Comparing Hard Neural Network based Undersampling and SMOTE Techniques for Handling Class Imbalance in Credit Card Fraud Detection

Lokiron Anda
ljanda22@earlham.edu
Earlham College
Richmond, Indiana, USA

ABSTRACT

Credit card fraud is a major issue that can result in significant financial losses for individuals and financial institutions. Machine learning techniques have been used to detect credit card fraud, but the imbalance of fraud transactions makes it challenging to develop accurate models. In this study, we are comparing two popular techniques: *Hard Neural Network based Undersampling* and *SMOTE (Synthetic Minority Oversampling Technique)* on an imbalanced credit card fraud dataset. We will apply both techniques to the training set and train a Random Forest algorithm classifier. We will evaluate the performance of the classifiers compared each other and to the baseline performance of the system without any techniques for addressing class imbalance. I have an hypothesis that the Hard Neural Network based Undersampling will result in a more interpretable compared to SMOTE because it directly removes instances from the training set than than generating synthetic instances. The findings in this study will be useful for financial institutions and researchers who are interested in developing accurate fraud detection models for credit card transactions.

ACM Reference Format:
Lokiron Anda. 2023. Comparing Hard Neural Network based Undersampling and SMOTE Techniques for Handling Class Imbalance in Credit Card Fraud Detection. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Credit card fraud is a significant problem for financial institutions and consumers worldwide. Machine learning algorithms have proven to be effective in detecting fraudulent transactions, but class imbalance is a common issue in credit card fraud detection, where the number of legitimate transactions heavily outnumber the number of fraudulent transactions [6]. Class imbalance can result in poor performance for machine learning models because machine learning models would not be able to have an accurate representation of the minority class causing the model to be inaccurate because of the lack of examples from the minority class compared to the majority class leading the model to have bias and more likely to incorrectly identify the fraud. The goal to fix class imbalance is to not have an overrepresentation of one of the classes. This will help the model to have a fair assessment between both classes and improve the performance of the model and it's accuracy. Therefore, addressing class imbalance is critical in improving the accuracy of fraud detection systems. This proposal aims to compare two techniques, *Hard Neural Network based Undersampling* [1] and *SMOTE* [4], to determine which technique is more effective in handling class imbalance in credit card fraud detection.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, July 2017, Washington, DC, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.1 Related Work

The techniques we used in this study are *Undersampling* and *Oversampling*. Undersampling is a technique used in machine learning to balance the class distribution of a dataset by reducing the number of instances in the over-represented class [3]. Oversampling is another technique used in machine learning to balance the class distribution by increasing the number of instances of the minority class sample instead of reducing the number of instances of the majority class sample [7]. In this section we will give an overview of an undersampling algorithm and an oversampling algorithm, Hard Neural Network based Undersampling and Synthetic Minority Oversampling Technique (SMOTE). The first technique involves training a neural network to identify fraudulent transactions that are difficult to detect and using these examples to balance the dataset [1]. The second techniques involves generating synthetic samples of the minority class to balance the dataset [4]. Both datasets will be used to train a machine learning model, *Random Forest*. The performances of the models will be evaluated on a test dataset. We will evaluate by using these performance metrics: *Precision*, *Recall*, *F1-score*, and *AUC-ROC*.

2 DESIGN

This study will examine the effectiveness of undersampling and oversampling in solving the class imbalance problem in credit card fraud detection. We will use an imbalanced credit card fraud dataset and apply the two techniques to solve the imbalance issue. We then apply the new datasets from the techniques to a Random Forest algorithm and evaluate the performance between the two techniques. Here are the objectives we will be looking for:

- Evaluate the performance of *Hard Neural Network based Undersampling* and *SMOTE* on imbalance credit card fraud detection datasets and compare it to the performance of the model without the techniques.

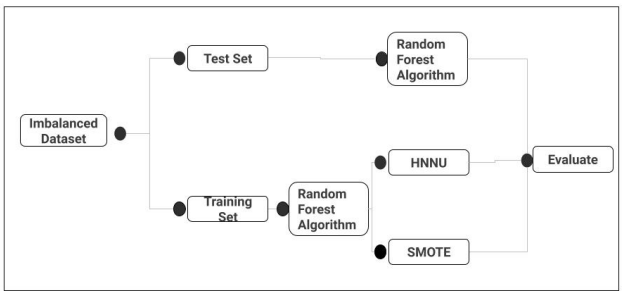

Figure 1: Visual of the Framework

- Compare the effectiveness of undersampling and oversampling in improving the performance of the machine learning model on an imbalanced datasets.
- Identify the strengths and weaknesses of each technique in terms of the model using the performance metrics: *Precision*, *Recall*, *F1-score*, and *AUC-ROC*

2.1 Framework

The dataset that is being used is *Credit Card Fraud Detection* by *The Machine Learning Group* [8]. We will spilt the data into training and testing sets. The data will be processed in a *Random Forest* algorithm. The Random Forest algorithm will be trained on the dataset to establish a baseline performance. We will train the *Hard Neural Network based Undersampling* technique to the majority class in the training set to create a balanced dataset. We will then train the Random Forest model on this balanced dataset and evaluate the performance. We will do the same procedure with *SMOTE*. We will apply the SMOTE technique to generate synthetic instances of the minority class in the training set to create a balanced dataset. We will train the Random Forest model with the balanced dataset and evaluate the performance. We will the evaluate the performance of Random Forest model with the techniques applied and the model without the techniques using the performance metrics. Lastly, we will give a statistical analysis determining if the techniques improved the performance of the model. The framework is shown in Figure 1. The reason why I chose this framework because it keeps the experiment consistent as each process will go through the same classifier and will showcase each technique and the improvement each of the techniques will show.

2.2 Dataset

The dataset contains transactions made by credit cards in September 2013 by European cardholders. It presents transactions that occurred in two days where we have 492 frauds out of 284,807 transactions. The dataset is highly imbalanced. The class of frauds on account for 0.172%. It contains only numerical input variables which are the result of a principle component analysis (PCA) transformation. Principal component analysis is used for analyzing large datasets containing a high number of dimensions per observation, as its goal is to keep the data understandable while preserving the maximum amount of information to protect the users identity.

2.3 Evaluation

Once the Random Forest model process all the datasets, we will evaluate the performance with these metrics [5].

Precision:

- Precision = True Positives / (True Positives and False Positives)

Recall:

- Recall = True Positives / (True Positives + False Negatives)

F1-score:

- F1-score = 2 * ((Precision * Recall) / (Precision + Recall))

AUC-ROC:

- AUC-ROC (Area Under the Receiver Operating Characteristic Curve) measures the overall performance of the classifier and provides a single value that summarizes the tradeoff between the Recall and false positive rate (1-specificity) across all possible threshold values
- ROC curve is created by plotting the true positive rate against the false positive rate for different threshold values.
- AUC-ROC is the area under this curve. it ranges from 0 to 1, with higher values indicating better performance.

2.4 Budget

This project uses Scikit-learn and Python, which are open source software and are available online. This project does not require any purchasing any software or hardware.

3 RANDOM FOREST

The machine learning model classifier that the two techniques will be applied to is the *Random Forest* algorithm [2]. Random Forest follows the principles of decision trees. Once the dataset is split into training and test sets, the algorithm builds a set of decision trees. The decision trees are made by randomly selecting features and data samples from the training set. Each decision tree is built using a random subset of the data. The process creates a diverse set of decision trees that can capture different aspects of the data. Once the decision trees have been built, they are used to make predictions on the test data. Each tree casts a vote, and the final prediction is made based on the majority vote. Random Forest has a lot of advantages over most algorithms. It's really good at handling large data-sets [2]. It's resistant against overfitting. It can handle missing data and outliers. Lastly, it's easy to interpret and visualize the results. Random Forest has been used to solve class imbalanced issues. For example, Zhu et al. [2] ran a study on the Random Forest algorithm and figured out a way to improved the performance of the algorithm when dealt with imbalanced medical data by implementing techniques to the data. Overall, a great algorithm to be used as the main classifier in this study.

4 HARD NEURAL NETWORK BASED UNDERSAMPLING (HNNU)

One approach to address imbalances in machine learning models is through the use of a technique called Hard Neural Network Based

Input: Training set $\mathcal{D}_n$, number of trees $M > 0$, $a_n \in \{1, \ldots, n\}$, $\texttt{mtry} \in \{1, \ldots, p\}$, $\texttt{nodesize} \in \{1, \ldots, a_n\}$, and $\mathbf{x} \in \mathcal{X}$.
Output: Prediction of the random forest at $\mathbf{x}$.

for $j = 1, \ldots, M$ **do**
  Select a_n points, with (or without) replacement, uniformly in $\mathcal{D}_n$. In the following steps, only these a_n observations are used.
  Set $\mathcal{P} = (\mathcal{X})$ the list containing the cell associated with the root of the tree.
  Set $\mathcal{P}_{\text{final}} = \emptyset$ an empty list.
  while $\mathcal{P} \neq \emptyset$ **do**
    Let A be the first element of $\mathcal{P}$.
    if *A contains less than* $\texttt{nodesize}$ *points or if all* $\mathbf{X}_i \in A$ *are equal* **then**
      Remove the cell A from the list $\mathcal{P}$.
      $\mathcal{P}_{\text{final}} \leftarrow Concatenate(\mathcal{P}_{\text{final}}, A)$.
    else
      Select uniformly, without replacement, a subset $\mathcal{M}_{\text{try}} \subset \{1, \ldots, p\}$ of cardinality $\texttt{mtry}$.
      Select the best split in A by optimizing the CART-split criterion along the coordinates in $\mathcal{M}_{\text{try}}$ *(see text for details)*.
      Cut the cell A according to the best split. Call A_L and A_R the two resulting cells.
      Remove the cell A from the list $\mathcal{P}$.
      $\mathcal{P} \leftarrow Concatenate(\mathcal{P}, A_L, A_R)$.
    end
  end
  Compute the predicted value $m_n(\mathbf{x}; \Theta_j, \mathcal{D}_n)$ at $\mathbf{x}$ equal to the average of the Y_i falling in the cell of $\mathbf{x}$ in partition $\mathcal{P}_{\text{final}}$.
end
Compute the random forest estimate $m_{M,n}(\mathbf{x}; \Theta_1, \ldots, \Theta_M, \mathcal{D}_n)$ at the query point $\mathbf{x}$ according to (1).

Figure 2: Random Forest Pseudocode [2]

Undersampling (HNNU) [1]. HNNU aims to balance the representation of minority and majority classes in the dataset. Here's how it works:

Data Preparation

- Determine the number of samples for both minority and majority classes, along with the number of attributes.
- The objective is to undersample the majority class

Model Training

- Train an autoencoder and an artificial neural network (ANN) exclusively on the minority class.
- Use a threshold value to decide whether a majority class sample is processed by the autoencoder or the ANN. If the number of attributes in a majority class sample exceeds the threshold, it is trained in the autoencoder; otherwise, it is trained in the ANN.
- All trained majority class samples are stored in a variable called "model".

Prediction and Selection

- Use the trained model to predict the majority class samples.
- Calculate the Euclidean distance between the original majority samples and the predicted majority samples.
- Sort the distances in ascending order and select samples with the highest distances, forming a new sorted list.

New Majority Class Samples

- Extract the selected samples from the sorted list to create a new set of majority class samples.

Dataset Creation

- Combine the new majority class samples with the original minority class samples, forming the final balanced dataset.

Model evaluation

- Apply this new dataset to a Random forest classifier, similar to the approach taken in the Random Forest section.

4.1 Summary

HNNU is preferable to randomly selecting a subset of the majority class for several reasons [1]:

Informed Undersampling

- HNNU doesn't randomly undersample the majority class; instead, it leverages informatuon from the minority class by training models (autoencoder and ANN) on it. This ensures that the undersampling process is more informed and take into account the specific characteristics of the minority class.

Feature Representation

- By using autoencoder and ANN, HNNU captures intricate feature representations of the minority class. Random undersampling may not consider the underlying patterns in the data, potentially leading to the loss of important information.

Adaptive Thresholding

- HNNU introduces a threshold mechanism to decide which majority class samples go to the autoencoder or the ANN. This adaptive thresholding allows for a more dynamic and tailored approach, as opposed to random sampling, which may not consider the varying complexities within the majority class.

Dimensionality Reduction

- The autoencoder component of HNNU can perform dimensionality reduction, focusing on the most relevant features. Random sampling does not inherently provide a mechanism for feature extraction or dimensionality reduction.

Euclidean Distance Criteria

- HNNU selects new majority class samples based on the Euclidean distance between original and predicted samples. This metric helps identify samples that are significantly different from the majority class distribution, resulting in a more representative subset.

Preserving Minority Class Information

- HNNU's process of generating new majority class samples is guided by the information learned from the minority class. This approach aims to preserve the important characteristics of the minority class during the undersampling process.

In summary, HNNU addresses class imbalance by strategically undersampling the majority class based on an autoencoder and ANN, resulting in a more balanced dataset for improved model training and evaluation.

5 SMOTE

Another technique that is used to handle class imbalance in machine learning models is SMOTE (Synthetic Minority Oversampling Technique). This technique involves oversampling the minority class by generating synthetic samples that are similar to the existing minority class samples. First, a minority class sample is selected from the original imbalanced dataset. Next, we select the k-nearest neighbors of the selected minority class sample. Once selected, we start generating synthetic examples by interpolating the selected minority class samples and the k-nearest neighbors. This involves taking the difference between each feature of the selected sample and its neighbor, multiplying this difference by a random value

$n_1 \leftarrow$ number of samples of the minority class
$n_2 \leftarrow$ number of samples of the majority class
$m \leftarrow$ number of attributes
$majSamples[1 \dots n_2] \leftarrow$ Samples of the Majority class
$minSamples[1 \dots n_1] \leftarrow$ Samples of the Minority class
if $m > threshold$ **then**
    $model \leftarrow autoencoder.train(minSamples[1 \dots n_1])$
else
    $model \leftarrow simpleANN.train(minSamples[1 \dots n_1])$
end if
$distArray \leftarrow \{\}$
for each $x \in n_2$ **do**
    $x' \leftarrow model.predict(x)$
    $d \leftarrow ||x - x'||_2^2$
    $index \leftarrow x.index$
    $distArray \leftarrow distArray \cup \{d, index\}$
end for
$sortedList \leftarrow sort(majSamples[1 \dots n_2], distArray)$ ▷ Sort the indices of n_2 samples according to descending order of distance
$selectedIndices \leftarrow sortedList[1..n_1]$ ▷ select first n_1 number of indices from the $sortedList$
$X_1 = \{\}$
for each $index \in selectedIndices$ **do**
    $X_1 = X_1 \cup majSamples[index]$
end for
$finalData = X_1 \cup minSamples[1 \dots n_1])$

Figure 3: HNNU Pseudocode [1]

```
Start
for i <- 1 to 10(k-nearest neighbors for 10)
    Compute k-nearest neighbors, and save the indices in the number of attribute.
end for
while
    Choose a random number between 1 and k, call it nn. Choose one of the k-nearest neighbors of 10.
        for j <- 1 to number of attribute.
            dif = MinorityClassSample(attribute(nn)(i)) - MinorityClass Sample[i][j]
            gap = rand() // between 0 and 1
            NewClassSample[newindex][j] = MinorityClassSample[i][j] + gap *dif
        end for
        newindex ++
end while
End
```

Figure 4: SMOTE Pseudocode [4]

between 0 and 1, and adding the result to the feature values of the selected sample. We repeat these steps until we generate enough synthetic examples. The synthetic examples will be combined with the original minority class sample to create a new balanced dataset [4]. This new dataset will then be processed through the Random Forest classifier.

6 FINAL RESULTS

In the context of addressing the issue of imbalanced data in a credit card fraud dataset, we applied Random Forest classification and experimented with two different techniques, namely HNNU (Hard Negative Mining Under-sampling) and SMOTE (Synthetic Minority Over-sampling Technique). The goal was to improve the classifier's performance in identifying fraudulent transactions.

	Precision Scores	Recall Scores	F1 Scores	AUC-ROC Scores
Random Forest	0.9412	0.7619	0.8421	0.8809
Random Forest & HNNU	0.9487	0.7551	0.8409	0.8775
Random Forest & SMOTE	0.9008	0.8027	0.8489	0.9013

Figure 5: Evaluation Scores Model

7 ANALYSIS

Precision:

- The dataset without sampling adjustments and the HNNU technique achieved similar high precision scores, indicating a low false positive rate in identifying fraudulent transactions. SMOTE had slightly lower precision, suggesting it may generate more false positives.

Recall:

- Recall: SMOTE showed a higher recall score, indicating a better ability to capture true fraudulent transactions. The dataset without sampling adjustments and the HNNU had lower scores than SMOTE, but very similar scores.

F1-Score:

- All three methods achieved comparable F1 scores, striking a balance between precision and recall.

AUC-ROC:

- AUC-ROC Score: The AUC-ROC scores for all three approaches were relatively close, indicating good overall model performance.

In summary, the Random Forest Classifier without any intervention demonstrated high precision but relatively lower recall. The introduction of HNNU slightly improved precision while maintaining a similar recall and F1-Score. On the other hand, applying SMOTE noticeably increased recall and AUC-ROC, suggesting better performance in handling the imbalanced nature of the dataset. The choice between these techniques depends on the specific goals and trade-offs in precision and recall that are acceptable for the given problem.

8 ACKNOWLEDGMENT

I would like to thank David Barbella for his support and feedback on my project idea and research proposal.

REFERENCES

[1] Md. Adnan Arefeen, Sumaiya Tabassum Nimi, and M. Sohel Rahman. Neural network-based undersampling techniques. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(2):1111–1120, 2020.

[2] Gérard Biau and Erwan Scornet. A random forest guided tour. *Test*, 25:197–227, 2016.

[3] Jason Brownlee. Random oversampling and undersampling for imbalanced classification, Jan 2021.

[4] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: Synthetic minority oversampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[5] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006.

[6] Sara Makki, Zainab Assaghir Assaghir, Yehia Taher, Rafiqul Haque, MOHAND-SAÏD Hacid, and Hassan Zeineddine. An experimental study with imbalanced classification approaches for credit card fraud detection, Jul 2019.

[7] Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah. Machine learning with oversampling and undersampling techniques: overview study and experimental results. In *2020 11th international conference on information and communication systems (ICICS)*, pages 243–248. IEEE, 2020.

[8] Machine Learning Group ULB. Credit card fraud detection, Mar 2018.