	(**.813**,.030, 10)

The AUC scores produced by SPAUC are nearly identical to those reported on the German and Diabetes datasets in [3]. Across datasets, we see little variation in the performance of the methods. The Random forest classifier performed better without using SMOTE or MetaCost on every dataset except Kyphosis, where the improvement from SMOTE actually was statistically significant ($p = .0002$). Across the other three datasets SPAUC performed the best although the improvement was not always statistically significant.

4.2 Run Time

Below we list statistics on the run time needed to fit each of the models across datasets. The tuple includes the mean run time and standard deviation. The number of splits can be found in the table in section 4.1. Statistically significant best run times were always achieved by the random forest classifier, with the exception of the Kyphosis dataset. Here, the dataset was so small that the SMOTE algorithm produced only a negligible increase in run time.

Data	RF	RF w/ SMOTE	RF w/ MetaCost	SPAUC	ADAM
Kyphosis	(.096,.006)	(.099,.010)	(.202, .012,)	(.622,.217)	(.968,.372)
Loan Data	(1.16, .038)	(2.243,.087)	(2.57,.039)	(126.090, 43.661)	(90.837,2.328)
German	(.148, .011)	(.170,.010)	(.310, .015)	(27.605,8.494)	(21.069,6.608)
Diabetes	(.147, .008)	(.164, .009)	(.292, .014)	(8.949, 2.349)	(9.640, 2.199)

[7] Here, if a score is not in bold, it was shown to be significantly lower than the highest AUC on the particular dataset.

As expected the run time for the random forest increases when using SMOTE and MetaCost. But the run time for SPAUC and ADAM is considerably larger than the random forest classifier, even if only iterating through the dataset once. Note, though, that the run times listed here for SPAUC and ADAM included several passes through the data, however many were needed to obtain model convergence (typically between three and ten passes).

Given that SPAUC typically only produced modest or statistically insignificant improvements over the random forest classifier, it may be best to use the random forest classifier on our credit card data when accounting for run time.

4.3 Cross-Validated Parameters

This section lists the parameters used in the random forest with MetaCost model, as well the SPAUC and ADAM algorithms, found through cross-validation[8].

Data	RF w/ MetaCost (mean(r), sd(r))	SPAUC (λ, θ)	ADAM (λ, θ)
Kyphosis	(5.74, 2.97)	(1000, 0.5)	(.0001, 0.5)
Loan Data	(8.1, 2.550)	(10, 0.5)	(10, 0.5)
German	(1.56, .803)	(100, .75)	(100, 1)
Diabetes	(5.4, 3.633)	(0, 0.5)	(1000, 1)

4.4 Credit Card Fraud

Having studied these four data sets we'll turn our attention to the dataset that motivated this project. Here we'll study a large, highly imbalanced dataset of credit card transactions[9] to be classified as (non)fraudulent. The dataset contains 284, 807 instances with 30 features. Only 492 instances are fraudulent (class imbalance is .0017).

Based on the results in sections 4.1 and 4.2 we expect a random forest classifier to produce good comparative results in the fastest time. We won't bother using the random forest with MetaCost algorithm here as it never produced results that were better than other models. In addition, cross validation of the cost ratio would be very time consuming.

The results of the random forest with SMOTE algorithm on the Kyphosis dataset were intriguing so we'll include that model in our analysis as well. Also, given that SMOTE is an algorithm designed for imbalanced data, and our credit card data is more imbalanced than the previous datasets studied, we could get better results. The increase in run time over the random forest classifier is expected to be more substantial here, as the SMOTE algorithm will nearly double the size of the dataset given how severe the class imbalance is.

Lastly, the SPAUC and ADAM algorithms produced comparable AUC test scores (neither ever did significantly better than the other) with comparable run times so we'll only use the SPAUC algorthm

[8]Due to time constraints, cross-validation of λ and θ on the loan data was only done with the SPAUC algorithm. The same parameters were then used with ADAM.

[9]This dataset can be found at https://www.kaggle.com/mlg-ulb/creditcardfraud

here. Cross validation of the parameters λ and θ is extremely time consuming, though, so we'll use $\lambda = 0$ and $\theta = 0.5$. The motivation for this is as follows: on the more imbalanced Kyphosis and Loan Data datasets, $\theta = 0.5$ was the best parameter. In addition, this parameter creates larger initial step sizes, allowing more movement of $\mathbf{w}$in the initial iterations of the algorithm. While regularization with λ produced better results on every dataset, the improvement in the AUC score from introducing the regularization term during cross-validation was frequently negligible. In fact, the results achieved on the German and Diabetes datasets in [3] were nearly identical to ours despite the authors not using a regularization term.

4.4.1 Experimental Results

The table below summarizes our experimental results on the credit card data set for three splits of the dataset into training and test sets. Statistically significant best AUC scores are in bold, using a Welch t-test.

Metric	Random Forest	Random Forest w/ SMOTE	SPAUC ($\lambda = 0$, $\theta = 0.5$)
Test AUC	(.943, .0058)	(**.973**,.0090)	(**.971**, .0098)
Run Time	(100.67, 1.76)	(215.65,2.85)	(4126.38,653.71)

Here the random forest did not perform nearly as well as SPAUC and the random forest with SMOTE. There was no significant difference in model performance between the random forest with SMOTE and the SPAUC algorithm, although as expected the run time on SPAUC is considerably larger. From a practical standpoint, the author was able to obtain the results of random forest with SMOTE while preparing a simple breakfast, while the SPAUC algorithm needed to run for several hours..

5 Conclusion and Further Work

Having compared the performance of the various classifiers across datasets, its clear that stochastic gradient descent methods typically provide AUC scores that are at least as good as that of a basic random forest classifier. However, in situations where gradient descent methods outperform the random forest classifier, modifying the random forest classifier with SMOTE or a cost-sensitive approach can make up for the difference in model performance, while still keeping run time well below the run times of stochastic gradient descent methods. However, the advantage of the stochastic gradient descent methods described in this project is that they can continually update in the presence of new data, making them more relevant in online applications where data is continuously streamed into the model.

The results of the stochastic gradient descent algorithms used in this paper are intriguing, as they were used on a simple linear predictive model while random forests are non-linear. Even better results may be achievable by either kernelizing the decision function in (1), or by replacing it with a non-linear score function parametrized by $\mathbf{w}$. Early progress on the latter approach is described in [6] using deep neural networks.

References

[1] A. Fernandex, "SMOTE for Learning from Imbalanced Data: Progress and Challenges: Marking the 15-year Anniversary" in *Journal of Artificial Intelligence Research*, vol. 61, 2018, pp. 863-905. Accessed: May 9, 2021. [Online] Available: https://www3.nd.edu/ dial/publications/fernandez2018smote.pdf

[2] P.Domingos, "MetaCost: A general method for making classifiers cost sensitive," in *Proceedings of the Fifth Iternational Conference on Knowledge Discovery and Data Mining,* ACM Press, 1999, pp. 155-164. Accessed: May 9, 2021. [Online] Available: https://weber.itn.liu.se/ aidvi/courses/06/dm/papers/MetaCost.pdf

[3] Y. Lei, "Stochastic Proximal AUC Maximization," *Journal of Machine Learning Research*, vol. 22, 2021, pp. 1-45. Accessed: May 9, 2021. [Online] Available: https://jmlr.csail.mit.edu/papers/volume22/19-418/19-418.pdf

[4] D.P. Kingma, "Adam: A Method for Stochastic Optimization," *3rd International Conference for Learning Presentations*, San Diego, 2015. Accessed: May 9, 2021. [Online] Available: https://arxiv.org/abs/1412.6980

[5] T. Hastie, *The Elements of Statistical Learning.* 2nd ed., New York: Springer, 2013, p. 305-313, 587-588.

[6] M. Liu, "Stochastic AUC Maximization with Deep Neural Networks," ICLR, 2020. Accessed: May 11, 2021. [Online] Available: https://openreview.net/pdf?id=HJepXaVYDr