Fraud Detection in Credit Card Transactions

Abstract

It is important for banks and other credit lenders to be able to recognise when a transaction is fraudulent as it occurs, as banks can then block the transaction and take the appropriate steps to secure the account. The goal of this report was to classify credit card transactions into categories depending on whether the transaction was fraudulent or not. The dataset used was the 'Credit Card Fraud Detection' dataset from kaggle. The dataset contains 284,807 transactions and fraudulent transactions count for only 492 of the transactions making the dataset imbalanced. This report used the Multi-Layer Perceptron classification model with undersampling to control for the imbalance in data. The results showed that undersampling improved the multi-layer perceptron classifier across all metrics. The MLP classifier performed with 95% accuracy, 0.05 MSE, 93% precision and 93% recall. In comparison to other studies this MLP model performed well, but was consistently outperformed by Random Forest models. However, comparison across studies showed that undersampling works well on this dataset.

Contents

Introduction

It is important for banks and other credit lenders to be able to recognise when a transaction is fraudulent as it occurs so that their customers are able to be informed that their information has been used without their permission, to block the transaction so that the customer is not charged for purchases they did not make, to secure the account and send a new card so that the old card with the fraudulent transactions can no longer be used. It would be impossible for banks to perform the task of checking for fraudulent transactions without the help of a program. The goal of this report was to classify credit card transactions into categories depending on whether the transaction was fraudulent or not.

The dataset used was the 'Credit Card Fraud Detection' dataset from kaggle. The dataset contains 284,807 transactions, the fraudulent transactions count for only 492 of the transactions making the dataset imbalanced as the majority of the data is non-fraudulent. The variables included in the dataset are 'time' (seconds elapsed between transactions), 'amount' (the transaction amount), and 'class' (used to show if a transaction was fraudulent or not, 1 indicating fraud and 0 indicating a normal transaction. Figure 1 provides a visualisation of the dataset. This allows for the visualisation of just how unbalanced the dataset is, as the fraudulent transactions making up only 0.172% of transactions can barely be seen.

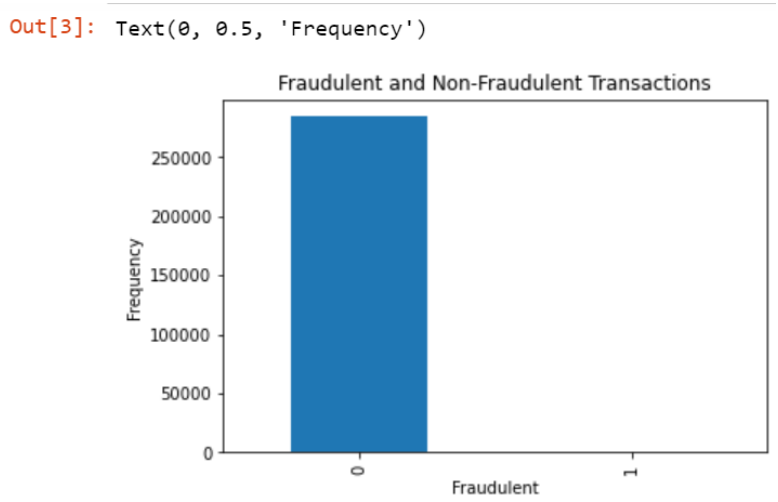


Figure 1: Bar Chart to show 'Fraudulent and Non-Fraudulent Transactions'

The deep learning models that were considered for the classification of this dataset were multi-layer perceptron and logistic regression. Other deep learning models such as CNN and RNN were not considered as it is primarily used for image/video datasets. SOM would not be suitable as the dataset contains a y column. The decision tree model was disregarded as the dataset is linear and decision trees work best for non-linear datasets.

Of the considered models the multi-layer perceptron model fits the dataset the best and produces the best result due to the hidden layers within the model which logistic regression lacks. Multi-layer perceptron models allow for a higher level of flexibility within the model, and work well with linear data.

Literature review

In this literature review, we consider papers that used the same dataset as used in this report. All papers identified that the data is very unbalanced (Dornadula & Geetha, 2019; Mahesh et al, 2022; Pundir & Pandey, 2021), this is to be expected with any real world credit card dataset as fraudulent transactions are not as frequent as non-fraudulent transactions. In order to control for the imbalance in the dataset several studies used resampling techniques. Praveen et al., (2019) found SMOTE (Synthetic Minority Oversampling Technique) to be an efficient method of resampling the data and making it balanced. Conversely, Dornadula & Geetha (2019) found that

using SMOTE didn't provide very good results and found that using a one-class SVM to be more effective. Mqadi et al., (2021) used undersampling to control for the imbalance in the data and found that to be an effective method of resampling within their model. However amongst the studies reviewed SMOTE was the more popular resampling technique. Melo-Acosta et al., (2017) observed that many different supervised and semi-supervised machine learning techniques are used for fraud detection. Dornadula & Geetha (2019 and Mahesh et al., (2022) found that logistic regression, decision tree and random forest algorithms gave the best results. Pundir & Pandey (2021) tested multiple models against one another and found that the random forest performed the best with 98.6% accuracy. Whereas both Carta et al., (2019), and Esenogo et al., (2022) used multi-layer perceptron models with good results.

Methodology

Packages and IDE

All packages were imported at the beginning to prevent the need for importing throughout. The packages that were imported were numpy to allow for the construction of arrays; pandas to read the data and store a dataframe; matplotlib.pyplot for chart construction and to allow for data visualisation; sklearn to use the preprocessing, neural network, model selection, utils, and metrics sub-packages.

Jupyter was used as the integrated development environment (IDE) used in this report. Jupyter was chosen as this IDE makes it easy to work cell by cell and show the output of each, making it perfect for data exploration. It is also the primary IDE used for research based coding and data visualisation.

Exploratory Data Analysis

Firstly pandas was used to read the csv file, .head was used to return the data in rows and columns.

Fraudulent and non-fraudulent operations were counted and displayed in a bar chart using pyplot. This bar chart shows that the data is clearly highly imbalanced with only 0.172% of the indices being fraudulent. A histogram was produced using numpy to show the amount of each transaction.

Testing the Model

First the variable amount was scaled using the standard scaler from sklearn preprocessing. The data was then separated into two arrays with x and y variables respectively. The data was split into training and testing datasets. The Classification was then implemented using the multi-layer perceptron model using hidden layer sizes of 200 and maximum iterations of 10,000, the data was scaled and fitted to the model. The model was then called, and accuracy recalled. The accuracy here is particularly low, this can be attributed to the data being imbalanced, this can be confirmed by the auPRC.

Undersampling and Implementing the Model

To control for the imbalanced data, undersampling will be implemented. Non-fraudulent indices (the majority type) will be removed to create a sample of data where the number of non-fraudulent indices and fraudulent indices will be equal, creating a balanced dataset.

Firstly the number of fraudulent cases were counted, then indices were extracted. Non-fraudulent indices were randomly generated to match the number of fraudulent indices, the arrays were then combined. X and Y arrays were then separated again. Train test split was then used again to split the undersampled arrays into training and testing samples.

Data was then scaled and fitted to the multilayer perceptron classifier, the classifier was called, returning an accuracy score which is much higher than previous.

Results

In this section results will be presented, I will be presenting the results of the model post-undersampling in comparison to the results pre-undersampling to demonstrate the importance of resampling an imbalance dataset and to provide the support for the efficacy of undersampling as a resampling method in machine learning models.

Pre-Undersampling & Post-Undersampling Comparison

Recall

The recall score is a measure of the classifier's ability to find all positive samples.

TP/(TP+FN)

	Pre-Undersampling	Post-Undersampling
Recall Score	0.52380	0.92517

As can be seen here the classifier post-undersampling performed better.

Precision-Recall Metrics

Below are the Precision-Recall metrics for Classifiers both pre and post undersampling.

	Pre-Undersampling		Post-Undersampling
Precision	[0.00172 0.78571 1.]		[0.49662 0.96454 1.]
Recall	[1. 0.52380 0.]		[1. 0.92517 0.]
Thresholds	[0 1]		[0 1]
Naive auPRC	0.65517		0.96343
Avg Precision Score auPRC	0.41238		0.92952

It can be seen that the undersampling of the data and therefore the balancing of the data improved the scores on every metric.

Classification Report

Pre-undersampling

	Precision	Recall	F1-score	Support
0	1.00	1.00	1.00	85296
1	0.79	0.52	0.63	147
Accuracy			1.00	85443
Macro avg	0.89	0.76	0.81	85443
Weighted avg	1.00	1.00	1.00	85443

Due to data imbalance, where there are many indices in class 0 and few in class 1,the classifier is overfitting hence the 1.00 score for class 0.

Post-undersampling

	Precision	Recall	F1-score	Support
0	0.93	0.97	0.95	149
1	0.96	0.93	0.94	174
Accuracy			0.95	296
Macro avg	0.95	0.95	0.95	296
Weighted avg	0.95	0.95	0.95	296

Here it can be seen that the undersampling has successfully controlled for the data imbalance as the classifier is no longer overfitting class 0.

Post-Undersampling Classifier

Metrics

Mean Squared Error: 0.05405

Accuracy: 0.94594

Precision: 0.92952

Recall: 0.92517

Receiver Operating Curve: (array([0. , 0.033557, 1.]), array([0. , 0.92517, 1.]), array([2, 1, 0], dtype=int64))

These metrics all indicate that the classifier can classify the indices well with a low margin of error.

Confusion Matrix

[[142 7]

[9 138]]

The confusion matrix shows the model to be performing well, the numbers of false positives and false negatives are low.

Discussion

Summary of results

The comparative results show the resampling method of undersampling to be working well in this classifier, the recall score improved by 40%. The classification report shows that the undersampling controlled for the model overfitting and produced good results.
The metrics show that the classifier performs well across all metrics and the confusion matrix confirms this.

Comparison with other studies

Dornadula and Geetha (2019) conducted their research on the same dataset and tested multiple models. In comparison to their results, this reports MLP performed better than their local outlier factor (accuracy: 0.4582 , precision: 0.2941), the MLP performed better than their isolation forest (accuracy: 0.5883 , precision: 0.9447) on accuracy but poorer on precision. Dornadula and Geetha's other models of logistic regression (accuracy: 0.9718 , precision: 0.9831), decision tree (accuracy: 0.9708, precision: 0.9814), and random forest (accuracy: 0.9998 , precision: 0.9996) out performed the MLP on this report.

Mahesh et al., (2022) used the same dataset and also tested multiple models with different resampling techniques, the MLP in this report out-performed KNN and Logistic Regression (LR) in all three resampling methods compared: undersampling (KNN: precision - 0.91, recall - 0.90; LR: precision - 0.92, recall - 0.91), SMOTE (KNN: precision - 0.82, recall - 0.86; LR: precision - 0.52, recall - 0.92), SMOTE-Tomek (KNN: precision - 0.82, recall - 0.91; LR: precision - 0.53, recall - 0.93). SVM was also out-performed for SMOTE (SVM: precision - 0.66, recall - 0.92) and SMOTE-Tomek (SVM: precision - 0.67, recall - 0.90). However, SVM out-performed the MLP model from this report in the undersampling method (precision - 0.94, recall - 0.93). Additionally the Random forest was only out performed by the MLP in the SMOTE method (precision - 0.91, recall - 0.92). For the other resampling methods the Random forest model outperformed the MLP (undersampling: precision - 0.94, recall - 0.94, SMOTE-Tomek: precision - 0.92, recall - 0.94).

Pundir & Pandey (2021), used the same dataset and compared the performance of decision trees, logistic regression and random forest classifiers. Comparing the accuracy of these three models with the MLP from this report, the MLP (accuracy: 95%) outperformed the decision tree (accuracy: 90.8%) but was outperformed by the logistic regression (accuracy: 98.5%) and random forest (accuracy: 99.1%) classifiers.

Conclusion

Multi-Layer Perceptron models fit the task of identifying credit card Fraud well, however MLP did not perform as well as some other models on the same dataset, the MLP was consistently

outperformed across studies by Random Forest classifiers (Dornadula & Geetha, 2019; Mahesh et al., 2022; Pundir & Pandey, 2021). In the literature review it was identified that SMOTE was the more popular resampling technique amongst studies reviewed?, however undersampling in this report performed as well as SMOTE, this is particularly prevalent in comparison to Mahesh et al., (2022) who also found within their own report that undersampling performed better. MLP classification models perform well alone but perform better as part of a larger system of models to identify fraudulent transactions as Carta et al (2019) identified.

References

Carta, S., Fenu, G., Reforgiato Recupero, D., & Saia, R. (2019). Fraud detection for E-commerce transactions by employing a prudential Multiple Consensus model. *Journal of Information Security and Applications*, *46*, 13–22. https://doi.org/10.1016/j.jisa.2019.02.007

Credit Card Fraud Detection. (n.d.). Kaggle: Your Machine Learning and Data Science Community. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?datasetId=310&language=Python&outputs=Data

Dornadula, V. N., & Geetha, S. (2019). Credit card fraud detection using machine learning algorithms. *Procedia Computer Science*, *165*, 631–641. https://doi.org/10.1016/j.procs.2020.01.057

Esenogho, E., Mienye, I. D., Swart, T. G., Aruleba, K., & Obaido, G. (2022). A Neural Network Ensemble With Feature Engineering for Improved Credit Card Fraud Detection. *IEEE Access*, *10*, 16400–16407. https://doi.org/10.1109/access.2022.3148298

Gupta, R. (n.d.). *6 Deep Learning Models and When to Use them*. Towards Data Science. https://towardsdatascience.com/6-deep-learning-models-10d20afec175

Mahesh, K., Ashar Afrouz, S., & Shaju Areeckal, A. (2022). Detection of fraudulent credit card transactions: A comparative analysis of data sampling and classification techniques. *Journal of Physics: Conference Series*, *2161*(1), 012072. https://doi.org/10.1088/1742-6596/2161/1/012072

Melo-Acosta, G. E., Duitama-Munoz, F., & Arias-Londono, J. D. (2017). Fraud detection in big data using supervised and semi-supervised learning techniques. In *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*. IEEE. https://doi.org/10.1109/colcomcon.2017.8088206

Mqadi, N. M., Naicker, N., & Adeliyi, T. (2021). Solving Misclassification of the Credit Card Imbalance Problem Using Near Miss. *Mathematical Problems in Engineering*, *2021*, 1–16. https://doi.org/10.1155/2021/7194728

Neural network models (supervised). (n.d.). scikit-learn. https://scikit-learn.org/stable/modules/neural_networks_supervised.html

Pundir, A., & Pandey, R. (2021). Data quality analysis based machine learning models for credit card fraud detection. *Journal of University of Shanghai for Science and Technology*, *23*(06), 318–344. https://doi.org/10.51201/jusst/21/05263

Appendices

Appendix A: Jupyter notebook

In [1]:

```
# importing packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import recall_score, roc_curve
from sklearn.metrics import classification_report, confusion_matrix
```

Exploratory Data Analysis

In [2]:

```
# reading csv
df = pd.read_csv("C:/Users/abbie/Documents/Comp Sci tings/Uni/Aplications of AI/Assignm
df.head()
```

Out[2]:

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

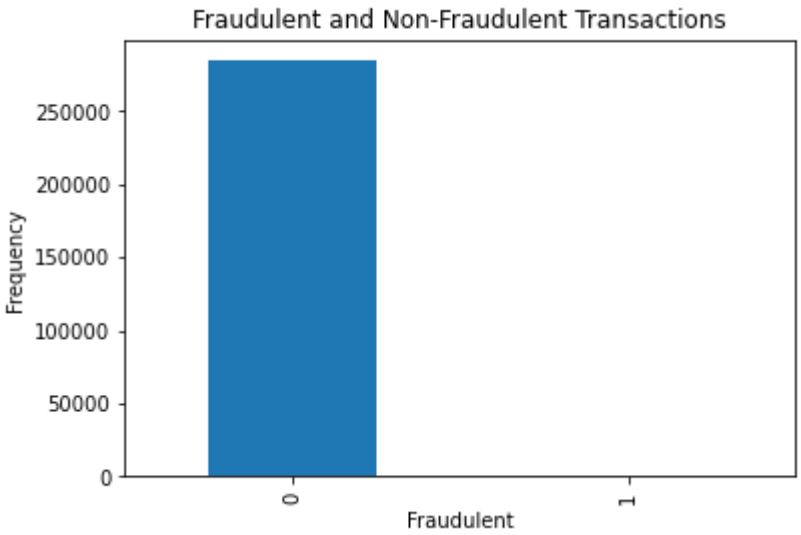
5 rows × 31 columns

In [3]:

```
# counting fraudulent and non-fraudlent operations
count_classes = pd.value_counts(df['Class'], sort = False)
# printing graph
count_classes.plot (kind='bar')
plt.title ("Fraudulent and Non-Fraudulent Transactions")
plt.xlabel ("Fraudulent")
plt.ylabel ("Frequency")
```

Out[3]:

```
Text(0, 0.5, 'Frequency')
```

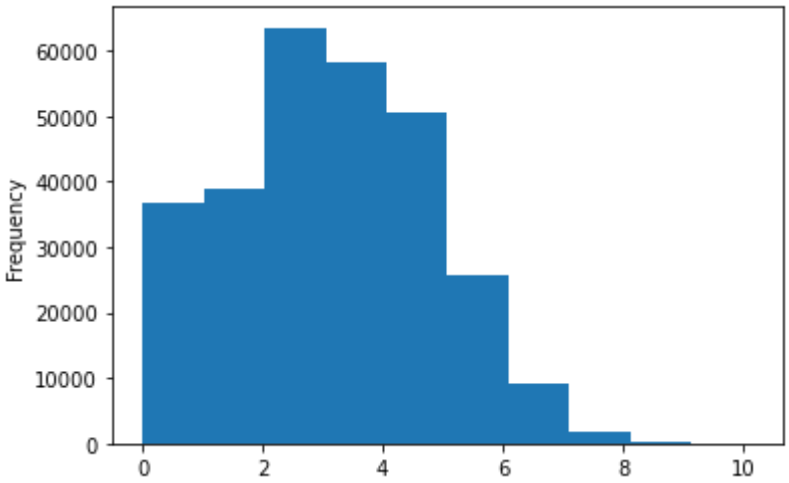


In [4]:

```
# applying formula to the columns of the df
df['logAmount'] = np.log(df['Amount']+1)
# printing histogram
df['logAmount'].sort_values().plot.hist()
```

Out[4]:

```
<AxesSubplot:ylabel='Frequency'>
```



Testing the data

In [5]:

```
# Scaleing the variable amount
df['normAmount'] = StandardScaler().fit_transform(df['Amount'].values.reshape (-1,1))
df = df.drop (['Time', 'Amount','logAmount'], axis = 1);
```

In [6]:

```
# Separating the data into two arrays with x and y variables respectively
X = df.iloc[:, df.columns != 'Class']
y = df.iloc[:, df.columns == 'Class']
len(y[y.Class ==1]);
```

In [7]:

```
#sorting with train test split
X_train, X_test, y_train, y_test = train_test_split (X,y, test_size = 0.3, random_state
```

In [8]:

```
#Using Multi-layer Perceptron Classifier Neural Network
MLPC = MLPClassifier(hidden_layer_sizes=(400,), max_iter=10000)
scaler = StandardScaler()
# fit to training data
scaler.fit(X_train)
scaler.fit(y_train)
scaler.fit(X_test)
MLPC.fit(X_train, y_train)
X_test = scaler.transform(X_test)
y_pred = MLPC.predict(X_test)
# Recall values
recall_acc = recall_score (y_test,y_pred)
recall_acc
```

```
C:\Users\abbie\anaconda3\lib\site-packages\sklearn\neural_network\_multilaye
r_perceptron.py:1109: DataConversionWarning: A column-vector y was passed wh
en a 1d array was expected. Please change the shape of y to (n_samples, ), f
or example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\abbie\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:
X does not have valid feature names, but MLPClassifier was fitted with featu
re names
  warnings.warn(
```

Out[8]:

```
0.5238095238095238
```

Unbalanced

The accuracy on the above classification is so low because the dataset is unbalanced. Below is a second measure of accuracy using auPRC since a confusion matrix cannot be used due to the unbalanced classification.

In [9]:

```
def pr(y_test, y_pred):
    precision, recall, thresholds = metrics.precision_recall_curve(y_test, y_pred)
    auc = metrics.auc(recall, precision)

    print('Precision: %s' % precision)
    print('Recall: %s' % recall)
    print('Thresholds: %s' % thresholds)
    print('Naive auPRC: %s' % auc)
    print('Average Precision Score auPRC: %s' % metrics.average_precision_score(y_test,

pr(y_test, y_pred)
```

```
Precision: [0.00172045 0.78571429 1.        ]
Recall: [1.         0.52380952 0.        ]
Thresholds: [0 1]
Naive auPRC: 0.655171534573592
Average Precision Score auPRC: 0.4123838854737148
```

In [10]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85296
           1       0.79      0.52      0.63       147

    accuracy                           1.00     85443
   macro avg       0.89      0.76      0.81     85443
weighted avg       1.00      1.00      1.00     85443
```

Undersampling

I will be using undersampling to control for the unbalanced data. Cases of the majority type (non-fraudulent transactions) will be removed to create a sample of data where the number of non-fraudulent indicies is equal to the number of fraudulent indicies.

In [11]:

```
# Count fraud cases
number_records_fraud = len (df[df.Class==1])
# Indicies extracted where the cases of fraud and non-fraud are
fraud_indices = np.array (df[df.Class==1].index)
normal_indices = np.array (df[df.Class==0].index)
```

In [12]:

```
# Randomly generating a number of non-fraud indicies that is equal to the number of fra
random_normal_indices = np.random.choice (normal_indices, number_records_fraud, replace
# arrays combined
undersample_indices = np.concatenate ([fraud_indices, random_normal_indices])
```

In [13]:

```
undersample_df = df.iloc[undersample_indices,:]
# separate x and y again
X_undersample = undersample_df.iloc [:, undersample_df.columns != 'Class'];
y_undersample = undersample_df.iloc [:, undersample_df.columns == 'Class'];
```

In [14]:

```
#sorting train test split
X_train, X_test, y_train, y_test = train_test_split (X,y, test_size = 0.3, random_state
X_train_under, X_test_under, y_train_under, y_test_under = train_test_split (X_undersam
```

In [15]:

```
#Using Multi-Layer Perceptron Classifier Neural Network
MLPC = MLPClassifier(hidden_layer_sizes=(400,), max_iter=10000)
scaler.fit(X_train_under)
scaler.fit(y_train_under)
scaler.fit(X_test_under)
MLPC.fit(X_train_under, y_train_under)
X_train = scaler.transform(X_train_under)
y_pred = MLPC.predict(X_test_under)
# Recall values
recall_acc = recall_score (y_test_under,y_pred)
recall_acc
```

```
C:\Users\abbie\anaconda3\lib\site-packages\sklearn\neural_network\_multilaye
r_perceptron.py:1109: DataConversionWarning: A column-vector y was passed wh
en a 1d array was expected. Please change the shape of y to (n_samples, ), f
or example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[15]:

```
0.9251700680272109
```

In [16]:

```
print(confusion_matrix(y_test_under, y_pred))
print(classification_report(y_test_under, y_pred))
```

```
[[144   5]
 [ 11 136]]
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       149
           1       0.96      0.93      0.94       147

    accuracy                           0.95       296
   macro avg       0.95      0.95      0.95       296
weighted avg       0.95      0.95      0.95       296
```

In [17]:

```
def pr(y_test_under, y_pred):
    precision, recall, thresholds = metrics.precision_recall_curve(y_test_under, y_pred
    auc = metrics.auc(recall, precision)

    print('Precision: %s' % precision)
    print('Recall: %s' % recall)
    print('Thresholds: %s' % thresholds)
    print('Naive auPRC: %s' % auc)
    print('Average Precision Score auPRC: %s' % metrics.average_precision_score(y_test_

pr(y_test_under, y_pred)
```

```
Precision: [0.49662162 0.96453901 1.        ]
Recall: [1.         0.92517007 0.        ]
Thresholds: [0 1]
Naive auPRC: 0.9634356186407859
Average Precision Score auPRC: 0.92952478096855
```

In [18]:

```
print(f"Mean Squared Error: {mean_squared_error(y_test_under, y_pred, sample_weight=Nor
print(f"Accuracy: {accuracy_score(y_test_under, y_pred, normalize=True, sample_weight=N
print(f"Precision: {average_precision_score(y_test_under, y_pred, average='macro', pos_
print(f"Recall: {recall_score(y_test_under, y_pred, labels=None, pos_label=1, average='
print(f"Receiver Operating Curve: {roc_curve(y_test_under, y_pred, pos_label=None, samp
```

```
Mean Squared Error: 0.05405405405405406
Accuracy: 0.9459459459459459
Precision: 0.92952478096855
Recall: 0.9251700680272109
Receiver Operating Curve: (array([0.        , 0.03355705, 1.        ]), arra
y([0.        , 0.92517007, 1.        ]), array([2, 1, 0], dtype=int64))
```