Machine Learning Engineer Nanodegree

Capstone Project

Vimugdh Lall.
April 22nd, 2017.

I. Definition

Project Overview

Domain Background

A credit card is a payment card issued to users (cardholders) to enable the cardholder to pay a merchant for goods and services, based on the cardholder's promise to the card issuer to pay them for the amounts so paid plus other agreed charges.

Credit card fraud is a wide-ranging term for theft and fraud committed using or involving a payment card, such as a credit card or debit card, as a fraudulent source of funds in a transaction. The purpose may be to obtain goods without paying, or to obtain unauthorized funds from an account. In the decade to 2008, general credit card losses have been 7 basis points or lower (i.e. losses of \$0.07 or less per \$100 of transactions) [**Source**: Paterson, Ken (December 2008). "Credit Card Issuer Fraud Management, Report Highlights" (PDF). Mercator Advisory Group. Archived from the original (PDF) on 29 December 2009]. In 2007, fraud in the United Kingdom was estimated at £535 million. [**Source**: *"Plastic card fraud goes back up". BBC. 12 March 2008. Retrieved 14 October 2013*]. Thus, we see other than the financial losses incurred by individual people, it's cumulative effects affect the economy of the entire country. With steps towards cashless economies, all outlets from large malls to small retail stores accept credit cards. Strides in internet technologies and our increasing dependency on e-commerce where plastic money governs most of the transactions, it has become even more important to make sure such fraudulent instances do not occur lest the people become apprehensive about using credit cards.

Datasets and Inputs

The dataset that is being used in this project is the Credit Card Fraud Detection dataset uploaded by Andrea Dal Pozzolo in Kaggle. The dataset has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. More details on current and past projects on related topics are available on http://mlg.ulb.ac.be/BruFence and http://mlg.ulb.ac.be/ARTML [**Citation:** Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015].

Problem Statement

The problem here is the theft and fraud committed using credit cards. It involves non-consensual/illegitimate use of a credit card. The logical definition of a credit card fraud might be to obtain goods without paying, or to obtain unauthorized funds from an account. Identifying such a transaction quickly (ideally in real time) is the problem that is to be solved.

Of course we cannot have a model that predicts a fraud before it happens but we can definitely have one that predicts whether or not a transaction was legit based on the details of that transaction, quick enough for the fraud to be detected and the perpetrator to be caught before it's too late.

We can create such a model by training it on the available data of all transactions to analyze the trends of the fraud as well as normal transactions such that the difference between them can be analyzed and ultimately the model learns to detect the fraud. The main problem here would be the extreme imbalance between the two sets of data. The fraud occurrences are so low compared to the normal transactions that even if the classifier identifies a True Negative as a False Negative, we would still get an accuracy of more than 99%. Thus we may have a variety of approaches to this problem:

- **Collect more data!**
  Collecting more data for the fraudulent transactions will lessen the imbalance. In this case, we will be able to use Accuracy because properly balanced data shall not penalize the Accuracy score as an imbalanced data would. But unfortunately, this isn't possible in this case as more of this data isn't available.
- **Choose a different Evaluation Metric.**
  Since the data imbalance severely affects the Accuracy scores, we could choose Recall or F-Score as performance metrics.
- **Resampling.**
  Resampling is a method that will essentially process the data to have an approximate 50-50 ratio. We shall use a sampling technique known as *undersampling* to adjust the class distribution of the dataset and produce more

data for the under-represented class. The model thus produced from the undersampled data can be updated and retrained with newer data of over and over again such that it achieves a good accuracy and over time becomes better at recognizing illicit activity quickly and efficiently with good accuracy.

In this project, we shall use both the second and the third approach to see how Recall acts as a performance metric in an imbalanced data.

Metrics

We shall use two different metrics to measure the performance of the various models that we use in this project – **Recall** and **Area Under Curve**.

When using the raw imbalanced data, in lieu of Accuracy, we shall use another evaluation metric that is frequently adopted in the research community to provide comprehensive assessments of imbalanced learning problems, namely, Recall.

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

Here, TP = Case was positive and predicted positive i.e., in this case, the transaction was fraud and it was predicted as fraud.

FN = Case was positive but predicted negative i.e., in this case, the transaction was fraud but we predicted it to be a normal transaction. We don’t want this to happen.

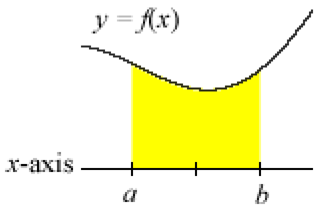
Intuitively, Recall is a measure of completeness (i.e., how many examples of the positive class were labeled correctly). However, unlike accuracy, Recall is not sensitive to changes in data distributions. A quick inspection on the Recall formula (mentioned above) readily yields that Recall is not sensitive to data distributions.

As we know, in this project, due to the imbalance of the data, many observations could be predicted as False Negatives, being, that we predict a normal transaction, but it is in fact a fraudulent one. Recall captures this. Obviously, trying to increase Recall, tends to come with a decrease of precision. However, in our case, if we predict that a transaction is fraudulent and it turns out not to be, is not a massive problem compared to the opposite.

The second metric that we shall use is Area Under Curve as has been used in the Benchmark Model to compare our results to theirs and have a common metric. It’s defined as follows:

The area between the graph of $y = f(x)$ and the x-axis is given by the definite integral below. This formula gives a positive result for a graph above the x-axis, and a negative result for a graph below the x-axis.

Note: If the graph of $y = f(x)$ is partly above and partly below the x-axis, the formula given below generates the net area. That is, the area above the axis minus the area below the axis.



$$Area = \int_a^b f(x)dx$$

II. Analysis

Data Exploration

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. It contains only numerical input variables which are the result of a PCA transformation. The following is a sample of the data.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...

...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

Features V1, V2, ..., V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise. Thus it clearly relates to the problem statement as it lists the fraudulent as well as legitimate transactions made using credit cards, which can be further analyzed to find the difference between the two.

The interesting fact about this dataset is that it is *highly unbalanced*. There are only 492 fraud transactions out of the total 284,807 transactions, making them only 0.17% of the total transactions. This histogram clearly shows the *huge* difference in the class distribution.

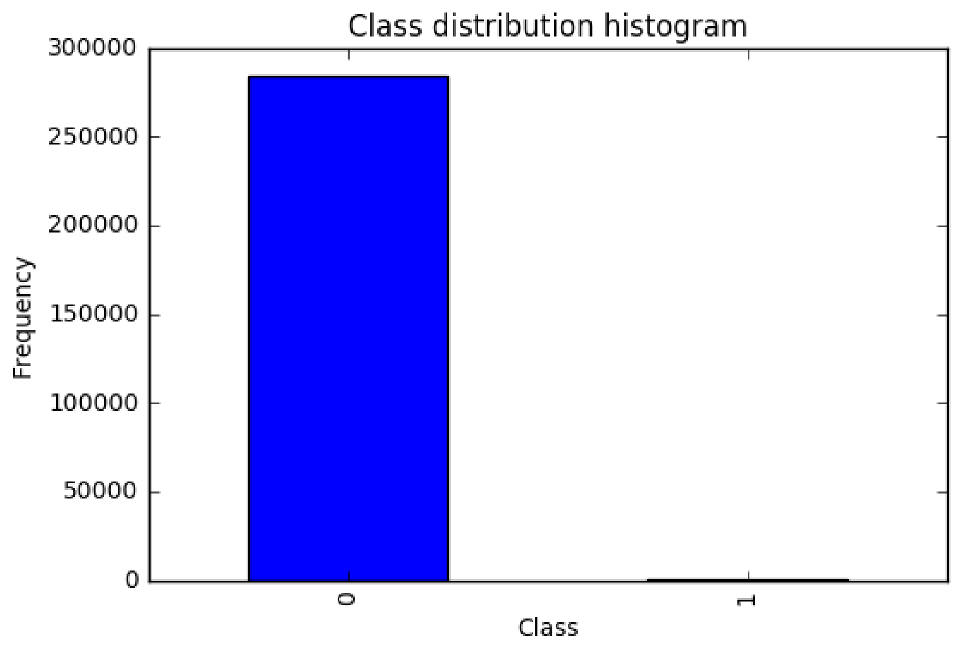


Fig 1: Histogram showing Class distribution between the Fraud and Normal transactions

We can clearly see the extreme difference between the data for the fraud and normal transactions in the 'Class' feature.

Exploratory Visualization

When comparing the 'Time' feature across the Fraud and Normal transactions (Fig 2), we find the Fraud transactions are more uniformly distributed than Normal transactions which have a more cyclic distribution. The Fraud transactions also have a zero-peak at a point which can also be used to identify a fraud transaction.

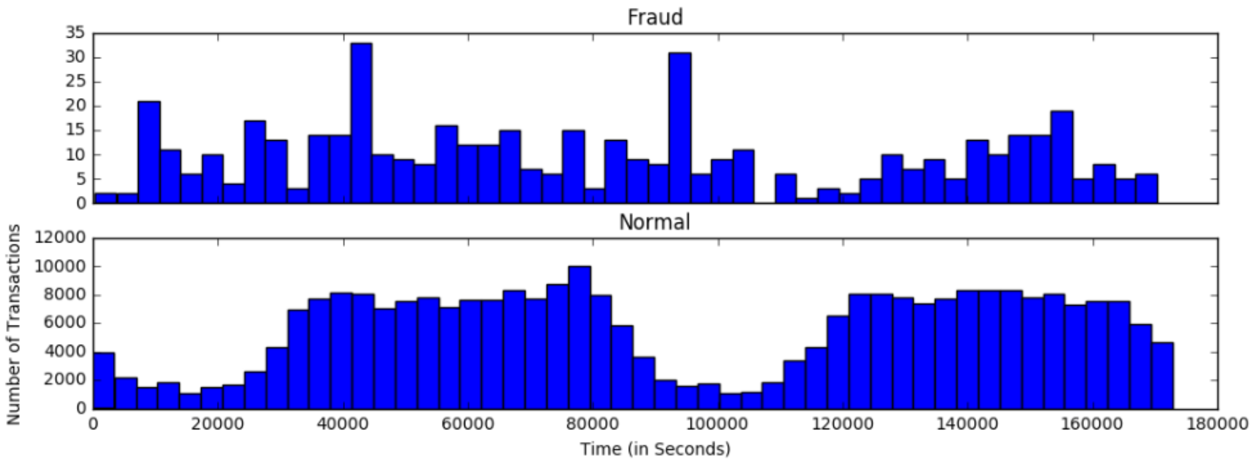


Fig 2: Histrograms show the Time stamps of the Fraud and Normal transactions

When comparing the 'Amount' feature across the Fraud and Normal transactions (Fig 3), we find a more interesting feature.

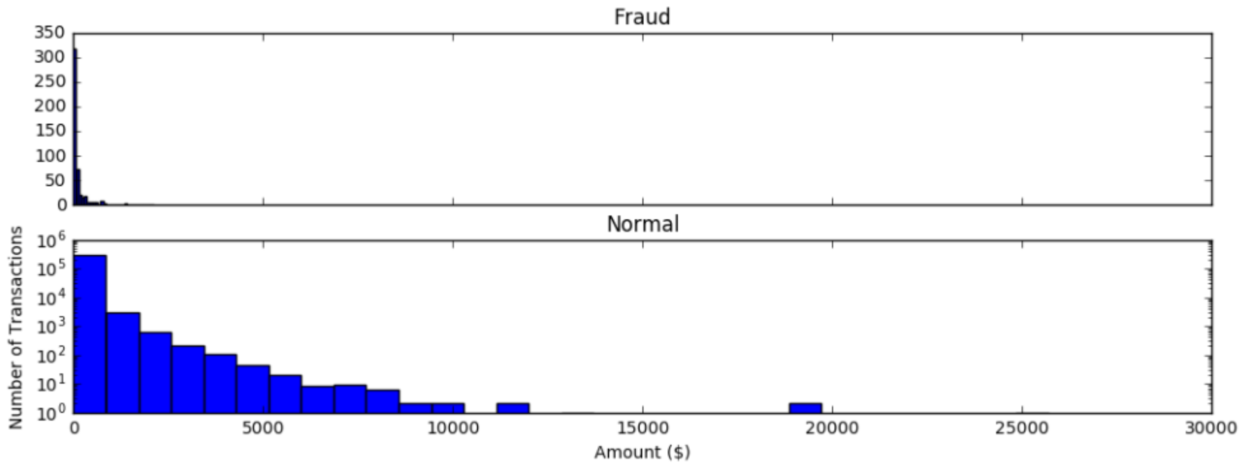


Fig 3: Histograms show the Amount of the Fraud and Normal transactions

As we can clearly see, the Fraud transactions are of very small quantities compared to Normal transactions. The maximum Fraud transaction is just $2125.87 when compared to the maximum Normal transaction of $25691.16. Also, as visible from the histogram, less than a 100 fraud transactions have maximum values as compared to the far greater number of normal transactions (>10^3) which have large transaction values.

Algorithms and Techniques

This is basically a problem of Classification where we have to predict if a transaction belongs to either the Fraud class or the Normal class. Hence we shall use a couple of algorithms based on our approach described earlier, for both the raw imbalanced and resampled data, and finally choose the one that performs the best.

The first classifier is the Linear Support Vector Classifier. Similar to SVC with parameter kernel = 'linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme and is also the first choice when dealing with Classification tasks that have <100K samples as shall be the case in our Undersampled dataset.

It has the following parameters that can be tuned to optimize it:

- *C* - Penalty parameter C of the error term.
- *loss* - Specifies the loss function.
- *penalty* - Specifies the norm used in the penalization.
- *dual* - Select the algorithm to either solve the dual or primal optimization problem.
- *tol* - Tolerance for stopping criteria.
- *multi_class* - Determines the multi-class strategy if y contains more than two classes.
- *fit_intercept* - Whether to calculate the intercept for this model.
- *intercept_scaling* - A "synthetic" feature with constant value equals to intercept_scaling is appended to the instance vector. The intercept becomes intercept_scaling * synthetic feature weight.
- *class_weight* - Set the parameter C of class i to class_weight[i]*C for SVC. If not given, all classes are supposed to have weight one.
- *verbose* - Enable verbose output.
- *random_state* - The seed of the pseudo random number generator to use when shuffling the data.
- *max_iter* - The maximum number of iterations to be run.

The second classifier is the Stochastic Gradient Descent Classifier. This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). This implementation works with data represented as dense or sparse arrays of floating point values for the features as is our data. The model it fits can be controlled with the loss parameter; by default, it fits a linear support vector machine (SVM). It is a good fit here because it works very well on a large number of samples (>100K).

It has the following parameter that can be tuned to optimize it:

- *loss:* Specifies the loss function.
- *penalty* - Specifies the norm used in the penalization.
- *alpha* - Constant that multiplies the regularization term.
- *l1_ratio* - The Elastic Net mixing parameter, with 0 <= l1_ratio <= 1.
- *fit_intercept* - Whether the intercept should be estimated or not.
- *n*_iter - The number of passes over the training data (aka epochs).
- *shuffle* - Whether or not the training data should be shuffled after each epoch.
- *random_state* - The seed of the pseudo random number generator to use when shuffling the data.
- *verbose* - Enable verbose output.
- *epsilon* - Epsilon in the epsilon-insensitive loss functions.
- *learning_rate* - The learning rate schedule.
- *power_t* - The exponent for inverse scaling learning rate.
- *average* - computes the averaged SGD weights and stores the result in the coef_ attribute.

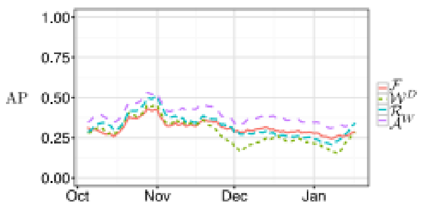
Thus, the two algorithms chosen are **Linear Support Vector Classifier (LinearSVC)**, which specializes in <100K data (as in undersampled dataset), and **Stochastic Gradient Descent Classifier (SGDClassifier)**, which specializes in >100K data (as in original dataset).

A technique known as *Random Undersampling* shall be used in this project to treat the data imbalance, in which data is randomly selected from the over-represented class and removed until it has the same number of samples as the previously under-represented class, thus creating a balanced dataset.

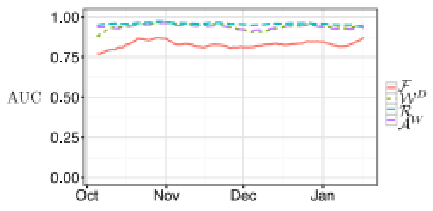
Benchmark

Citation: Dal Pozzolo, Andrea, and Gianluca Bontempi. "Adaptive machine learning for credit card fraud detection." (2015). [1]

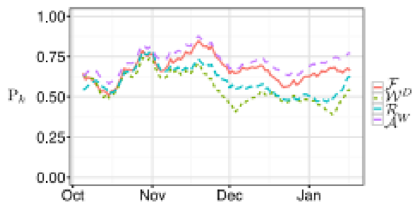
This research paper by Andrea Dal Pozzolo - who is also incidentally the source of the dataset used in this project - deals with different approaches to making an FDS (Fraud Detection System). They have considered two general approaches for fraud detection: a sliding window and an ensemble of classifiers. They have then compared FDSs that separately learn on feedbacks and delayed samples against FDSs that pool all the available supervised information together. Experiments run on real-world streams of transactions show that the former strategy provides much more precise alerts than the latter, and that it also adapts more promptly in concept-drifting environments. They have used accuracy as well as precision as their evaluation metrics. Their final model is based off on using smaller real-world inputs rather than delayed larger sets of supervised outputs to update their models. Thus, this also leads to using higher weights for the more recent data, giving it more priority, so that the smaller real-world input set is not dominated by the larger set of delayed inputs. In the end, they have made two separate models-the first by aggregating sets of real-world inputs and delayed inputs (Adaptive Aggregation) with a feedback mechanism and the other model by pooling together all supervised inputs in which the former model has shined forth as being better.



(a) Ranking measured in terms of **AP**



(b) Ranking measured in terms of **AUC**



(c) Ranking measured in terms of $\mathbf{P}_k$

Fig 4: Detection accuracy of A_t^W , F_t, W_t^D and R_t measured using different performance measures on the 2013 dataset. AUC and AP are measures of global ranking while P_k is a measure of ranking in the top *k t*ransactions with the largest probability of being fraudulent.

These are their results which are in line with the performance of our industrial partner (sometimes even better). However, for confidentiality reason, they are not allowed to disclose figures regarding the performance of their partner. The AUC scores that we shall use to compare our own model against theirs, is on an average 85% (red curve in Fig 4b).

III. Methodology

Data Preprocessing

There are several things that we shall do in data preprocessing.

i. Normalize the Amount feature.
ii. Random Undersampling (resampling) to balance the dataset and remove the skewedness mentioned in the **Data Exploration** section.
    Here on, we shall have two datasets - one that shall be undersampled and the other that shall be our actual dataset.
iii. Split each of the above mentioned datasets into test-train splits as we shall use Cross Validation for calculating performance.
iv. The performance of the Benchmark model is declared here for comparison later

Implementation

The implementation section has been divided into two parts which each part having to sub parts. The two major parts are:

❖ **Undersampled data**

The data is undersampled using Random Undersampling. A function has been written for that. The data is then split into two halves of Train and Test sets for cross validation.

- **LinearSVC**
  - The LinearSVC classifier is first used (with the fit( ) function) by training on the undersampled data.
  - The predictions are made (using the predict( ) function) on both the test sets of the undersampled and the raw dataset.
  - Recall and AUC scores are calculated for both.

- **SGDClassifier**
  - The SGDClassifier is first used (with the fit( ) function) by training on the undersampled data.

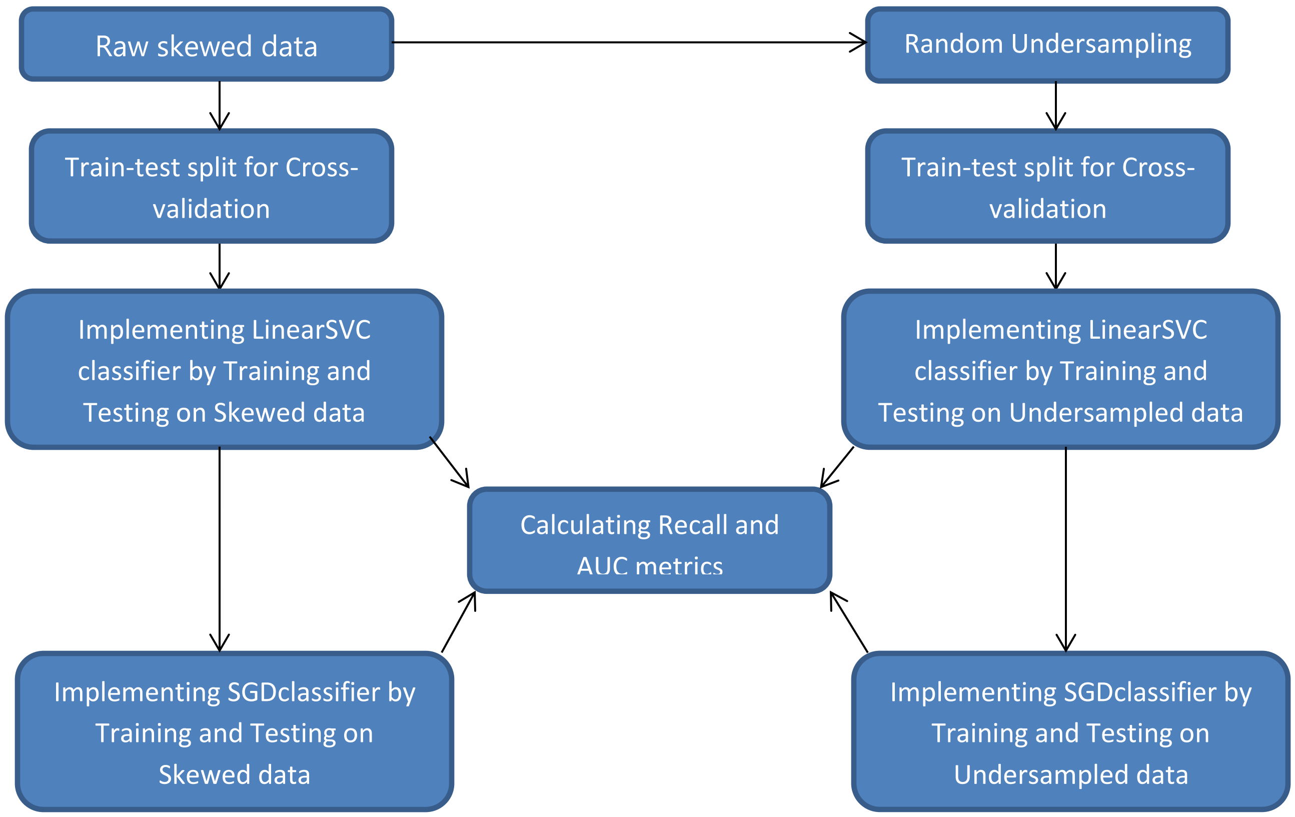
- The predictions are made (using the predict( ) function) on both the test sets of the undersampled and the raw dataset.
- Recall and AUC scores are calculated for both.

❖ **Skewed data**

The data is left raw and nothing is done to treat the imbalance of the data. The data is split into two halves of Train and Test sets for cross validation.

- **LinearSVC**
  - The LinearSVC classifier is used (with the fit( ) function) by training on the skewed data.
  - The predictions are made (using the predict( ) function) on the test sets of the raw dataset.
  - Recall and AUC scores are calculated.

- **SGDClassifier**
  - The SGDClassifier is used (with the fit( ) function) by training on the skewed data.
  - The predictions are made (using the predict( ) function) on the test sets of the raw dataset.
  - Recall and AUC scores are calculated.


Raw skewed data
Random Undersampling
Train-test split for Cross-validation
Train-test split for Cross-validation
Implementing LinearSVC classifier by Training and Testing on Skewed data
Implementing LinearSVC classifier by Training and Testing on Undersampled data
Calculating Recall and AUC metrics
Implementing SGDclassifier by Training and Testing on Skewed data
Implementing SGDclassifier by Training and Testing on Undersampled data

A slight complication arose while implementing the LinearSVC classifier that the '**l1**' *penalty* was not allowed to use with the '**squared_hinge'** *loss* function with the *dual* parameter set to '**True**' hence it had to be set to '**False**'.

Refinement

I have optimized the LinearSVC classifier by changing the *C parameter*. The function tries 5 iterations with a certain C parameter from the set {0.01, 0.1, 1, 10, 100} and compute their mean Recall score and at the end chooses the best C parameter to choose for the best Recall score.

```
-------------------------------------------
C parameter:  0.01
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.90000000000000002)
('Iteration ', 2, ': recall score = ', 0.93506493506493504)
('Iteration ', 3, ': recall score = ', 0.91891891891891897)
('Iteration ', 4, ': recall score = ', 0.92307692307692313)
('Iteration ', 5, ': recall score = ', 0.83333333333333337)

Mean recall score  0.902078822079

-------------------------------------------
C parameter:  0.1
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.88571428571428568)
('Iteration ', 2, ': recall score = ', 0.93506493506493504)
('Iteration ', 3, ': recall score = ', 0.91891891891891897)
('Iteration ', 4, ': recall score = ', 0.93846153846153846)
('Iteration ', 5, ': recall score = ', 0.84848484848484851)

Mean recall score  0.905328905329

-------------------------------------------
C parameter:  1
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.88571428571428568)
('Iteration ', 2, ': recall score = ', 0.93506493506493504)
('Iteration ', 3, ': recall score = ', 0.91891891891891897)
('Iteration ', 4, ': recall score = ', 0.92307692307692313)
('Iteration ', 5, ': recall score = ', 0.86363636363636365)
```

```
Mean recall score  0.905282285282
```

```
-------------------------------------------
C parameter:  10
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.87142857142857144)
('Iteration ', 2, ': recall score = ', 0.93506493506493504)
('Iteration ', 3, ': recall score = ', 0.91891891891891897)
('Iteration ', 4, ': recall score = ', 0.93846153846153846)
('Iteration ', 5, ': recall score = ', 0.84848484848484851)

Mean recall score  0.902471762472

-------------------------------------------
C parameter:  100
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.87142857142857144)
('Iteration ', 2, ': recall score = ', 0.93506493506493504)
('Iteration ', 3, ': recall score = ', 0.91891891891891897)
('Iteration ', 4, ': recall score = ', 0.93846153846153846)
('Iteration ', 5, ': recall score = ', 0.87878787878787878)

Mean recall score  0.908532368532

*************************************************************************
Best model to choose from cross validation is with C parameter =  100.0
*************************************************************************
```

These are the results reported as the process tries different C parameters. We can clearly see that the C parameter of 100.0 has the best mean Recall score of the five iterations and hence we choose it.

Similarly I have optimized the SGDClassifier by altering the *loss* function with a similar function that makes 5 iterations with different *loss* functions from the set {'hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_loss', 'huber', 'epsilon_insensitive','squared_epsilon_insensitive'} and computing their mean Recall score and at the end choose the best *loss* function for the best Recall score.

```
-------------------------------------------
Loss:  hinge
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.88571428571428568)
('Iteration ', 2, ': recall score = ', 0.94805194805194803)
```

```
('Iteration ', 3, ': recall score = ', 0.91891891891891897)
('Iteration ', 4, ': recall score = ', 0.9538461538461539)
('Iteration ', 5, ': recall score = ', 0.89393939393939392)

Mean recall score  0.920094140094


-------------------------------------------
Loss:  log
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.88571428571428568)
('Iteration ', 2, ': recall score = ', 0.93506493506493504)
('Iteration ', 3, ': recall score = ', 0.91891891891891897)
('Iteration ', 4, ': recall score = ', 0.93846153846153846)
('Iteration ', 5, ': recall score = ', 0.89393939393939392)

Mean recall score  0.91441981442

-------------------------------------------
Loss:  modified_huber
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.90000000000000002)
('Iteration ', 2, ': recall score = ', 0.93506493506493504)
('Iteration ', 3, ': recall score = ', 0.93243243243243246)
('Iteration ', 4, ': recall score = ', 0.92307692307692313)
('Iteration ', 5, ': recall score = ', 0.89393939393939392)

Mean recall score  0.916902736903

-------------------------------------------
Loss:  squared_hinge
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.90000000000000002)
('Iteration ', 2, ': recall score = ', 0.94805194805194803)
('Iteration ', 3, ': recall score = ', 0.91891891891891897)
('Iteration ', 4, ': recall score = ', 0.96923076923076923)
('Iteration ', 5, ': recall score = ', 0.87878787878787878)

Mean recall score  0.922997902998

-------------------------------------------
Loss:  perceptron
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.88571428571428568)
('Iteration ', 2, ': recall score = ', 0.94805194805194803)
('Iteration ', 3, ': recall score = ', 0.91891891891891897)
('Iteration ', 4, ': recall score = ', 0.92307692307692313)
('Iteration ', 5, ': recall score = ', 0.93939393939393945)

Mean recall score  0.923031203031
```

```
-------------------------------------------
Loss:  squared_loss
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.98571428571428577)
('Iteration ', 2, ': recall score = ', 0.42857142857142855)
('Iteration ', 3, ': recall score = ', 0.067567567567567571)
('Iteration ', 4, ': recall score = ', 0.86153846153846159)
('Iteration ', 5, ': recall score = ', 0.83333333333333337)

Mean recall score  0.635345015345

-------------------------------------------
Loss:  huber
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.11428571428571428)
('Iteration ', 2, ': recall score = ', 0.54545454545454541)
('Iteration ', 3, ': recall score = ', 0.90540540540540537)
('Iteration ', 4, ': recall score = ', 0.1076923076923077)
('Iteration ', 5, ': recall score = ', 0.51515151515151514)

Mean recall score  0.437597897598

-------------------------------------------
Loss:  epsilon_insensitive
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.11428571428571428)
('Iteration ', 2, ': recall score = ', 0.41558441558441561)
('Iteration ', 3, ': recall score = ', 0.77027027027027029)
('Iteration ', 4, ': recall score = ', 0.16923076923076924)
('Iteration ', 5, ': recall score = ', 0.66666666666666663)

Mean recall score  0.427207567208

-------------------------------------------
Loss:  squared_epsilon_insensitive
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.071428571428571425)
('Iteration ', 2, ': recall score = ', 0.42857142857142855)
('Iteration ', 3, ': recall score = ', 0.83783783783783783)
('Iteration ', 4, ': recall score = ', 0.92307692307692313)
('Iteration ', 5, ': recall score = ', 0.31818181818181818)

Mean recall score  0.515819315819

**************************************************************************
Best model to choose from cross validation is with Loss =  perceptron
**************************************************************************
```

These are the results reported as the process tries different *loss* functions. We can clearly see that the *loss* function of *perceptron* has the best mean Recall score of the five iterations and hence we choose it.

[**Note**: *One should however keep in mind that since we have chosen Random Undersampling and not Informed Undersampling as our resampling technique, the undersampled dataset that is chosen may be different everytime the function for the undersampling is compiled or executed. Hence, one must keep in mind that these values of the parameters (C and loss in LinearSVC and SGDClassifier resp.) have come out to be the best ones for the 'present' undersampled dataset and it is not at all necessary that they might have the same value for some other undersampled dataset. Thus, these values are chosen for the current undersampled dataset. However, even with different parameters, the results are always chosen to be the best because of the way the deciding function is written and there is merely a difference of 1-2% between the different sets of undersampled data.*]

IV. Results

Model Evaluation-Validation and Justification

Undersampled				**Skewed**			
LinearSVC		**SGDClassifier**		**LinearSVC**		**SGDClassifier**	
Recall	**AUC**	**Recall**	**AUC**	**Recall**	**AUC**	**Recall**	**AUC**
94.28%	99%	90.71%	97%	59.18%	96%	70.06%	88%

As we have seen from the **Benchmark Model** section, our benchmark AUC score is 85%. Also, as we have chosen Recall as our primary metric since it provides the most astute results when it comes to correctly predicting whether or not a transaction is fraud.

So, if we have to work with Undersampled data, I'd choose LinearSVC as our final model as it has really good Recall scores and it's AUC score is also higher than the AUC scores of both the Benchmark model as well the SGDClassifier and this agrees with our earlier hypothesis that it works better with data <100K.

But, if we have to work with Skewed data, I'd choose SGDClassifier as our final model because although, it's AUC score is lesser than the LinearSVC model, it is higher than the Benchmark model's AUC results. Most importantly, since it is our top priority to predict a sample correctly and Recall is our goal metric, I shall choose SGDClassifier as the final model here since it's Recall scores are way higher (difference of 10%) than that of LinearSVC model.

Both the models in their own domains have their results comparably as good as the Benchmark model and our expectations and hence, they can be trusted. Even if the undersampled data changes, the function that chooses the best parameter for both the models (as we have seen from the **Refinement** section), shall always try and produce the best results and although the model's solution shall be affected by different probabilities of the undersampled dataset selection, there shall be maximum difference of 1-2% and that should say that both these models are pretty robust in their own domains.

V. Conclusion

Free-Form Visualization

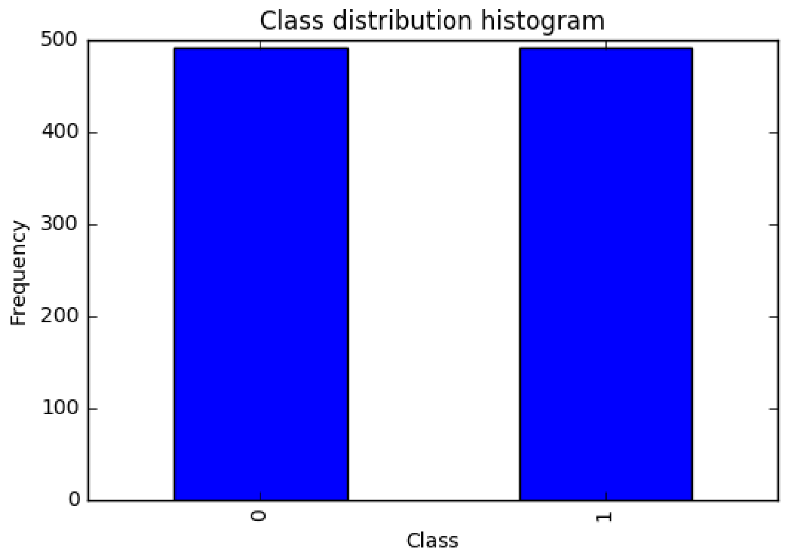
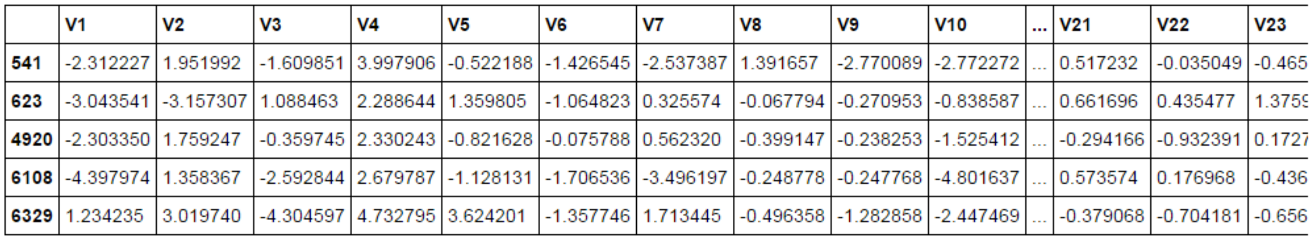


Fig 5: Histogram showing Class distribution between the Fraud and Normal transactions after Undersampling.

The thing that made this dataset unique was the extreme imbalance in the data as we saw before in Fig 1. To combat this, we decided to use Random Undersampling as a resampling technique so that we could have a more balanced dataset. Fig 5 shows the

Class distributions after Undersampling has been done and we see how perfectly the dataset is balanced. Now, in Andrea Dal Pozzolo's model, he has used real time data and used a weighing system where the incoming transactions details are given higher weights so that the algorithm treats them with higher priority. However, here we have not employed any weighted system and hence, we have no way of deciding which transaction is more important in the Normal transactions and that which we're going to keep in our Undersampled dataset and hence we shall have to provide equal weightage to all our samples. Due to this, there is no way of making an informed decision of which transaction to remove and which to keep. This is the reason we did not use *Informed Undersampling* in our project as a resampling technique and *Random Undersampling* was chosen instead. To check if the undersampled dataset is really random, here is a depiction of the first five samples of the dataset.



	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23
541	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.537387	1.391657	-2.770089	-2.772272	...	0.517232	-0.035049	-0.465
623	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.325574	-0.067794	-0.270953	-0.838587	...	0.661696	0.435477	1.375
4920	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.562320	-0.399147	-0.238253	-1.525412	...	-0.294166	-0.932391	0.1727
6108	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.496197	-0.248778	-0.247768	-4.801637	...	0.573574	0.176968	-0.436
6329	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.713445	-0.496358	-1.282858	-2.447469	...	-0.379068	-0.704181	-0.656

We can clearly see from the index positions (first column) that the samples are absolutely randomly chosen and have equal weightage.

Reflection

This project deals with two things-

- Dealing with data imbalance and finding different ways to overcome it. The two discussed here are resampling using randomized undersampling and changing metrics (using Recall in place of Accuracy).
- Using two algorithms to see which one performs better under which circumstance.

We have used also used cross validation to divide both the skewed dataset and the undersampled dataset into training and testing sets. The challenging part was choosing the best values for the parameters of each classifier to obtain the best results and also implementing both the algorithms and checking which one performed better in which circumstance. It was interesting to note that, undersampling other than reducing the data complexity and imbalance, also reduced the size of the dataset, which leaded to both the models performing extremely good on the datasets and at the same time, training and testing on the skewed datasets greatly reduced the recall scores of both the classifiers as was expected not only because of the imbalance of data but also due to the sheer size of the dataset. Although the results are somewhat good, they cannot be expected to be used in a general setting because credit card transaction frauds require real time predictions along with a steady influx of new data to keep updating the models which use a much more complex algorithm and consider more features than used here. Andrea’s model [1] is a more realistic reflection of what a real world model should look like.

Improvement

The model used here can be improved by trying other classifiers and checking their performance. The data provided here is also enough to train a neural network or use Deep Learning to get better results. In this project however, we could analyze the dataset much more deeply to see which features from the anonymized features play more important roles in prediction and remove the ones that probably don’t play as much role. The algorithm used by Andrea Dal Pozzolo’s research paper itself is amazing and is much more complex as it was built to cater to the real world predictions of fraud and if I knew how, I would implement it definitely. My solution is definitely not the best and it can definitely be improved by the above mentioned changes.
