Unbalanced Credit Card Fraud Detection

Yazan Obeidi
Dept. of Systems Design Engineering
University of Waterloo
Waterloo, Canada
yazan.obeidi@uwaterloo.ca

Credit-card fraud is a growing problem worldwide which costs upwards of billions of dollars per year. Advanced classification methods provide the ability to detect fraudulent transactions without disturbance of legitimate transactions and unnecessarily spent resources on fraud forensics for financial institutions. However some key risks exist, namely imbalance learning and concept drift. In this paper, current state-of-the-art techniques for handling class imbalance at the data level and algorithm level are examined on European credit card transactions over the period of two days. The results are compared to baseline for three algorithms which have been suggested in previous research to have high performance with the task of fraud detection: linear support vector machine, random forest, and multi-layer perceptron. Results suggest that sophisticated generative sampling methods can easily fail to generalize the minority class when extreme class imbalance exists leading to worse performance over more conventional class imbalance techniques such as cost-based methods.

Keywords — unbalanced fraud detection adaptive synthetic sampling random forest support vector machine multi-layer perceptron

I. Introduction

In 2015, credit, debit and prepaid cards generated over $31 trillion in total volume worldwide with fraud losses reaching over $21 billion [1]. In that same year there were over 225 billion purchase transactions, a figure that is projected to surpass 600 billion by 2025 [2]. Fraud associated with credit, debit, and prepaid cards is a significant and growing issue for consumers, businesses, and the financial industry.

Historically, software solutions used to combat credit card fraud by issuers closely followed progress in classification, clustering and pattern recognition [3, 4]. Today, most Fraud Detection Systems (FDS) continue to use increasingly sophisticated machine learning algorithms to learn and detect fraudulent patterns in real-time, as well as offline, with minimal disturbance of genuine transactions [5].

Generally, FDS need to address several inherent challenges related to the task: extreme unbalancedness of the dataset as frauds represent only a fraction of total transactions, distributions that evolve due to changing consumer behaviour, and assessment challenges that come with real time data processing [6]. For example difficulties arise when learning from an unbalanced datasets as many machine intelligence methods are not designed to handle extremely large differences between class sizes [5]. Also dynamic trends within the data require robust algorithms with high tolerance for concept drift in legitimate consumer behaviours [7].

Although specialized techniques exist that may handle large class imbalance such as outlier detection, fuzzy inference systems and knowledge based systems [8, 9], current state-of-the-art research suggests that conventional algorithms in fact may be used with success if the data is sampled to produce equivalent class sizes [10, 11, 12]. The newest sampling techniques involve creating synthetic samples using a clustering algorithm such as k-Nearest-Neighboor (kNN). This is valuable as it means a wider range of, in some cases off-the-shelf, typical classification algorithms may be used which mitigates existing algorithmic limitations due to extreme class imbalance. Not only can fraud detection capabilities potentially increase due to a larger scope of potential methods, the cost of development can be decreased due to reduced reliance on highly specialized niche methods, expert systems, and continued research into algorithmic methods which handle class imbalance directly.

II. Material and Methods

A. Dataset

The credit card fraud dataset used in this paper is obtainable from Kaggle.com [13] and contains a subset of online European credit card transactions made in September 2013 over a period of two days, consisting of a highly imbalanced 492 frauds out of 284 807 total transactions [5, 6, 7, 14, 15].

For confidentiality the dataset is simply provided as 28 unlabeled columns resulting from a PCA transformation. Additionally, there are three labeled columns: Class, Time, and Amount.

Note that the dataset is highly imbalanced towards legitimate transactions which have the label of “0”, visible in Fig. 1.

```
data.Class.value_counts()

0    284315
1       492
Name: Class, dtype: int64
```

Fig 1: Class imbalance in he Kaggle credit-card fraud dataset

B. Methods

1) Imbalance Learning: Standard decision trees such as ID3 and C4.5 use information gain as the splitting criterion for learning which results in rules biased towards the majority [3]. Research also shows that imbalanced datasets pose a problem for kNN, neural networks (NN), and support vector machines (SVM) [3, 16, 17]. This problem is most pronounced when the two classes overlap as in the case of the Kaggle dataset; the majority of machine intelligence algorithms are not suited to handle both class unbalanced and overlapped class distributions [3, 10, 11].

Fortunately, particular algorithms exist that can take class imbalance into account. In addition there are techniques at the data level and algorithm level which can reduce the negative effects of these biases.

2) Concept Drift: Credit card fraud is prone to concept drift as consumer trends change due to changing preferences, seasonality and new products, as well as evolving fraud attack strategies [7]. The net effect of this is that the statistical properties of the underlying data change over time. Recent research has shown that it is possible to overcome this while still maintaining conventional machine intelligence techniques. For example, a sliding window approach where a classifier is trained everyday on the most recent samples, or an ensemble approach where the oldest component is replaced with a new classifier [7].

However, for the purposes of this paper the challenges associated with concept drift, as well as their resolutions, are not explored for two reasons. The first is that the dataset used is collected over a period of two days, which may not be sufficient for concept drift to actually take place. The second reason is that as mentioned, research shows that concept drift in FDS may be appropriately tackled by using conventional methods that are employed to maintain only a local temporal memory of learned attributes [7]. In other words, once a method is found that performs well for short periods of time, i.e. sufficiently small periods of time where concept drift does not take place, its implementation can then be further improved to account for concept drift. Therefore the fraud detection methods explored in this paper would need further refinement before being applied to a data stream of longer than a handful of days. These refinements are discussed in more detail at the end of the paper.

3) Sampling: Sampling methods are used to compensate for the unbalancedness of the dataset by reducing the classes to near equivalence in size. Undersampling and oversampling are two roughly equivalent and opposite techniques which use a bias to achieve this purpose. More complex algorithms such as synthetic minority oversampling technique (SMOTE) and the state-of-the-art adaptive synthetic sampling approach (ADASYN) actually create new data points based on known samples and their features instead of simply replicating the minority class [3, 18]. However these algorithms rely on assumptions of the minority class and are generally computationally expensive. Particularly, the created data generally is an interpolation of prior data which may not actually provide a realistic approximation of if the classes were in fact, balanced. Despite this, sampling methods can provide a more robust approach to imbalance learning than other methods, for example cost-based techniques which penalize errors differently depending on class such that the minority class is favoured [12, 19]. For example, what cost to use?

In either case, sampling using ADASYN on the training data is compared with classic undersampling in prior research, and no sampling at all. The package used is “Imbalance-Learn” in Python2.7 [20]. The testing and validation data are not sampled such that reported final accuracies are not distorted. This is representative of real-life where the fraudulent cases would be in the extreme minority class. Finally, these results will be compared to cost-based balancing methods, if applicable.

4) Classification: Most FDS use supervised classification techniques to discriminate between fraudulent and legitimate transactions. Research on similar credit-card fraud datasets shows Random Forest (RF) having superior performance than NN and SVM when using undersampling to correct for class imbalance [6]. This finding is verified by exploring three general classes of techniques against the Kaggle dataset: linear methods, ensemble methods, and neural networks. Specifically, linear SVM, RF, and multi-layer perceptrons (MLP) with training data subjected to ADASYN are explored in this paper.

a) Support Vector Machine: The aim of a SVM is to fit a hyperplane between data points in space, i.e. support vectors, such that the samples are separated by the largest gap possible. Classification occurs by determining which side the gap new data points fall on. Typically a binary linear classifier is used although there exist non-linear methods for SVM. In this paper. The SVM algorithm used is from “scikit-learn” v0.18.1 using Python2.7 [21]. This SVM implementation provides a means to apply cost-based balancing which will be briefly explored.

b) Random Forest: Ensemble methods generally use several weak classifiers to obtain a more optimal performance than any one single algorithm. Notably, RF is a technique which constructs a multitude of decision trees, i.e. a forest. RF has the advantages of being effective on relatively small amounts of data, efficient and robust on large amounts of data, and can easily apply cost-based balancing [13]. The RF algorithm used is from “scikit-learn” v0.18.1 using Python2.7 [21].

c) Multi-Layer Perceptron: A multi-layer perceptron (MLP) is a fully connected feedforward neural network. MLPs can classify data which is not linearly seperable due to the use of a non-linear activation function at each node in the network. The network is trained using standard backpropogation.The MLP is implemented using “scikit-learn” v0.18.1 with Python2.7 [21]. “Tensorflow" v1.0 is also used to explore variant MLP architectures [22].

5) Validation: It is important to consider that a highly unbalanced dataset will by default report a high baseline accuracy. For example, the Kaggle dataset used in this paper has a baseline accuracy of of 99.827%. That is, if a binary classifier always chose the class of “no fraud”, since there are so few fraudulent transactions compared to legitimate ones a high accuracy would in fact be reported. But clearly the goal of flagging fraudulent transactions would not be accomplished. Therefore a more careful approach must be used: a confusion matrix that provides the True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN).

TABLE 1: Confusion Matrix

	True Fraud	True Legitimate
Predicted Fraud	TP	FP
Predicted Legitimate	FN	TF

From these rates, two other key metrics may be formulated: precision, the fraction of instances correctly classified as fraudulent, and recall, the fraction of fraudulent instances that are correctly classified.

$$recall = \frac{TP}{TP + FN} \tag{1}$$

$$precision = \frac{TP}{TP + FP} \tag{2}$$

The confusion matrix as well as precision and recall metrics provide a rich description of the performance of a FDS [3, 23]. Namely, FP is acceptable whereas FN is not, as it is preferred for a financial institution to expend some resources on an alert that ends up being legitimate than to miss a fraud altogether. Fraud analytics departments are viewed as a necessary cost of business by banks and other credit card issuers but the impacts of missed frauds are far greater, including loss of customer trust and legal ramifications. In such a case, the recall metric is important as any FN will decrease the score. Confusion matrices are created using *scikit-learn* v0.18.1 in Python2.7 [20].

Classification report provided by *scikit-learn* v0.18.1 is also used to assess algorithm performance. This report provides precision and recall values with respect to both classes as well as f-scores. This information is valuable for determining the degree to which the algorithm provides FP and FN.

By examining the area under the precision-recall (PR) curve (AUPR), different classification algorithms may be more effectively compared than by simply considering accuracy alone [4, 7, 23]. In this fashion an evaluation of a classifier may be obtained for the full range of thresholds providing a complete picture of its performance. The AUPR is formulated again using *scikit-learn* v0.17.1 in Python2.7 [21].

Finally, the AUPR is supplemented with the area under the receiving operating characteristics (ROC) curve (AUROC). The ROC is simply the TP rate plotted against the FP rate as the discriminatory threshold is varied for a binary classifier [23]. It is similar to the PR curve in that generally, the greater the area under the curve the greater the performance, but provides a different evaluation of an algorithm, namely recall v.s. the fall out, that is the probability of classifiying a legitimate transaction as fraudulent.

III. Results

A. Support Vector Machine

Precision-recall and ROC curves for different thresholds of the “c” threshold parameter in the linear SVM classifer on the training data upsampled using ADASYN are as follows:

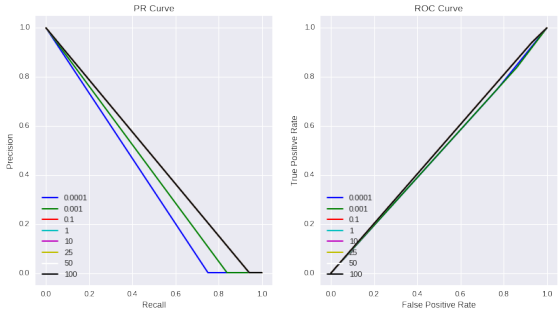

Fig 2: SVM PR & ROC for c threshold, ADASYN

Note the poor performance visible by the constant negative slopes for all threshold values. This is contrasted by Fig. 3 which demonstates the same algorithm and threshold values but with the raw unsampled data:

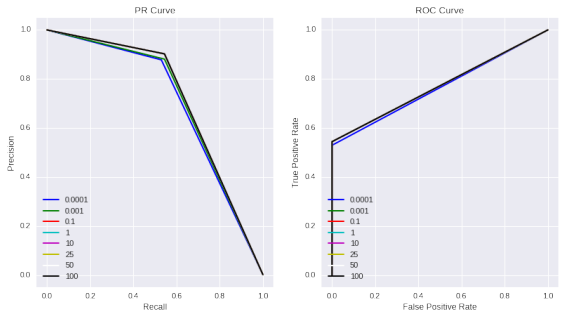

Figure 3: SVM PR & ROC for c threshold, unsampled

From these plots it is clear that the training data resampled with ADASYN actually resulted in worse performance over the raw unsampled data. Nevertheless both results suggest that a threshold value of c > 0.1 produces optimal performance.Using this result to the alternative cost-based method of class reweighting, the following plots are obtained:

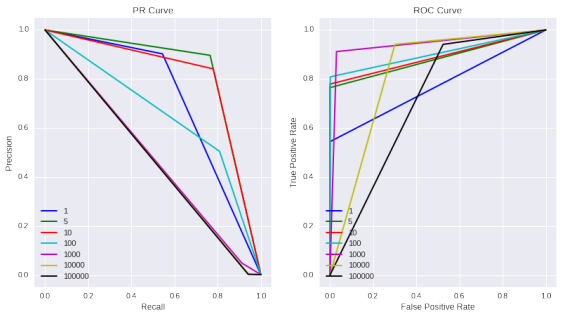

Fig 4: SVM PR & ROC for class_weights, c=1, unsampled

Based on the AUPR in Fig. 4 is it clear that a minority class weighting of 5:1 over the majority class on the unsampled data produces superior performance over both ADASYN sampled data and unsampled data without class reweighting. Considering the AUROC provides a different perspective: increasing the minority class weighting up to 1000 would increase the TP rate for a given FP rate with minimal impact over the FP rate.

Attempting to apply class reweighting to ADASYN upsampled data increases the performance over ADASYN unsampled data without class reweighting, but produces AUPR and AUROC less than the results obtained in Fig. 4, suggesting that ADASYN does not offer valuable contribution to the classifier:

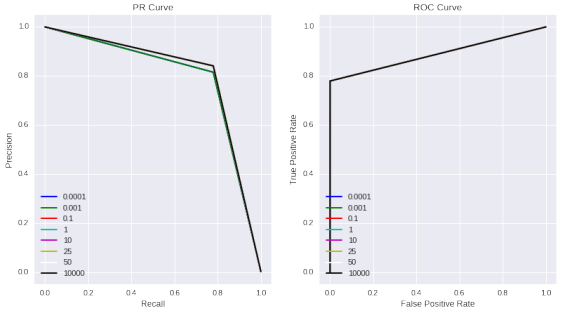

Fig 5: SVM PR & ROC for c threshold, class_weight={1:5,0:1}, unsampled

For a closer comparison of ADASYN without class reweighting, and the unsampled data with class reweighting observe the FN rates in the proceeding confusion matrices:

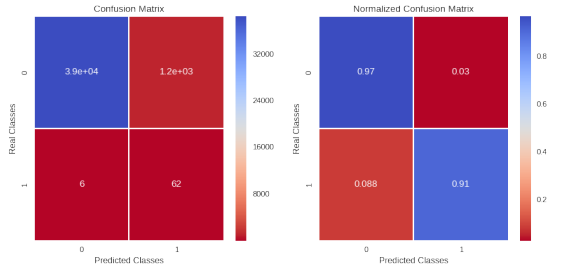

Figure 6: SVM, c=1, unsampled, class_weight={1:1000,0:1}

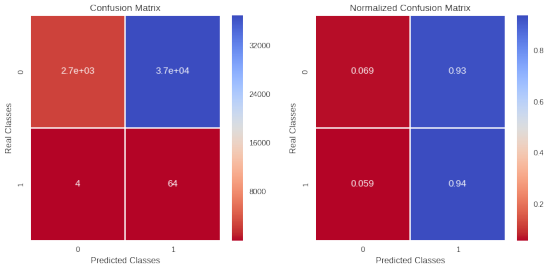

Figure 7: SVM, c=1, ADASYN

At first glance at Fig. 7 it seems that the classifer identifies fraudulent transactions to a greater degree using ADASYN than when trained with unsampled reweighted data, shown in Fig. 6. However the increased minority class recall comes at the cost of a substantially higher FP rate of 93%. Clearly the classifier trained using ADASYN is immensely biased to the minority class. SVM trained using unsampled data with class weighting obtains superior performance: 91% TP and 97% TN. Compared to unsampled training data without class reweighting, the FN rate is reduced by over 37%.

B. *Random Forest*

Although RF is suggested to have optimal performance over SVM and NN [3], the obtained results indicate otherwise. Figure 8 below demonstrates extremely poor performance for any value of the number of estimators parameter when training with ADASYN sampled training data:

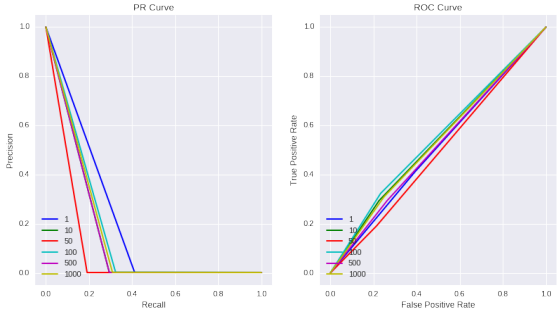

Fig 8: RF PR & ROC for n_estimator, ADASYN

This contrasts substatially with the same algorithm using unsampled and unweighted data:

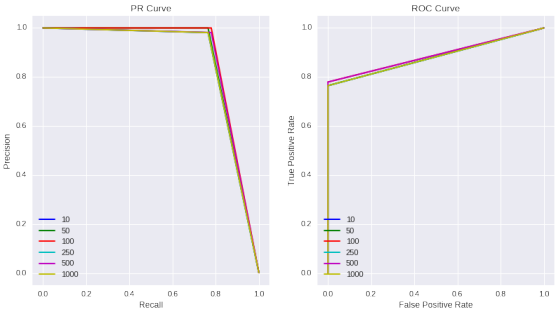

Fig 9: RF PR & ROC for n_estimator, unsampled & unweighted

In the case of RF, taking the best performing value for the number of estimators and observing AUPR and AUROC for varying class reweighting does not have as much performance benefit than in the case of SVM, observable in Fig. 10 below:

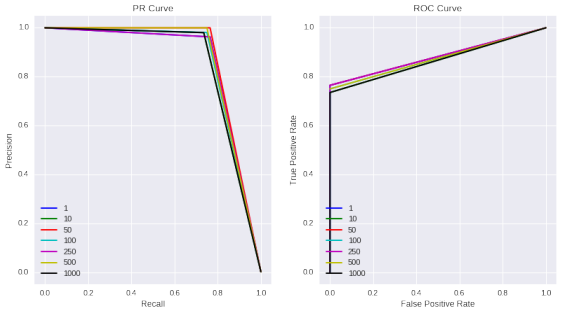

Figure 10: RF PR & ROC for class_weight, 100 estimators, unsampled

Once again, comparing the confusion matrices for RF when using ADASYN sampled training data and when using unsampled data with class reweighting shows that ADSYN produces substantially worse performance: both TP and FP rates are compromised:

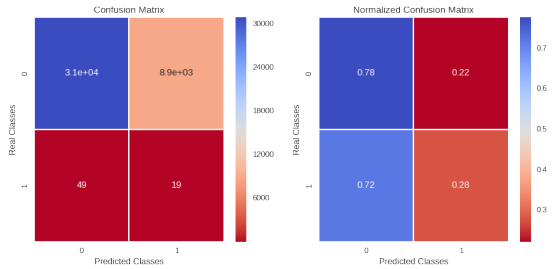

Figure 11: RF confusion matrices, 100 estimators, ADASYN

Clearly the classifier trained using ADASYN has very low fraud detection performance as it is biased towards the majority class. This is interesting as it is opposite from the results obtained using SVM with ADASYN.

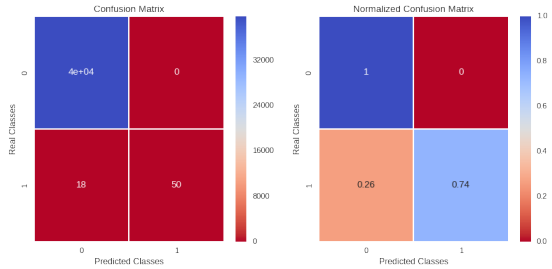

Figure 12: RF confusion matrices, 100 estimators, class_weight={1:10,0:1}

Figure 12 above shows that RF on unsampled but reweighted classes produces a perfect FP rate, but a TP rate of 74%, much lower than that obtained using SVM.

C. *Multi-Layer Perceptron*

The same trend is followed when using MLP. Varying the number of layers using ADASYN sampled training data and stochastic gradient descent (SGD) shows poor performance across all parameter values:

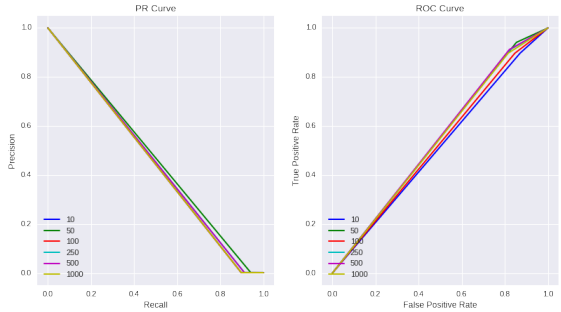

Figure 13: MLP PR & ROC for n_layers, ADASYN using SGD

According to the AUPR and AUROC, a layer size of 50 is optimal but clearly any layer size will produce either a high TP or TN rate, but not both.

This is constrasted in Fig. 14 below created by running the same algorithm but using unsampled unweighted data.

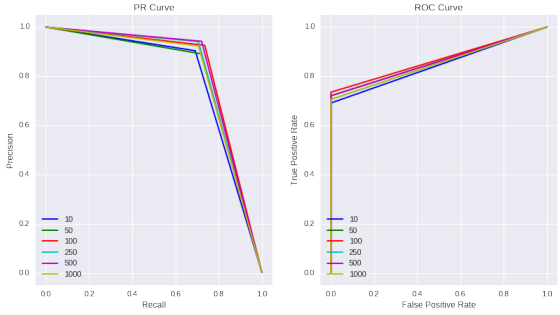

Figure 14: MLP PR & ROC for n_layers, unsampled & unweighted using SGD

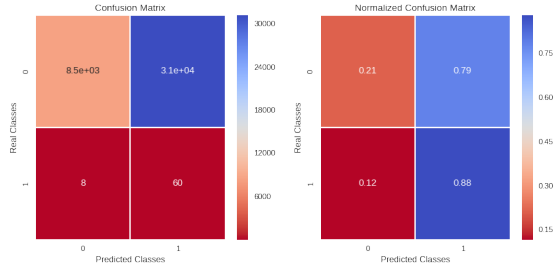

Figure 15: MLP confusion matrices, 100 layers, SGD, ADASYN

It is evident that using ADASYN sampled training data in fact cripples the MLP. Observing the confusion matrices confirms this finding:

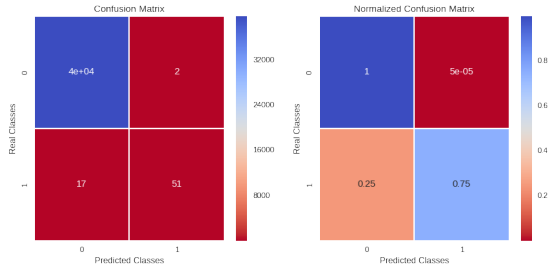

Figure 16: MLP confusion matrices, 100 layers, SGD, unweighted & unsampled

Note that in Fig. 15 the TP rate is actually fairly strong, but comes at the cost of a high FP rate indicating bias towards the minority class.

Based on these results an MLP trained on unsampled data has better FDS performance compared to ADASYN sampled training data, but compared to other methods has a much greater FN rate.

The performance of the MLP trained with SGD on unweighted and unsampled training data is nearly identical with the results of RF using reweighted classes trained with unsampled data. The differences are a slightly higher FP rate and a slightly lower FN rate, which is perefered by FDS.

IV. Discussion

The results largely show than compared to the unsampled dataset, the dataset upsampled with ADASYN had much worse performance. Additionally, the unsampled dataset using alternate class weighting to favour the miority class had superior performance than both the unsampled and ADASYN sampled datasets. This in fact contradicts current research which showed that ADASYN lead to increased overall performance in fraud detection versus other techniques [3, 18].

There are three suspected reasons that explain the discrepancy.

1) Thee generated data was not representative of true fraudulent transactions. This is possible in three parts: ADASYN failed to create synthetic samples which captured the underlying characteristic of fraudulent transactions, or that fraudulent transactions, at least in the used dataset, do not have strong enough differentiation when compared to legitimate transactions. Or, the class imbalance was simply too great to mitigate using ADASYN. However the positive results obtained using the unsampled and unweighted methods suggest that the classes are in fact separable. Therefore it is suspected that ADASYN did not manage to create new fraudulent samples that were representative of the data. Whether or not this is due completely to the extreme class imbalance or due to the nature of the data itself is unknown.

2) The specific implementations of the SVM, RF, and MLP algorithms used were not able to generalize to highly imbalanced test and validation data, after training with equally balanced data generated by synthetic sampling. This is supported by the high FP rate when ADASYN is used.

3) It is possible that the Python2.7 package used for the ADASYN algorithm, “Imbalance-Learn” [20], contained errors in the implementation that led to the poor results observed. The package is flagged as “experimental” and “use at your own risk”.

Comparing the performance of the different classifiers used, it is clear that linear SVM produces the best performance over RF and MLP . In such case it was the unsampled training data with class reweighting that obtained optimal classification. MLP trained with unweighted and unsampled data came in second but with a FN rate three times high compared to SVM. RF came closely in third using unsampled training data with class reweighting.

V. Conclusions

The obtained results contradicts results from prior research which suggest upsampling using techniques such as ADASYN increase performance in binary classification of highly imbalanced datasets. It is clear that when compared to conventional methods for dealing with class imbalance such as cost-based methods, or even undersampling, creating synthetic samples can result in much worse performance. In many cases it can actually handicap the classifier and produce results worse than not accounting for class imbalance at all.

It is recommended that further research is done to examine the conditions upon which upsampling techniques such as ADASYN may successfully increase performance in extremely unbalanced datasets. Particularly it is suggested that a closer examination of the upsampled data in this paper is done to visualize and understand the characteristics of the synthetic samples and to evaluate to what degree they are representative of actual fraudulent samples.

Second, it is recommended that examination of the results of using ADASYN to upsample the minority class to a lesser degree is done, that is, rather than upsampling to produce equivalent class sizes, only upsample to reduce the class imbalance. When the class imbalance is 99.8% biased towards the majority class, is is feasible that attempting to upsample too much produces a dataset that when trained on a classifier, will result in a classifier that is always biased towards the minority class, indicated by high FP rates.

Third, it is recommended that upsampling techniques such as ADASYN combined with conventional class imbalance mitigation techniques such as class reweighting are further explored. Although in this paper combining the two techniques lead to poorer performance compared to class reweighting alone, it is conceivable that by correcting for the performance impacts that ADASYN produced on the results in this paper, applying an additional technique that is shown to increase performance will result in a globally optimal FDS which may use off-the-shelf implementations of conventional classifiers.

Finally, although SVM trained with unsampled data with class reweighting reported higher performance in fraud detection than MLP trained with unsampled data and unweighted classes, it is recommended that applying class reweighting towards MLP training is explored. This is because a higher performance is noted for MLP than SVM on unsampled and unweighted data. It is expected that if class reweighting may be applied towards the MLP, despite being an unconventional technique, similar performance gains will be obtained than what was observed with SVM.

REFERENCES

[1] *"The Nilson Report," HSN Consultants., Carpinteria, CA, Issue 1096, Oct. 2016. [Online] Available: https://nilsonreport.com/publication_newsletter_archive_issue.php?issue=1096*

[2] "The Nilson Report," HSN Consultants., Carpinteria, CA, Issue 1101, Jan. 2017. [Online] Available: https://nilsonreport.com/publication_newsletter_archive_issue.php?issue=1101

[3] A. D. Pozzollo, "Adaptive Machine Learning for Credit Card Fraud Detection," Ph.D. dissertation, Dept. Comp. Sci., Univ. Libre de Bruxelles, Bussels, Belgium, 2015.

[4] L. Delamaire, H. Abdou, and J. Pointon. Credit card fraud and detection techniques: a review. Banks and Bank Systems, 4(2):57–68, 2009.

[5] A. D. Pozzolo, *et. al*., "Calibrating Probability with Undersampling for Unbalanced Classification" in *Symposium on Computational Intelligence and Data Mining (CIDM)*, 2015 © IEEE. doi: 10.1109/SSCI.2015.33

[6] A. D. Pozzollo, *et. al*., "Learned lessons in credit card fraud detection from a practitioner perspective", submitted for publication in Expert Systems with Applications, Feb 2014.

[7] A. D. Pozzollo, "Credit Card Fraud Detection and Concept-Drift Adaptation with Delayed Supervised Information" in *International Joint Conference on Neural Networks (IJCNN)*, 2015 © IEEE. doi: 10.1109/IJCNN.2015.7280527

[8] M Krivko. "A hybrid model for plastic card fraud detection systems". Expert Systems with Applications, 37(8):6070–6076, 2010.

[9] Y. Sahin, S. Bulkan, and E. Duman. "A cost-sensitive decision tree approach for fraud detection". Expert Systems with Applications, 40(15):5916–5923, 2013.

[10] H. He and E. A Garcia. "Learning from imbalanced data. Knowledge and Data Engineering", IEEE Transactions on, 21(9):1263–1284, 2009.

[11] G. Batista, A. Carvalho, and M. Monard. "Applying one-sided selection to unbalanced datasets" in *MICAI 2000: Advances in Artificial Intelligence*, pages 315–325,2000.

[12] J. Laurikkala. "Improving identification of difficult small classes by balancing class distribution". Artificial Intelligence in Medicine, pages 63–66, 2001.

[13] Kaggle. (2017, Jan. 12). *Credit Card Fraud Detection* [Online]. Available: https://www.kaggle.com/dalpozz/creditcardfraud

[14] https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

[15] A. D. Pozzollo, "When is undersampling effective in unbalanced classification tasks?" in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery*, Porto, Portugal, 2015, pp. 200-215.

[16] S. Visa and A. Ralescu. "Issues in mining imbalanced data sets-a review paper" in *Proceedings of the sixteen midwest artificial intelligence and cognitive science conference*, pages 67–73. 2005.

[17] I. Mani and I. Zhang. "knn approach to unbalanced data distributions: a case study involving information extraction" in *Proceedings of Workshop on Learning from Imbalanced Datasets*, 2003.

[18] H. He, *et al*., "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". in *IEEE International Joint Conference on Neural Networks*, pages 1322–1328. IEEE, 2008.

[19] G. M. Weiss and F. Provost. "The effect of class distribution on classifier learning: an empirical study". Rutgers Univ, 2001.

[20] G. Lemaitre, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning," submitted for publication.

[21] Pedregosa, *et. al*., "Scikit-learn: Machine Learning in Python" in JMLR 12, pp. 2825-2830, 2011. [Online] Available: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

[22] M. Abadi, *et. al*., "TensorFlow: Large-scale machine learning on heterogeneous systems", 2015. [Online] Available: http://download.tensorflow.org/paper/whitepaper2015.pdf

[23] J. Davis and M. Goadrich. "The relationship between precision-recall and roc curves". in *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

APPENDIX

```
### Code Sample –
# contact yazan.obeidi@uwaterloo.ca for complete codebase

from collections import Counter
import pickle
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix,
classification_report, auc, precision_recall_curve,
roc_curve
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
from sklearn.utils import shuffle
import matplotlib.gridspec as gridspec
%matplotlib notebook


## Dataset preparation
data = pd.read_csv("data/creditcard.csv")
#Create dataframes of only Fraud and Normal transactions.
Also Shuffle them.
fraud = shuffle(data[data.Class == 1])
normal = shuffle(data[data.Class == 0])
# Produce a training set of 80% of fraudulent and 80%
normal transactions
X_train = fraud.sample(frac=0.8)
X_train = pd.concat([X_train, normal.sample(frac = 0.8)],
axis = 0)
# Split remainder into testing and validation
remainder = data.loc[~data.index.isin(X_train.index)]
X_test = remainder.sample(frac=0.7)
X_validation =
remainder.loc[~remainder.index.isin(X_test.index)]
# Use ADASYN to rebalance dataset
ada = ADASYN(n_jobs=3)
data_resampled, data_labels_resampled =
ada.fit_sample(np.array(X_train.ix[:, X_train.columns !=
'Class']), np.array(X_train.Class))
# Pickle for later retrieval
with open('pickle/train_data_resampled.pkl', 'wb+') as f:
    pickle.dump(data_resampled, f)

with open('pickle/train_data_labels_resampled.pkl', 'wb+')
as f:
    pickle.dump(data_labels_resampled, f)
# At this point if we print out a count of the resampled
data labels we get
# Counter({0: 227452, 1: 227324})
X_train_resampled = pd.DataFrame(X_train_resampled)
X_train_labels_resampled =
pd.DataFrame(X_train_labels_resampled)
```

```
X_train_resampled = pd.concat([X_train_resampled,
X_train_labels_resampled], axis=1)
X_train_resampled.columns = X_train.columns
X_train_resampled.head()
# Shuffle the datasets once more to ensure random feeding
into the classification algorithms
X_train = shuffle(X_train)
X_test = shuffle(X_test)
X_validation = shuffle(X_validation)
X_train_ = shuffle(X_train_resampled)
X_test_ = shuffle(X_test)
X_validation_ = shuffle(X_validation)
data_resampled = pd.concat([X_train_, X_test_,
X_validation_])
# Normalize the data to an average of 0 and std of 1, but
do not touch the Class column
for feature in X_train.columns.values[:-1]:
    mean, std = data[feature].mean(), data[feature].std()
    X_train.loc[:, feature] = (X_train[feature] - mean) /
std
    X_test.loc[:, feature] = (X_test[feature] - mean) / std
    X_validation.loc[:, feature] = (X_validation[feature] -
mean) / std
for feature in X_train_.columns.values[:-1]:
    mean, std = data_resampled[feature].mean(),
data_resampled[feature].std()
    X_train_.loc[:, feature] = (X_train_[feature] - mean) /
std
    X_test_.loc[:, feature] = (X_test_[feature] - mean) /
std
    X_validation_.loc[:, feature] = (X_validation_[feature]
- mean) / std
# Create labels
y_train = X_train.Class
y_test = X_test.Class
y_validation = X_validation.Class
y_train_ = X_train_.Class
y_test_ = X_test_.Class
y_validation_ = X_validation_.Class
# Remove labels from X's
X_train = X_train.drop(['Class'], axis=1)
X_train_ = X_train_.drop(['Class'], axis=1)
X_test = X_test.drop(['Class'], axis=1)
X_test_ = X_test_.drop(['Class'], axis=1)
X_validation = X_validation.drop(['Class'], axis=1)
X_validation_ = X_validation_.drop(['Class'], axis=1)
# Pickle and save the dataset
dataset = {'X_train' : X_train,
           'X_train_': X_train_,
           'X_test': X_test,
           'X_test_': X_test,
           'X_validation': X_validation,
           'X_validation_': X_validation_,
           'y_train': y_train,
           'y_train_': y_train_,
           'y_test': y_test,
           'y_test_': y_test_,
           'y_validation': y_validation,
           'y_validation_': y_validation_}
with open('pickle/data_with_resample.pkl', 'wb+') as f:
    pickle.dump(dataset, f)


## Sample Precision-Recall curve generating code for SVM on
ADASYN training data:
fig = plt.figure(figsize=(12,6))
ax1 = fig.add_subplot(1,2,1)
ax1.set_xlim([-0.05,1.05])
ax1.set_ylim([-0.05,1.05])
ax1.set_xlabel('Recall')
ax1.set_ylabel('Precision')
ax1.set_title('PR Curve')

ax2 = fig.add_subplot(1,2,2)
ax2.set_xlim([-0.05,1.05])
ax2.set_ylim([-0.05,1.05])
ax2.set_xlabel('False Positive Rate')
ax2.set_ylabel('True Positive Rate')
ax2.set_title('ROC Curve')

for c,k in zip([0.0001, 0.001, 0.1, 1, 10, 25, 50,
100],'bgrcmywk'):
    lsvm_ = svm.LinearSVC(C=c, dual=False,
class_weight={1:1,0:1})
    lsvm_.fit(dataset['X_train_'], dataset['y_train_'])
    y_pred = lsvm_.predict(dataset['X_test_'])

    p,r,_ = precision_recall_curve(dataset['y_test_'],
y_pred)
    tpr,fpr,_ = roc_curve(dataset['y_test_'], y_pred)

    ax1.plot(r,p,c=k,label=c)
    ax2.plot(tpr,fpr,c=k,label=c)

ax1.legend(loc='lower left')
ax2.legend(loc='lower left')
plt.show()


## Linear SVM on weighted unsampled training data
lsvm = svm.LinearSVC(C=1, dual=False,
         class_weight={1:1000,0:1})
lsvm.fit(dataset['X_train'], dataset['y_train'])
y_pred = lsvm.predict(dataset['X_test'])
y_pred_validation = lsvm.predict(dataset['X_validation'])
plot_confusion_matrix(dataset['y_test'], y_pred)
plot_confusion_matrix(dataset['y_validation'],
y_pred_validation)
```