Fraud Detection - A Machine Learning Perspective

Vibhavasu Pasumarti - EP20BTECH11015

***Abstract*— In this work, I explore how various classification algorithms detect genuine or fraudulent transactions in a dataset containing transactions done in September 2013 in Europe. The dataset, derived from PCA transformation, contains numerical features, with "Time elapsed between two transactions" and "Amount" preserved. The challenge lies in imbalanced dataset, with more real transactions than fraudulent ones. Different classifiers are tested to see how they handle this imbalance. Subsequently, various imbalance handling algorithms will be applied, and their effects on the results will be observed.**

I. Introduction

Credit card fraud is a big problem today. Quickly detecting fraudulent transactions might give a better chance of recovery.

Dataset

- The dataset obtained from Kaggle contains 284,807 transactions done in Europe during September 2013, out of which 492 have been marked fraudulent. Each transaction is marked whether fraudulent or not.
- The dataset contains 30 features, of which 28 features have been obtained from PCA transformation of original features.
- This has been done to preserve confidentiality.
- However, "Time elapsed between two transactions" and "Transaction Amount" are presented as such and have not been treated with the former method.

However, the dataset is highly skewed. The no.of normal transactions significantly outnumber the no.of positive transactions. For the preliminary run, the dataset is split into training, and testing sets in the ratio (70:30). For the final run, split into training, validation and testing sets in the ratio (70:15:15)

II. Objective

Explore how different Machine Learning algorithms work to classify the transactions in the given dataset as legitimate or fraudulent.

In an ML perspective, this represents a *Classification* problem.

Algorithms employed

For this project, the following classification algorithms are used:

- k - Nearest Neighbours Classifier
- SVM (Support Vector Machine) + soft margin
- Logistic Regression Classifier
- Naïve Bayes Classifier
- Decision Trees
- Neural Networks

The following metrics will be used to compare the results.

- F1 - Score
- Log loss
- Recall/True Positive Rate
- Negative Predictive Value[i]
- True Negative Rate[i]
- False Negative Rate[i]
- False Discovery Rate[i]
- Precision/Positive Predictive Value[*]*

III. Preliminary Results

The dataset is classified using *vanilla* SVM, k-NN, and Logistic Regression classifiers in this run.

A. Confusion Matrices

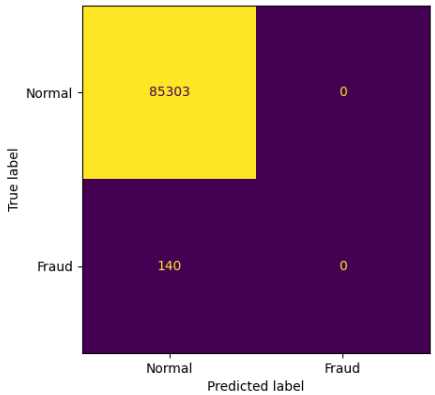

Fig. 1: Support Vector Classifier; C = 15

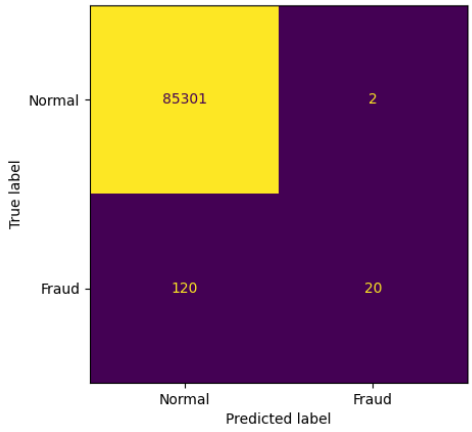

Fig. 2: k-NN Classifier

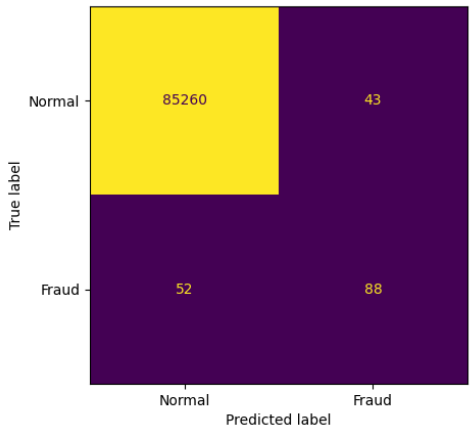

Fig. 3: Logistic Regression Classifier

B. Discussion

Observation: From the confusion matrces of SVM Fig:1 and kNN Fig:2, we see that most transactions are predicted as *normal*.

Accuracy is a bad metric as the number of FN is almost equal to the total no.of datapoints. This is because of the unbalanced nature of the dataset as discussed in I. *Precision* is not useful because it does not depend on True Negatives or False Negatives, and our aim is to detect fraudulent or True Negatives. It will be used as the last criterion while selecting a classifer.

The metrics (in table I) such as *F1-Score* and, *Recall* are very low, but they are moderately good for Logistic regression Fig:3.

[i]Not displayed for other classifiers as final report is instructed to be an *extension* to the intermediate report

Model	Precision	Recall	F1 Score	Log loss
SVM	0	0	0	0.059
k-NN	0.909	0.143	0.247	0.051
Logistic regression	0.672	0.629	0.649	0.040

TABLE I: Metrics for SVM and k-NN

Reason: This is because the data is highly imbalanced

1) For SVM: The objective is

$$\min(\|w\|^2 + C\sum_{i=1}^{n}\xi_i)$$
$$\text{s.t. } y_i(w\cdot f(x_i)+b) \geq 1-\xi_i \quad \text{where } \xi_i \geq 0 \quad (1)$$

The objective function has two parts: the first aims to maximize the separation between classes, while the second works to minimize mis-classifications. The parameter "C" is the cost assigned to mistakes, treating errors in both positive and negative cases equally. In situations where one class significantly outnumbers the other, especially near the class boundary, the majority class (here: "normal"/negative class) tends to influence the decision boundary against the majority class. However, this shift can lead to more mistakes, and in extreme cases of imbalance, the model may even classify all points as belonging to the minority (here: "fraudulent/positive") class as "normal". [1]

2) k-NN classifier: decides the category of a new item by looking at the classes of its closest neighbors from the training data. However, in imbalanced datasets where some classes are larger than others, i.e, most of the nearest neighbours belong to the majority class. This makes kNN to favor the larger classes when making predictions.

3) Logistic Regression Classifier: is robust to imbalanced datasets as it uses a function of negative log-likelihood (NLL $\cong$ MLE) as the loss function and gives probabilistic output. The NLL/MLE seeks parameters that maximize the likelihood of accurately predicting both majority and minority class instances, so the model remains balanced towards both classes. As the model outputs probabilities rather than discrete predictions, The logistic loss function used in logistic regression penalizes misclassifications proportionally,

effectively handling imbalances in the data.
This also means that imbalance handling algorithms(IV) do not increase the performance for Logistic Regression, as the maximum LLH looks to predict both positive and negative instances, so the no.of instances of each class do not create much bias.

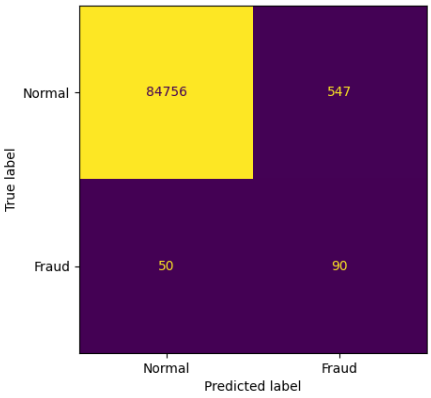

Fig. 4: Naïve Bayes Classifier

4) Naïve Bayes Classifier: is unfazed by data imbalance due to its underlying assumption of *feature independence*. This assumption allows the classifier to estimate the class probabilities based on the individual feature probabilities, regardless of the class distribution. Additionally, Naïve Bayes uses the maximum likelihood estimation to estimate the parameters of the feature probabilities. This estimation method is less affected by imbalanced datasets compared to other methods that rely on the overall class distribution[ii]

5) Decision Trees: is resistant to data imbalances as at each node, the because decision is taken based on the *Gini* index. (or other impurity measures) which depend on *proportions* rather than absolute counts. As a result, decision trees focus on the relative distribution of classes at each node, allowing them to be less sensitive to the absolute number of instances in each class.

$$\text{Gini index } G = \sum_c \Pi_c (1 - \Pi_c)$$

where

$$\Pi_c = \frac{\text{no.of points in the subspace with label } c}{\text{no.of points in the subspace}}$$

[ii]However, in an actual dataset, the assumption of independence does not hold, so the Naïve Bayes classifier cannot detect all the fraudulent transactions.

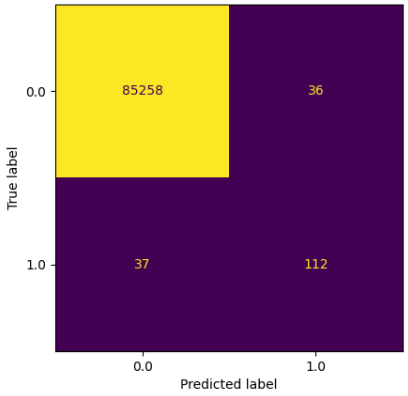

Fig. 5: Decision Tree Classifier

Since these depend on *proportions* rather than absolute counts decision trees focus on the relative distribution of classes at each node, allowing them to be less sensitive to the absolute number of instances in each class.

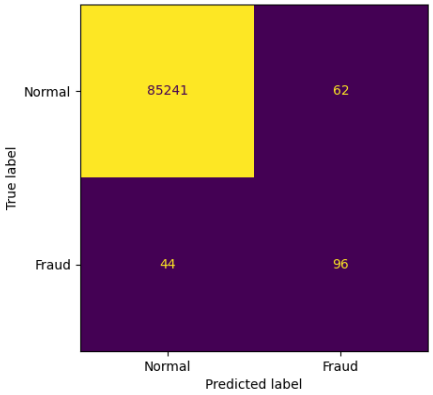

Fig. 6: Multi layer Perceptron

6) Multi-Layer Perceptron/Neural Network: In the training of an MLP, weights are updated using back-propagation and gradient descent. The weight updates are influenced by the gradient of the loss function with respect to the weights. In the case of imbalanced data, where one class has significantly fewer instances, the gradients associated with the minority class might be smaller. This can result in relatively smaller updates to the weights corresponding to the minority class during training. However, with higher dimensional MLPs, the overall prediction can be robust to imbalances as the number of nodes can counterbalance the small updates to the slow updation of weights.

IV. Results after imbalance handling

A. Oversampling

Oversampling[iii] is a technique to combat data imbalance. In this method, instances from minority class are randomly selected with replacement. Then these newly sampled instances are added to the original dataset and this process is repeated again till the no.of instances in minority and majority class are (nearly) equal.

B. SMOTE[2]

SMOTE, or Synthetic Minority Oversampling Technique, is an oversampling method particularly useful when the quantity of data available for the minority class is insufficient.

SMOTE begins by randomly picking a data point from the minority class. It then identifies its neighbors. A synthetic data point is then created at a random point between the chosen data point and one of its neighbors. This new point is not an exact copy of an existing data point, but rather a blend of two points, which helps to increase the diversity of the minority class in the dataset.

The key advantage of SMOTE is that it creates synthetic, yet realistic, examples of the minority class, rather than simply duplicating existing instances, which can lead to overfitting. This makes the learning algorithm more effective as it has a richer, more diverse parts of the distribution to learn from.

C. Undersampling

Undersampling is a data imbalance handling technique used when one class has significantly more instances than another.

Instances in the majority class are randomly removed until it's size is approximately to the size of the minority class.

The result is a balanced dataset, with roughly equal numbers of instances for each class. A main drawback with this is that it can lead to the loss of information, as instances from the majority class are randomly removed.

D. Bagging

Bagging, short for Bootstrap Aggregating, is a technique used in machine learning to improve the stability and accuracy of prediction models. It's a type of ensemble learning method where multiple base models are trained independently and their predictions are combined to make the final prediction.

[iii] Random Oversampling

Here's a simplified explanation of how bagging works:

Bagging begins by creating multiple subsets of the original dataset using bootstrapping. Each subset is then used to train a separate base model. The base models are trained independently of each other, and they can be of any type (decision trees, logistic regression, etc.). Once all the base models are trained, they each make predictions on the same test data. The final prediction is made by aggregating the predictions from all the base models. The aggregation method depends on the type of problem. For regression problems, the average of the predictions is taken. For classification problems, the class that gets the most votes is chosen (also known as majority voting).

E. Boosting[??]

Boosting is an ensemble method, which means it combines several "weak" models to create a "strong" model. The weak model is used to make predictions on the dataset, and the misclassified instances are identified. A new model is then trained, given more importance or *weight* to the instances that were misclassified by the previous model. This process is repeated several times, with each new model focusing on the errors of the previous model. All the models are combined to make a final prediction.

V. Choosing the best classifier for this data

1) The entire dataset is split into 3 parts: *Train*, *Validation* and *Test*; in the ratio $0.7 : 0.15 : 0.15$
2) Each classifier is tested along with the above Imbalance handling techniques. The imbalance handling technique that shows the most performance is selected for that classifier.
3) The (hyper-) parameters of each classifier are tuned using Grid Search Cross Validation using training and validation data.
4) The *Train* and *Validation* data are combined each classifier with the optimized parameters along with the imbalance handling techniques, is trained on this combined dataset to create a final model.
5) These final models are now tested on the *untouched* test data and performance metrics of all models are used to determine which model suits the best for the given dataset.

Classifier	Chosen Technique
SVM Classifier	Undersampling
k-NN Classifier	Oversampling
Logistic Regression Classifier	Bagging
Naïve Bayes Classifier	Boosting
Decision Trees	Bagging
Multi-layer Perceptron	SMOTE

TABLE II: Classifier and the imbalance handling technique that results in the best performance

VI. Results and Conclusion

Metric	SVM Classifier	Logistic Regression	kNN Classifier	Naive Bayes Classifier	Multi-Layer Perceptron	Decision Tree Classifier
F1-score	0.055	0.649	0.383	0.519	0.207	0.819
Log Loss	0.088	0.033	0.049	0.044	0.375	0.019
Recall	0.044	0.529	0.265	0.412	0.853	0.765
NPV	0.998	0.999	0.999	0.999	1.000	1.000
FPR	0.001	0.000	0.000	0.000	0.010	0.000
TNR	0.999	1.000	1.000	1.000	0.990	1.000
FNR	0.956	0.471	0.735	0.588	0.147	0.235
FDR	0.929	0.163	0.308	0.300	0.882	0.119

TABLE III: Results after parameter optimization

The table III shows the performance metrics of all the classifiers used in this analysis. The metrics are listed in the priority order they are used. i.e, F1-score is the metric with highest priority followed by Log loss and the rest.

From these metrics we conclude that *Decision Tree Classifier* with *Gini* criterion and a *maximum depth* of 6,is the best classifier for the above data. In decreasing order of choice of classifiers for this data set:

1) Decision Trees + Bagging
2) Logistic Regression Classifier + Bagging
3) Naïve Bayes Classifier + Boosting
4) k - Nearest Neighbours Classifier + Oversampling
5) SVM (soft margin) Classifier + Undersampling
6) Multi-layer Perceptron

Learning Outcomes

- Learnt the workings of different ML algorithms:
  1) k - Nearest Neighbours Classifier
  2) SVM (soft margin) Classifier
  3) Logistic Regression Classifier
  4) Naïve Bayes Classifier
  5) Decision Trees
  6) Multi-layer Perceptron
- Learnt practical difficulties with datasets: Imbalanced datasets and methods used to handle such datasets.
- Implemented these algorithms using python using sklearn, cuML, etc

Future works - What can be improved?

1) The cross validation of each classifier was mainly limited by the computing time. For a stronger reasoning to choose a particular classifier, it can be better to include a wider range of (hyper-) parameters.
2) Further optimization may be achieved if the parameters of the imbalance handling algorithms are also tuned.

References

[1] R. Akbani, S. Kwek, and N. Japkowicz, "Applying support vector machines to imbalanced datasets," in *Machine Learning: ECML 2004*, J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 39–50.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *arXiv e-prints*, p. arXiv:1106.1813, Jun. 2011.

- Dataset
- Code