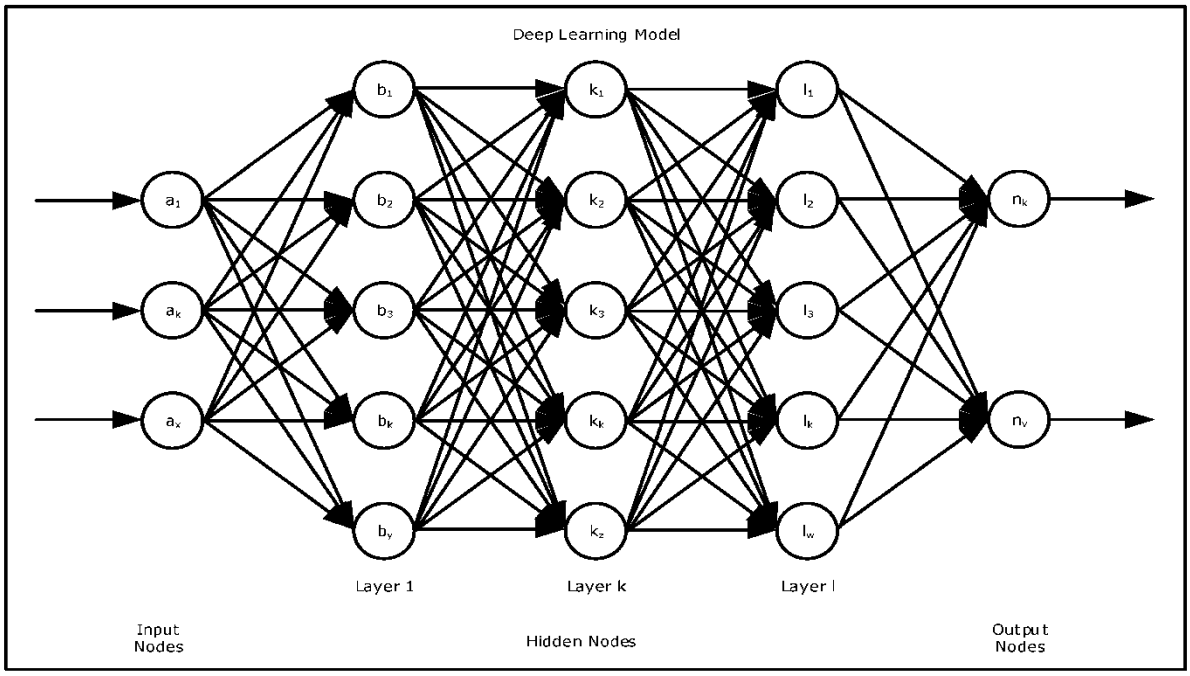
Deep Neural Network

Project Report 2021

Wande B. Adeyeye

DNN 2021

DEEP NEURAL NETWORK



CREDIT CARD FRUAD DETECTOR CLASSIFICATION

BY:

WANDE B. ADEYEYE

E-MAIL:

ADEYEYEWANDE@GMAIL.COM

QUESTIONS

Problem Statement

Fraud transactions or fraudulent activities are significant issues in many industries like banking, insurance, etc. Especially for the banking industry, credit card fraud is a pressing issue to resolve. These industries suffer due to fraudulent activities towards revenue growth and lose customer's trust. So, these companies need to find fraud transactions before it becomes a huge problem for them. Unlike the other machine learning problems, in credit card fraud detection the target class distribution is not equally distributed. It is popularly known as the class imbalance problem or unbalanced data issue; this makes this problem even more challenging to solve. Detecting fraud transactions is of great importance for any company because the cases of fraud grows largely annually. It's the responsibility of companies to look out for the safety of their customer and to detect potential frauds so that customers are protected and not charged for items that they did not purchase.

Goal

After the problem have been stated and understood, the next step is to set the goal to solve the problem and in this case is to build a classifier that tells if a transaction is fraudulent or not. To do that, the next thing we're going to do is gather useful data that will help us answer the problem then we will build our credit card fraud detection using different Machine Learning classification algorithms Model such as Decision Tree and Random Forest algorithm and Deep Neural Network, Comparison and Sampling Techniques to Improve Performance in the other below

- Data review
- Data processing
- DNN Model building
- DNN Model evaluation
- Decision Tree
- Random Forest
- Sampling
- Summary

About the Data

The dataset is the Kaggle Credit Card Fraud Detection dataset here. It contains transactions history by cardholders which the details have been anonymized. The dataset contains 492 frauds out of 284,807 transactions. Thus, it is highly unbalanced, with the positive (frauds) accounting for only 0.17%. Looking at the data, it only contains numerical variables. Features V1, V2, … V28 are the principal components obtained with PCA transformation. The only features which have not been transformed are 'Time' and 'Amount'. 'Time' is the seconds elapsed between each transaction and the first. 'Amount' is the transaction amount. 'Class' is the response variable with 1 as fraud and 0 otherwise. The dataset contains no null values.

Understanding the need to solve fraudulent Problem

Let's begin the discussion by understanding why we need to find fraudulent transactions/activities in any industry.

For many companies, fraud detection is a big problem because they find these fraudulent activities after they experience high loss. Fraud activities happen in all industries. We can't say only particular companies/industries suffer from these fraudulent activities or transactions. But when it comes to financial-related companies, this fraud transaction becomes more of an issue/problem. So, these companies want to detect fraud transactions before the fraud activities turn into significant damage to their company.

In the current generation, with high-end technology, still, on every 100 credit card transactions, according to the Nilson report, global card losses are expected to exceed $35 billion by 2020.

Reports of credit card fraud have skyrocketed in recent years. To best solve this problem, it's important to understand the main sources of credit card fraud and how they have changed over time.

Annual study highlights the growth of credit card fraud from various sources, the cost of fraud to consumers, businesses, and the government.

The FTC report shows credit card fraud continues to be one of the fastest-growing forms of identity theft. Reports of credit card fraud jumped by 107% from Q1 2019 to Q4 2020. To put this rapid growth in perspective, the number of card fraud reports between Q1 2017 and Q1 2019 grew by only 27%. The opening of new credit card accounts was the second-most reported type of credit card fraud in 2020, with 393,000 cases reported. This is an increase of 60% from 2019's 246,000 cases reported.

Using funds recovered in fraud cases, the FTC returned $483 million to victims of fraud and identity theft in 2020, an increase of 108% from 2019's $232.2 million. In total, the FTC has returned an astonishing $11.1 billion since July 2018 to consumers who were victims of some type of fraud.

Cost of Fraud

Fraud and identity theft are not only costly to the victims, but also to banks and payment networks that issue refunds to consumers. Government agencies, such as the FTC, also expend significant resources to investigate and litigate fraudulent companies so that victims can recover some or all of their money.

The amount of money lost by victims can vary greatly depending on the method of fraud, the creditworthiness of the victim, and even the age of the victim. In 2020, victims aged 80 and over only accounted for only 18% of reported cases that involved a dollar loss, but lost an average of $1,300 per case, the highest average of any age range.

The highest percentage of total fraud reports, 18%, came from those aged 60–69. This age range lost a total of $290 million to fraud and identity theft in 2020 alone, the highest total for any age range. Aging populations should continue to stay alert as they shop online or answer the phone. Foe more info check the link here

Data Pre-processing

Dataset has 284807 rows and 31 features. The result of the shape variable is a tuple that has the number of rows, number of columns of the dataset. We can see how the dataset looks like using the head (). The target variable Class has 0 and 1 values. Here

- 0 for non-fraudulent transactions
- 1 for fraudulent transactions

Because we aim to find fraudulent transactions, the dataset's target value has a positive value for that. we have 284315 non-fraudulent transaction samples & 492 fraudulent transaction samples.

Feature Scaling

The variable 'Amount' ranges from 0 to 25,691.16. To reduce its wide range, See the values of the Amount feature values are in high range compared to other feature values. We will change values within a smaller range .we use Standardization to remove the mean and scale to unit variance, so that 68% of the values lie in between (-1, 1) and drop the original amount column. If you want more details on this topic, read this article here. We will also drop the time column because we don't need it at the moment. Note we have to use ".values.reshape(-1, 1)" to convert a Series to an array and reshape to a 2D array.

Assign dataset to X and y

Now let's assign the data into independent variable X holding the features and dependent variable y holding target variable. Now we need to split the whole dataset into train and test dataset. Training data is used at the time of building the model and a test dataset is used to evaluate trained models.

By using the train_test_split method from the sklearn library we can do this process of splitting the dataset to train and test sets. 30% test and 70% train

Now our dataset is ready for building models. Let's jump to the development of the model using machine learning algorithms such as decision tree and random forest classification algorithms from the sklearn module.

DNN Model building

Here we will build a 5-layer deep neural networking using the Sequential model in Keras. For the 1st hidden layer, 'input_dim' is the number of input variables. 'units' is the number of nodes or neurons in each layer.

We use Rectified Linear Unit (ReLU) as an activation function for the hidden layers. ReLU normally performs better than Sigmoid and Hyperbolic Tangent functions when building deep neural networks. This is because Sigmoid and Tanh tends to saturate when the input value is either too large or too small. In addition, they only show a high gradient around their mid-points, such as 0.5 for sigmoid and 0 for tanh.

We use the Sigmoid function in the output layer for a binary classification problem. because our output will either be '0' or '1', that's why we used sigmoid activation function instead of 'relu' because we want the output to be saturated.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 16)                480
_________________________________________________________________
dense_6 (Dense)              (None, 24)                408
_________________________________________________________________
dropout_1 (Dropout)          (None, 24)                0
_________________________________________________________________
dense_7 (Dense)              (None, 20)                500
_________________________________________________________________
dense_8 (Dense)              (None, 24)                504
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 25
=================================================================
Total params: 1,917
Trainable params: 1,917
Non-trainable params: 0
_________________________________________________________________
```

From the above output, here is the summary of our model:

- It consists of 1917 parameters
- Dense layer and parameters
- Dropout layer and parameters
- Output layer and parameters

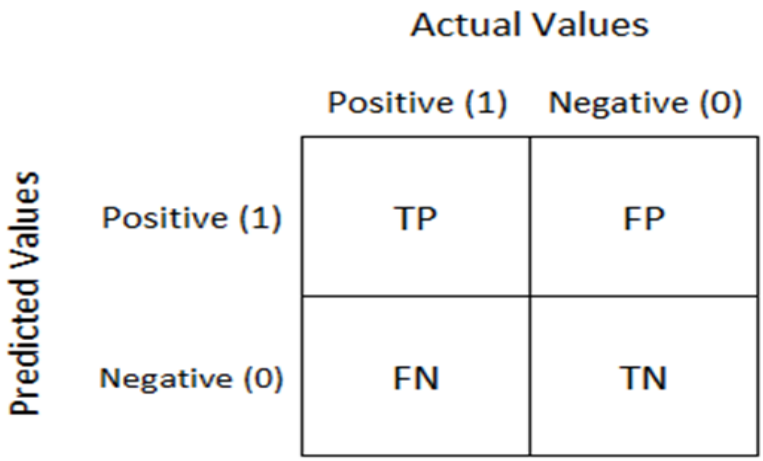
If we sum up the total number of the parameters it will add up to the number of parameters.

Next, we will compile our model and specifier optimizer as ‘Adam’, loss as 'binary_crossentropy', and metrics = ['accuracy'] after that we can fit our classifier with train dataset and specifier the number of epochs which is a hyperparameter that defines the number of times that the learning algorithm will work through the entire training dataset. DNN learn through experience, learn over time like humans, so every time we take the data and feed it to the network then we update the weights, we go back and repeat again and we keep repeating over and over again until the number of epochs.

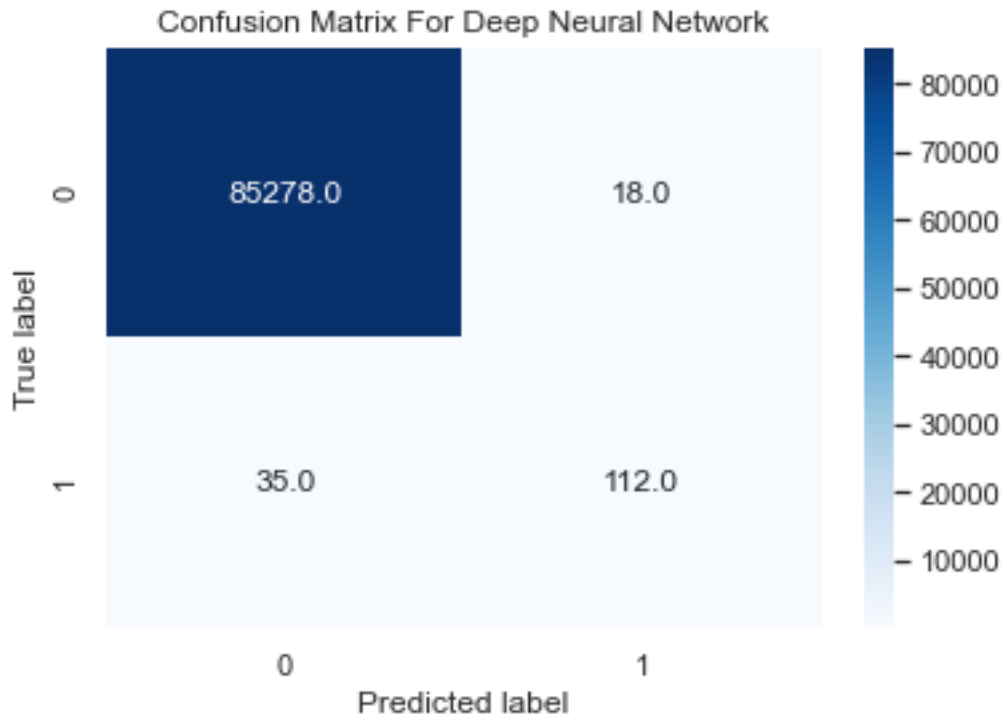
Great. Now let’s evaluate the model below:

```
2671/2671 [==============================] - 3s 1ms/step - loss: 0.0040 - accuracy: 0.9994
[0.004000866319984198, 0.9993796944618225]
```

The model is found with 99.9% accuracy but since we use just accuracy, and with imbalanced data. If we may recall, accuracy is the sum of Ture Negative and True Positive divided by total dataset size. If 95% of the dataset is Negative (non-frauds), the network will cleverly predict all to be Negative, leading to 95% accuracy. However, for fraud detection, detecting Positive matters more than detecting Negative. Therefore, we need better metrics. To get more out of our model we use confusion matrix to visualize the classification using test data. The confusion matrix displays the correctly predicted as well as incorrectly predicted values by a classifier. The sum of TP and TN, from the confusion matrix, is the number of correctly classified entries by the classifier. As shown in the figure below:



The confusion matrix for our classification model is in the fig below:



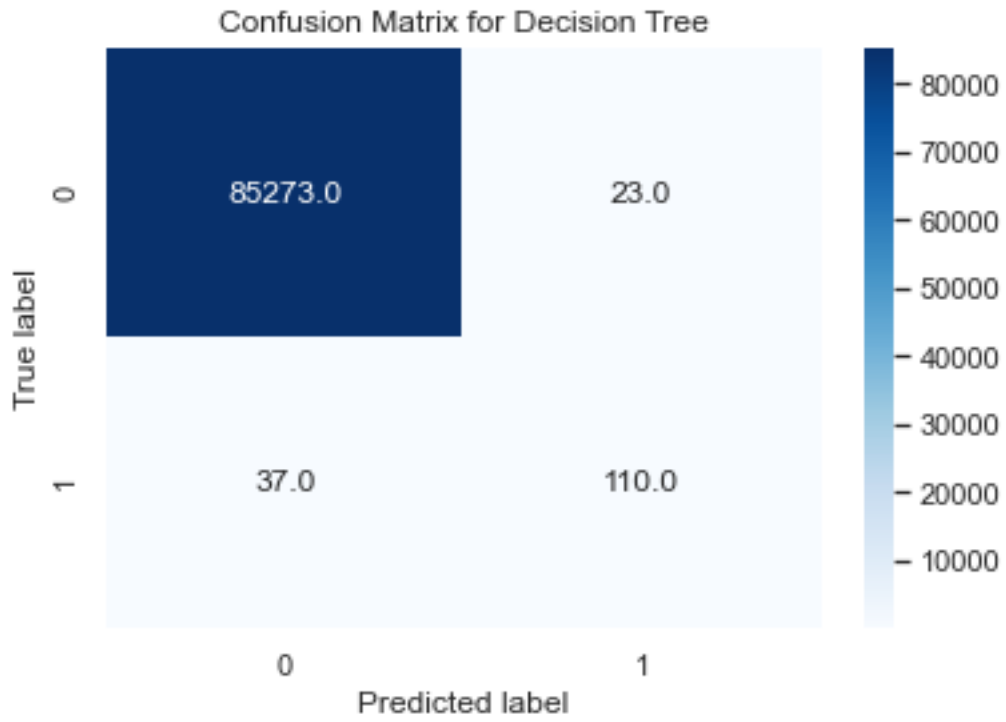
DNN shows a precision of 86%, a recall of 76%, and an F1 score of 80%. About 20% of frauds are misclassified as non-frauds, leading to these extra payments for the customers, though the accuracy is 99.94%. So, there is enough space to improve our model and we will be looking at other classification models

Decision Tree

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. weather if sunny or rainy), and each leaf node is a class label made after all tests. A decision tree aims to learn ways to split datasets based on conditions. So, it is non-parametric.

We will use the DecisionTreeClassifier class from the sklearn library to train and evaluate models. We use X_train and y_train data for training purposes. X_train is a training dataset with features, and y_train is the target label.Sampling

From the fig. shown below, the confusion matrix of the model. The decision tree gives a precision of 83%, a recall of 75%, and an F1 score of 79%, Our decision tree models is almost equal to the DNN model or sometimes less accurate



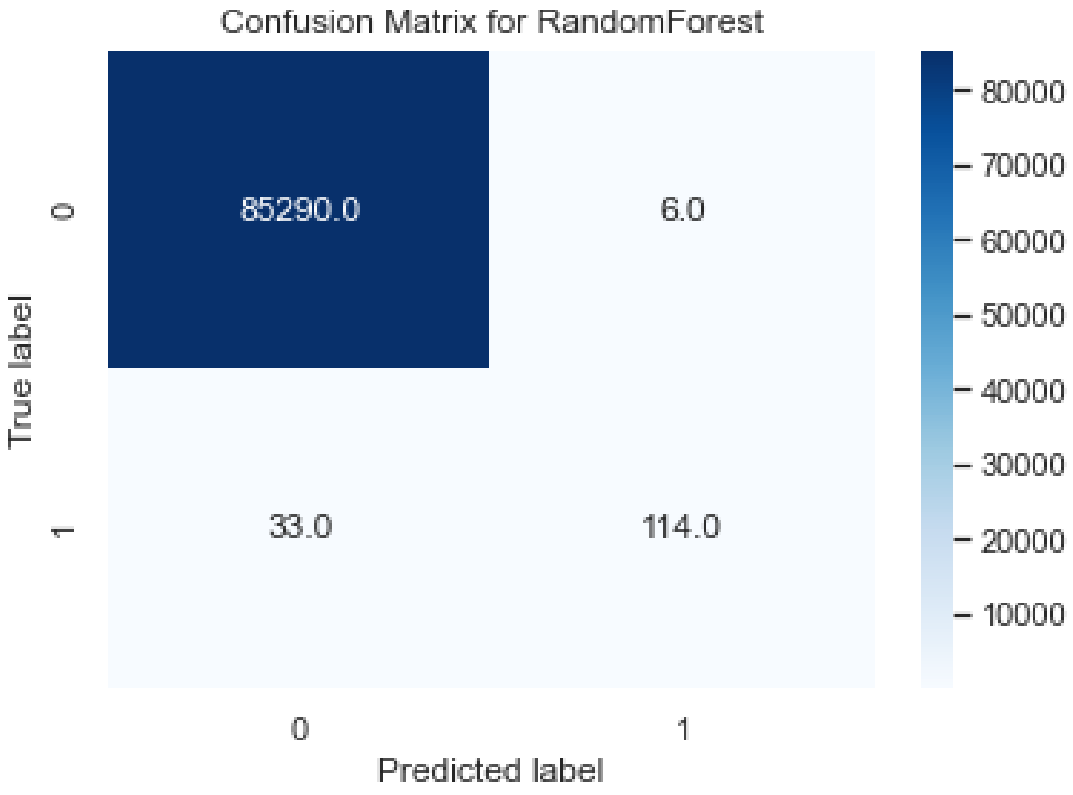
Random forest

Conceptually, a random forest (RF) is a collection of decision trees. Each tree votes a class and the class received the most votes is the predicted class. A decision tree is built on the whole dataset, while a random forest randomly selects features to build multiple decision trees and average the result. If you want to learn more about how RF works and parameter optimization, read this article.

Specifically,

Same as the above decision tree implementation, we use X_train and y_train dataset for training purposes and X_test for evaluation. Here we train the ensemble technique model of RandomForestClassifier from the sklearn. We can see the variations in the evaluation results.Random forest algorithm Implementation using sklearn library

From the fig. shown below, the confusion matrix of the model. The random forest gives a precision of 95%, a recall of 78%, and an F1 score of 85%, Our decision tree models is almost equal to the DNN model or sometimes less accurate



Model Improvement Using Sampling Techniques

Data sampling is the statistical method for selecting data points (here, the data point is a single row) from the whole dataset. In machine learning problems, there are many sampling techniques available.

Here we take undersampling and oversampling strategies for handling imbalanced data.

What is this undersampling and oversampling?

An example of a dataset that has nine samples.

Six samples belong to class-0,

Three samples belong to class-1

Oversampling = 6 class-0 samples x 2 times of class-1 samples of 3

Undersampling = 3 Class-1 samples x 3 samples from Class-0

Here what we are trying to do is the number of samples of both target classes to be equal.

In the oversampling technique, samples are repeated, and the dataset size is larger than the original dataset.

In the undersampling technique, samples are not repeated, and the dataset size is less than the original dataset.

The simplest strategy is to randomly select the majority class to balance with the minority class. But the limitation is that data randomly removed from the majority class may be useful to create a robust model.

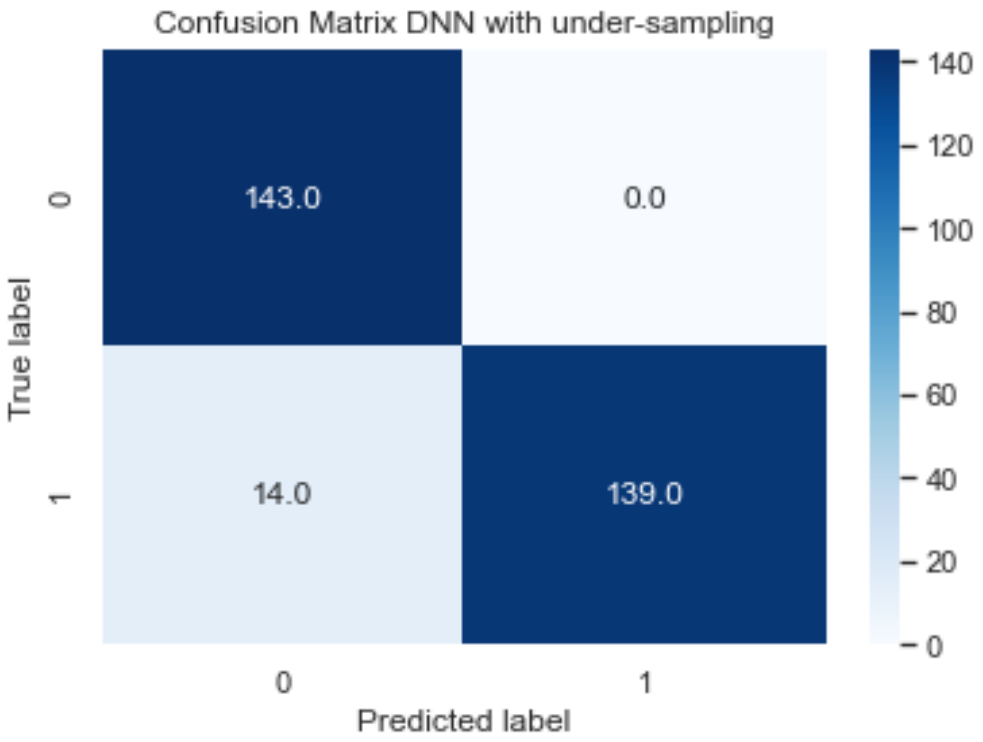
Let's implement this strategy first. Specifically,

An issue about the dataset we have here is a class imbalance. Only 0.17% of 284,807 transactions are frauds. Sadly, the model is more sensitive to detect the majority class than the minority class. In general, there are two techniques to tackle class imbalance, under-sampling, and over-sampling.

Under-sampling

The simplest strategy is to randomly select the majority class to balance with the minority class. But the limitation is that data randomly removed from the majority class may be useful to create a robust model.

Let's implement this strategy first. Specifically,



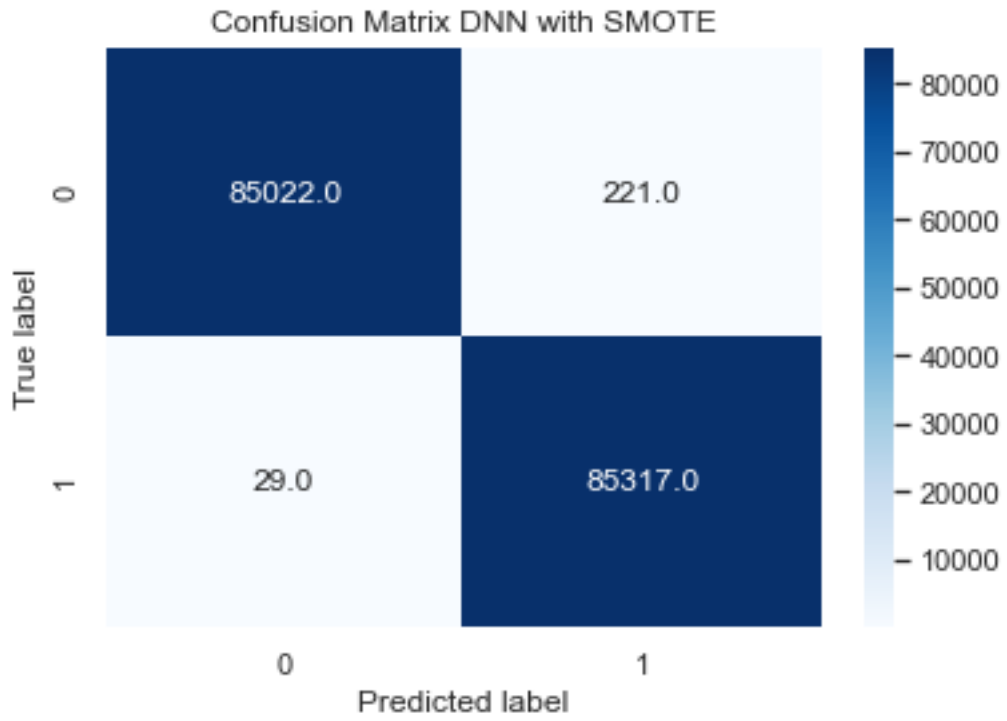
Above, we randomly selected the same amount of non-frauds as the fraud and created a new dataset. With the down-sized data, re-train the DNN model. In the end, under-sampling gives a precision of 100%, a recall of 90%, and an F1 score of 95%. Much better than DNN without under-sampling

SMOTE

The simplest way of over-sampling is to duplicate data in the minority class, but no new information will be added to the model. Alternatively, we can synthesize data from existing ones, referred as Synthetic Minority Over-sampling, or SMOTE for short.

SMOTE, initially presented by Nitesh Chawla in 2002, works by selecting data that are close or similar in the feature space and drawing a line between data and make new data at a point on the line. It is effective because new data are close to the minority class in the feature space. If you want to learn about SMOTE, please feel free to read Nitesh's article.

Now, let's implement SMOTE and re-train the model. Specifically,



The above code created a balanced 'y_resample' with 284,315 frauds and 284,315 non-frauds. After re-training the DNN model, SMOTE gives a precision of 99%, a recall of 99%, and an F1 score of 99%. Much better than DNN with and without under-sampling

Summary

We created 5 models, DNN, Decision Tree, Random Forest, DNN with under-sampling, and DNN with SMOTE. As shown in the Table below, DNN with SMOTE performs best.

	Pression (%)	Recall (%)	F1 (%)
DNN	86	76	80
Decision Tree	83	75	79
Random Forest	95	78	85
DNN with under-sampling	100	90	95
DNN with SMOTE	99	99	99

Useful links:

Dataset link here

More info on fraud cases here

More on sampling here

More on SMOTE link here

My Github profile here