CSE 547 Project: Financial Fraud Detection

Brian de Silva
University of Washington Applied Mathematics
bdesilva@uw.edu

Daiwei He
University of Washington Mathematics
dwhe@uw.edu

Abstract

In this report we study the problem of fraud detection in a synthetic data set. Two problems common in financial data sets also plague this one. The cost of misclassifying data points is example-dependent and the ratio of positive and negative examples is highly skewed. We discuss and employ techniques for handling these issues. We conclude that the most economical solution is, perhaps unsurprisingly, to combine approaches for tackling both problems. However if computational cost is of no concern, one can achieve the best results with a decision tree or random forest modified to take the example-dependent costs into account.

1 Introduction

Financial fraud detection has been studied by researchers and industry practitioners alike for decades. As such there is a sizeable body of literature on applying machine learning and data mining techniques to this problem. [1] gives a review of the work that has been done in automated fraud detection and provides rough classifications of the methods used and the problems to which they are applied. [2] performs some the the same analyses, but at a more technical level. It also provides some criticisms of existing work and suggestions of how their approaches might be improved. [3] compares the performance of six of the most popular machine learning techniques that are used for fraud detection. It also identifies features that are given large weight by many of these algorithms.

There are properties of financial data that make fraud detection an especially difficult problem from a machine learning perspective, especially in the real-world setting where one obtains data in a streaming fashion. The principal among these issues is the extreme skewness of class distributions. In most data sets there are many more examples of legitimate transactions than fraudulent ones. Classic machine learning algorithms tend to have trouble dealing with this imbalance. Many models trained on such data sets will become very good at identifying legal transactions, but cannot accurately classify illegitimate ones. Furthermore, if the data is coming from a stream, then the proportion of fraudent cases to legal ones will vary over time. Multiple styles of fraud can be present within the same data set, each with their own characteristics. A good model will be expected to possess the ability to detect all these types of fraud. Fraudsters will attempt to make their illegal transactions look as similar to legitimate ones as they can. They will *intentionally* try to make it hard for our models to distinguish between their transactions and legal ones. As time passes new fraudsters will arise, old ones will retire, and some will adapt their fraud schemes in response to the state-of-the-art detection techniques. Finally the financial costs in misclassifying different transactions is example-dependent, e.g. classifying a fraudulent transaction as a legitimate one may result in a larger loss in revenue to a financial institution than mistakenly classifying a legitimate transaction as a fraudulent one.

For this project we attempt to tackle only the issues of imbalanced class distributions and the example-dependent cost of missclassification. The remainder of this report is organized as follows. In Section 2 we describe the data set used for our project. We next give some reasonable metrics by which to measure classifier performance in Section 3. Sections 4 and 5 present techniques for dealing with example-dependent costs and imbalanced classes. Section 6 provides a brief overview

of the machine learning algorithms we used. Finally, in Sections 7 and 8 we summarize our findings and give some concluding remarks.

2 Data Set

Our data set comes from a Kaggle problem[1] wherein a program called PaySim simulates mobile monetary transactions based on a large sample of financial data from a country in Africa. We have 6,362,620 examples. This data set is comprised of a binary target variable, "isFraud," and 10 input features, of which three (i.e. a temporal feature "step" representing the date of the transaction, and categorical features, "nameOrig" and "nameDest") are removed since they do not aid in prediction. Of the different transaction types–`CASH_OUT`, `PAYMENT`, `CASH_IN`, `TRANSFER`, and `DEBIT`–only `TRANSFER` and `CASH_OUT` were found to have any instances of fraud, so we removed transactions of the other types. This left us with 2,770,393 examples, 8197 ($\sim$ 0.3%) of which are fraudulent. Each of these examples has eight features: `amount`, `oldbalanceOrig`, `newbalanceOrig`, `oldbalanceDest`, `newbalanceDest`, `isFlaggedFraud`, `transfer`, and `cash_out`. The first five describe the transaction amounts and the balances in the accounts on either side of the transactions before and after the transactions take place. `isFlaggedFraud` is a binary variable which is active if the existing system flagged the transaction as being fraudulent. Only 16 transactions have a nonzero entry in this feature. `transfer` and `cash_out` are binary variables which indicate if the transaction is of the `TRANSFER` or `CASH_OUT` type. The features can be shifted and rescaled to have mean 0 and standard deviation 1 and the examples can be normalized, but we found neither of these provided any benefits. We set aside a random third of the data set for testing and train on the remaining two-thirds.

3 Metrics

Let TP, TN, FP, and FN denote the number of true positives, true negatives, false positives, and false negatives in a given method's predictions, respectively. There are many ways we could measure the performance of classifiers applied to our data set. We present some reasonable choices below.

- **Accuracy** - the misclassification rate, namely $\frac{\text{FN+FP}}{\text{FN+FP+TN+TP}}$. Clearly accuracy cannot give us a complete picture of a classifier's performance since FN $\gg$ FP for fraud detection. Even the "dummy" classifier which labels every transaction as legitimate has an accuracy of 0.997 on our test set.
- **Recall** - the portion of fraudulent cases a classifier is able to catch, namely $\frac{\text{TP}}{\text{TP+FN}}$. This is our best "out-of-the-box" measure of performance since false negatives are much more costly than false positives in this domain. It is not a perfect metric, however, an increase in recall often results in an increase in FP. Notice that a classifier marking everything as fraud will give us a recall of 1 but a low accuracy. Hence considering only recall may be problematic.
- **Precision** - the fraction of cases flagged as fraudulent which actually were, namely $\frac{\text{TP}}{\text{TP+FP}}$. Given that a transaction is flagged as being fraudulent, this metric gives an idea of the likelihood that it actually is. Precision's drawback is that it does not take into account FN.
- **F1-score** - gives information about both recall and precision simultaneously. The definition is $\frac{2\times\text{precison}\times\text{recall}}{\text{recall+precision}}$, i.e. the harmonic mean of the precision and recall. A drawback of the F1-score is that it gives the precision and recall scores equal weight when recall can actually be more important in the fraud detection setting.
- **Cost** - approximation of cost incurred by financial instituion as a result of a method's classifications. We assume there is some constant overhead cost associated with investigating a transaction marked as fraudulent, call it C_a. We assume the cost of missing a fraudulent transaction is equal to the amount of the transaction. Finally we assume that if a transaction of an amount less than C_a is flagged as being fraudulent the financial institution simply ignores it since the cost of investigating it will outweigh the potential savings. Then the costs associated with each classification are given in the following table:

[1] `https://www.kaggle.com/ntnu-testimon/paysim1`

	Actual negative	**Actual positive**
Predicted negative	0	Amt_i
Predicted positive	$\mathbb{1}_{\{\text{Amt}_i > C_a\}} \times C_a$	$\min\{C_a, \text{Amt}_i\}$

Table 1: Cost of different classificiations of the i-th transaction

where Amt_i is the amout of the ith transaction and $\mathbb{1}_E$ is 1 if E occurs and is 0 otherwise. Note that the overall cost of a fixed set of transactions will vary as C_a is changed.

- **Savings** - one way to measure the performance of a classifier is to compare its cost against that of the classifier which always predicts that transactions are legitimate. Define the *savings* of a classifier as the fraction of cost recouped when the classifier is used compared to when it is not used, i.e.

$$\textbf{Savings} = \frac{\text{Cost with no classifier} - \text{Cost using classifier}}{\text{Cost with no classifier}}.$$

It should be noted that it is impossible for a method to earn a savings score of 1 since there is still some nonzero cost that comes with correctly identifying a fraudulent transaction. For the test set used in this report the best possible savings score is about 0.94.

4 Example-dependent costs

Since it is often much more expensive to misclassify a fraudulent transaction than a legal one, cost-sensitive methods will often automatically place extra emphasis on correctly identifying examples in the minority class. As a result, these methods can assuage some of the problems caused by class imbalance. In fact, in some cases cost-sensitive techniques have proven to be better solutions to these issues than the sampling methods designed for imbalanced learning problems [4].

We use two techniques for modifying our classifications to take financial cost into account. The first is fairly straightforward: force the models to use these costs during training. For example, in the training process for a decision tree, the quality of a split is typically measured using some cost-insensitive criteria such as the Gini impurity or information gain. One can design a cost-sensitive decision tree by using a criterion that is a function of the cost of the resulting split, e.g. the direct cost of the classifications resulting from the possible splits. The resulting decision tree may not achieve as high an accuracy score as one trained in the standard way, but it can make classifications which lead to greater savings.

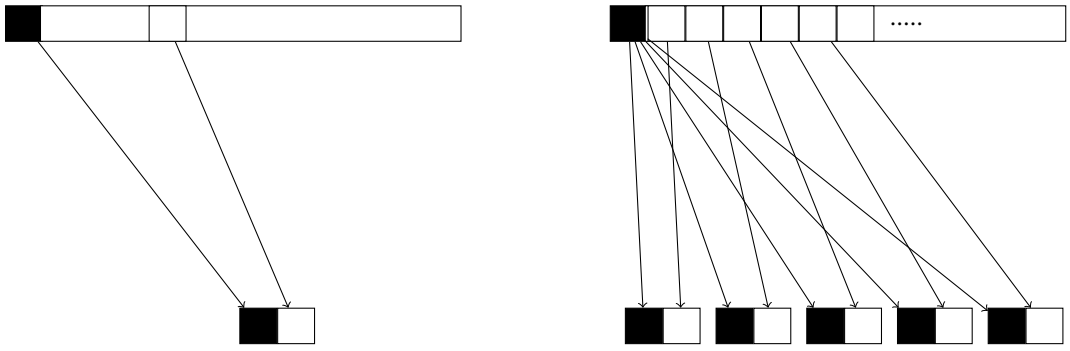
Many classifiers used for machine learning are capable of providing probabilities associated with their classifications. These probabilities give an indication of how certain the classifier is about the labels it has assigned its examples. Given the cost matrix in Table 1, one can use these probabilities to estimate the expected cost of different possible classifications. Given an example to be classified, a typical model will compute the probabilities that the example is a member of each possible class and assign it to the one with the highest probability. The Bayes Minimum Risk framework simply says to instead label examples based on the expected costs of the possible classifications. More concretely, let $c_i \in \{0, 1\}$ be the class of example x_i and let $\hat{p}_i$ be the probability output by the model that $c_i = 1$. Here $c_i = 1$ if x_i is a fraudulent transaction. Furthermore, let C_{TN_i}, C_{FN_i}, C_{TP_i}, and C_{FP_i} be the costs of x_i being a true negative, false negative, true positive, and false positive, respectively. Then the *risk* associated with classifying x_i as being a legal transaction is

$$R(c_i = 0|x_i) = \mathbb{E}(c_i = 0|x_i) = C_{TN_i}(1 - \hat{p}_i) + C_{FN_i} \cdot \hat{p}_i, \tag{1}$$

and the risk for classifying x_i as being fraudulent is

$$R(c_i = 1|x_i) = \mathbb{E}(c_i = 1|x_i) = C_{TP_i} \cdot \hat{p}_i + C_{FP_i}(1 - \hat{p}_i). \tag{2}$$

Note that (1) and (2) are the expected costs of the two possible labelings of x_i. A Bayes Minimum Risk classifier labels x_i as being a legal transaction if $R(c_i = 0|x_i) < R(c_i = 1|x_i)$, and labels it as fraudulent otherwise, i.e. it selects the lower-risk classification. In some cases this decision will differ from that of the base classifier. For example, suppose there is some example, x_i, with a sizeable associated transaction amount which the base method finds hard to classify, i.e. $\hat{p}_i \approx 1 - \hat{p}_i$. Then the risk associated with classifying x_i as legitimate will be much higher than the risk of labeling



Undersampling on training data

Meta-learner distribution of training data

Figure 1: Left: the undersampling procedure we use to create a balanced training set. Right: a visual representation of the undersampling procedure we use to construct meta-learners.

it as fraudulent since $C_{FN_i} = Amt_i$ is presumably large. It is much better, from the perspective of financial cost, to mark x_i as fraudulent, despite the fact that the model is unsure about its class membership. The Bayes Minimum Risk framework takes this imbalance into account and acts accordingly, classifying x_i in the most cost-effective way.

In order to achieve the best possible performance using a Bayes Minimum Risk classifier it is important to calibrate the probabilities output by base models in a particular way. See [5] for details about how to accomplish this. For more techniques for dealing with example-dependent costs see [6] and [4].

5 Class imbalance

As was mentioned previously, extreme class imbalance can be detrimental to classifier performance. [6] and [4] present some general techniques for working with data sets with skewed class distributions. [7] introduces some specific algorithms that can be used with such data sets.

For clear presentation, we establish here some of the notations used in this section. Denote by S the original training set. Furthermore, we define subsets $S_{frad} \subset S$ and $S_{legt} \subset S$, where S_{frad} are the fraudulent examples in S and S_{legt} are the legitimate examples in S.

The first method we use to deal with the imbalance of training data is a form of undersampling, namely we remove data from original training set in order to balance the class distribution. In particular, we randomly select a set of legitimate transactions S'_{legt} from S_{legt} so that the new training set is constructed as $S_{new} = S_{frad} \cup S'_{legt}$. Consequently, undersampling give a way to adjust the balance of the original training data S. [6] reveals that a proportion of $|S'_{legt}| : |S_{frad}| = 1 : 1$ is a reasonable choice for financial fraud problem with customized cost as an optimization objective. Our construction of the new balanced training set also use a $|S'_{legt}| : |S_{frad}| = 1 : 1$ undersampling.

The second method was proposed in [6]. A similar variant, called EasyEnsemble, was introduced in [7]. We apply the undersampling technique described above multiple times (without replacement for samples drawn from S_{legt}) and train classifiers on each balanced set. That is, from S_{legt} we sample k subsets S^k_{legt} with $S^j_{legt} \cap S^k_{legt} = \emptyset$ if $j \neq k$ then combine them with the fraudulent data to construct k training sets: $S_k = S_{frad} \cup S^k_{legt}$. We train models on each such set then then combine the individual classifiers into an ensemble or meta-learner. The final prediction is obtained by an average or majority vote method. Using cross-validation we found that we struck a good balance between training time and good performance using 25 learners. Figure 1 provides a visualization of the undersampling and meta-learner approaches.

In the case of undersampling, we are able to eliminate class imbalance entirely. Undersampling typically improves recall, however. It also allows models to be trained more quickly since the size of the training set is significantly smaller. Real world financial data sets have many more transactions that

ours, enough that it can be infeasible to train a model on the full set. The drawback of undersampling is obvious: we throw away a sizeable amount of legitimate transactions. This typically worsens the precision and F1-scores. It can also increase the variance in our models since the way we select the legitimate data can significantly influence the resulting model. The meta-learning approach can partially alleviate the problem of wasting data since it uses more of the legitimate transactions. It shares the same advantages as naive undersampling, and decreases variance in the models it prodcues. The structure of EasyEnsemble makes it amenable to parallel computing since each member of the ensemble can be trained separately. However, prediction takes longer because the votes of all the constituent classifiers must be tallied. Furthermore, if the classifiers are not combined in a smart way, the performance of the overall method can be worse than many of its members.

6 Methods

Multiple sources we read asserted that a variety of classifiers can achieve good performance in the realm of fraud detection [1, 2, 3]. For this reason we decided to compare many methods against one another. In particular, we worked with the standard scikit-learn implementations of the following classifiers: random forest (RF), decision tree (DT), gradient boosted regression tree (GB), and logistic regression (LR)[2]. In the interest of time, we used the default parameters for each method. It is possible we could have improved the methods' scores by tuning them further. An SVM was also tested, but we found that it did quite poorly, so we omit it from this report.

We also tested a neural network (NN) implemented using the Keras package[3] with the backend handled by TensorFlow. We use accuracy as objective. Cross validation revealed that a neural network with three hidden layers, eight nodes in each layer, and with linear rectifiers as activation functions is the optimal configuration for our particular problem. Hence this is the setup we used.

Each method is capable of giving approximations of the probabilities that examples are fraudulent or legitimate, $\hat{p}_i$ and $1 - \hat{p}_i$, but we should expect that logistic regression and the neural network will give the most accurate probability estimates [3]. This may result in these methods being better base models from which to construct Bayes Minimum Risk (BMR) classifiers. We perform the probability calibrations and form the BMR classifiers using the costcla package [4].

Beyond the algorithms described above, we also test the costcla implementations of cost-sensitive decision trees (CSDT), cost-sensitive random forests (CSRF), and cost-sensitive logistic regression (CSLR).

7 Results

In order to determine a reasonable value of C_a and to we trained some standard classifiers on a subset of the training set. We then had these models predict the classes of the examples in the test set and computed the costs of these classifications as C_a was varied. The results of this experiment are shown in Figure 2. `dummy_neg` and `dummy_pos` are the classifiers which always predict that transactions are legal and fraudulent, respectively. `RF`, `LR`, `GB`, `LRW`, and `DT` are random forest, logistic regression, gradient boosted trees, weighted logistic regression, and a decision tree, respectively. For weighted logistic regression we gave the fraudulent examples 10 times as much weight as the legal examples in the log-loss function. The classifier we compare against to compute a model's savings is `dummy_neg`. Notice that for small values of C_a we can essentially afford to investigate every transaction and still save a lot of money. As the price of investigating grows, this becomes infeasible. For most of our experiments we use the value $C_a = 10^5$ since this choice appears to strike a good balance between making investigations expensive and not ruling out too many transactions because their amounts are below the threshold. The mean transaction amount is also on the order of 10^5, which provides some further justification for using this value.

Next we attempted to explore the benefits of the techniques described in Section 5. We trained each of the standard classifiers on both an undersampled version of the training data and on the

[2] `http://scikit-learn.org/stable/`

[3] `https://keras.io/`

[4] `https://pypi.python.org/pypi/costcla`

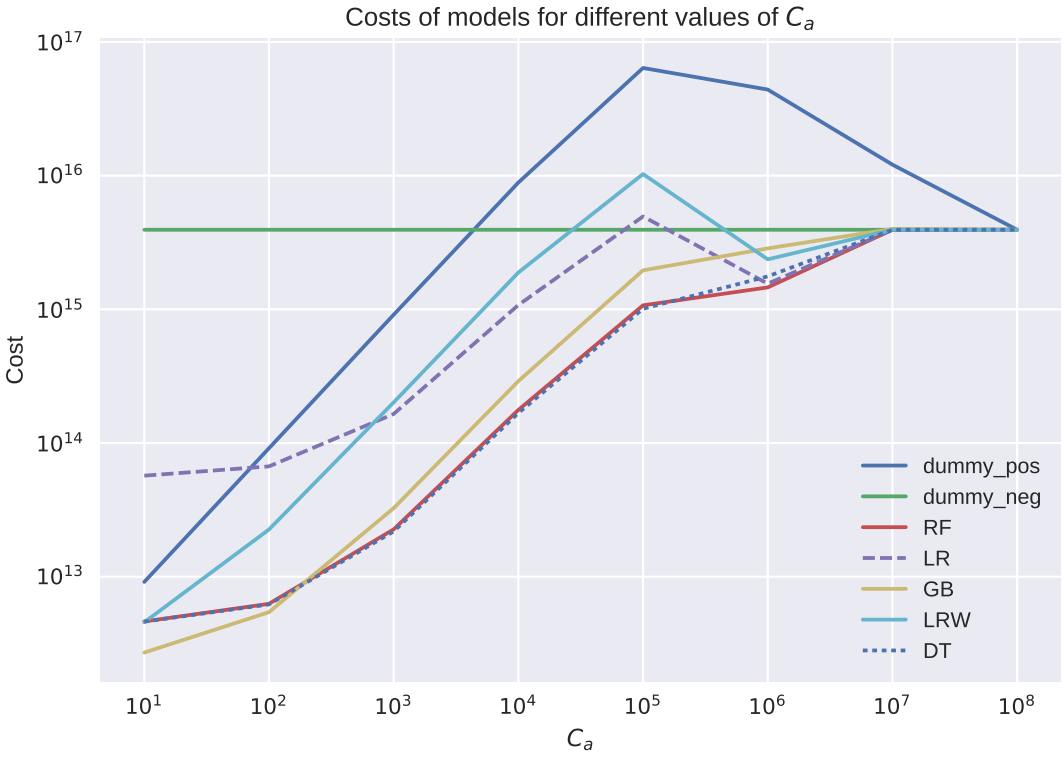


Figure 2: Costs of a fixed set of classifications as C_a is varied.

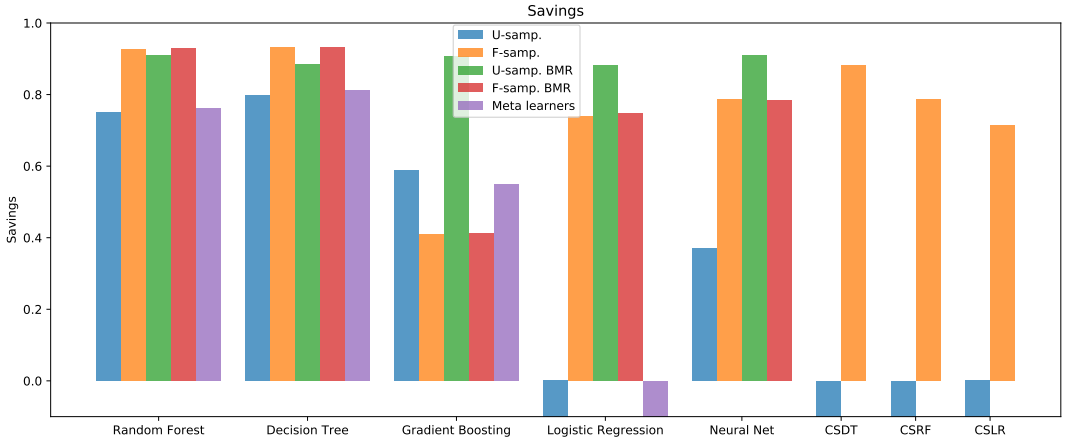


Figure 3: The savings scores for some of the models tested when $C_a = 10^5$. "U-samp." and "F-samp." refer to the models trained on the undersampled and full training sets, respectively. "BMR" denotes the Bayes Minimum Risk versions of each model and "Meta learners" the ensembles consisting of many instances of each classifier. "CSDT," "CSRF," and "CSLR" refer to the cost-sensitive decision tree, cost-sensitive random forest, and cost-sensitive logistic regression methods, respectively.

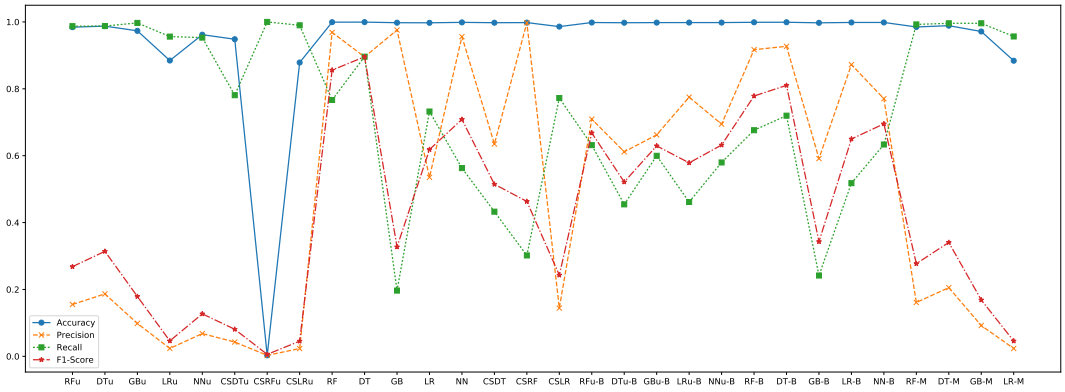


Figure 4: Metrics for some of the models tested when $C_a = 10^5$. Names with a "u" were trained on a subsampled version of the training set. Names with a "-B" are the Bayes Minimum Risk versions of the methods. Names with "-M" refer to ensemble metods. "CSDT," "CSRF," and "CSLR" refer to the cost-sensitive decision tree, cost-sensitive random forest, and cost-sensitive logistic regression methods, respectively.

full training set, then evaulated them on the test set. The savings scores earned by the methods are shown as the blue and yellow bars in Figure 3. The undersampled versions of the methods typically saved less money than their counterparts using the full training set. Undersampling did not appear to affect the random forest and decision tree methods as much as the others. The undersampled gradient boosting method actually outperformed the fully sampled one. For logistic regression undersampling resulted in classifications which actually cost us money. The other metrics for these methods are summarized in the left part of Figure 4. Notice the jump in F1-score and precision that occurs when we switch from undersampling to utilizing the full training set. Although the recall drops off at the same point, the fully-sampled methods still realize better savings. This suggests that precision and F1-score may be better proxies for savings than recall, which makes sense since there are so many more negative examples than positive ones. If we miss just a few positive examples the recall score can change considerably, while misclassifying the same number of negative examples will not elicit as large a shift in the precision and F1-scores.

Perhaps the issue with undersampling is that it throws away too much of the training data. With this in mind, we experimented with the meta-learners discussed in Section 5. We neglected to build a meta-learner consisting of individual learners which are neural nets due to the amount of time required to train such a model. The purple bars of Figure 3 plot the savings of these methods and Figure 4 plots their scores in the other metrics. The effectiveness of the meta-learners was generally slightly better than that of the vanilla undersampled methods. [7] suggests potentially better ways to construct these ensembles which may improve results further (e.g. BalanceCascade), but we did not have time to implement these.

After having trained all these base models we can turn them into Bayes Minimum Risk classifiers using the procedure outlined in Section 4. The green and red bars in Figure 3 show the savings for these BMR methods. The undersampled BMR methods perform strikingly well, given that they do not see over 99% of the training data. The undersampled BRM versions of gradient boosting, logistic regression, and the neural net surprisingly outdo their fully-sampled dopplegangers. This is especially astounding for logistic regression since the predictions from the undersampled model result in negative savings. However, modifying these predictions using the BMR framework, the method instantly becomes competitive with the more complicated algorithms. We suspect that this is because the undersampled algorithms are better able to estimate the probabilities that examples are fraudulent. This results in superior approximations of risk by the BMRs, which translates into better savings. The fully-sampled BMR models for the decision tree and random forest achieved higher savings than the undersampled ones, but it should be noted that these models took a factor of about 450 times longer to train. If this training time is taken into account, the undersampled BMR methods begin to look more attractive. Turning our attention again to Figure 4, a perhaps interesting observation is that each BMR methods tends to have precision, recall, and F1-scores which are close to each other (the F1-score always lies between the precision and recall since it is their harmonic

mean), while maintaining relatively high overall accuracy. This gives us some insight into how these models are choosing to balance the tradeoffs between accruing too many false positives and false negatives. It would appear that the better BMR methods tend to prioritize the precision over the recall somewhat.

Finally, we trained cost-sensitive versions of the decision tree, random forest, and logistic regression methods using the full training set and a subsampled version. The blue and yellow bars on the right side of Figure 3 show the savings achieved by each undersampled and fully-sampled cost-sensitive method, respectively. Undersampling is clearly detrimental to these methods. The fully-sampled variants all produce reasonable savings, but they fail to outshine the undersampled BMR methods in terms of savings. Their other metrics are shown in Figure 4. These methods took considerably longer to train than the others, leading the authors to believe that there are better approaches to driving cost down, namely the BMR methods.

8 Conclusions

In this report we introduced methods for working with imbalanced data with example-dependent costs. We applied these methods to the problem of financial fraud detection using a large data set obtained from Kaggle. Looking at the collection of results from all the methods there are a few conclusions we may draw. In terms of maximizing savings, the BMR-modified decision tree and random forest methods trained on the full data set work best. However, for a larger data set, training a model using every example can be intractable. In this domain a Bayes Minimum Risk method trained on a subsampled portion of the data set may be more desirable. These methods are faster to train and give similar, albeit slightly worse, performance compared to the models trained on the full data sets. Furthermore they help take into account both the class imbalance via undersampling and example-dependent costs via BMR. Further, the undersampling procedure actually enhances the BMR modification by improving the probability estimates of the underlying classifiers. Of the methods tested, logistic regression was the fastest to train and gave good performance, illustrating the fact that one does not always need an advanced technique to obtain decent results.

References

[1] EWT Ngai, Yong Hu, YH Wong, Yijun Chen, and Xin Sun. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, 50(3):559–569, 2011.

[2] Clifton Phua, Vincent Lee, Kate Smith, and Ross Gayler. A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*, 2010.

[3] Johan Perols. Financial statement fraud detection: An analysis of statistical and machine learning algorithms. *Auditing: A Journal of Practice & Theory*, 30(2):19–50, 2011.

[4] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[5] Alejandro Correa Bahnsen, Aleksandar Stojanovic, Djamila Aouada, and Björn Ottersten. Improving credit card fraud detection with calibrated probabilities. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 677–685. SIAM, 2014.

[6] Philip K Chan and Salvatore J Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *KDD*, volume 1998, pages 164–168, 1998.

[7] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550, 2009.