

Credit card fraud detection

Presently a day's online exchanges have become a significant and vital piece of our lives. It is crucial that credit card companies can distinguish false credit card transactions with the goal that clients are not charged for things that they didn't buy. As recurrence of transactions are expanding, number of fraudulent transactions are additionally expanding quickly. Such issues can be handled with Machine Learning with its algorithms. This project intends to illustrate the modelling of a data set using machine learning with Credit Card Fraud Detection. The Credit Card Fraud Detection Problem incorporates displaying past charge card exchanges with the information of the ones that ended up being misrepresentation. This model is then used to perceive whether another exchange is fake or not. Our goal here is to distinguish 100% of the fraudulent transactions while limiting the incorrect fraud classification. In this process, we have focused on analyzing and pre-processing data sets as well as the deployment of multiple anomaly detection algorithms such as Local Outlier Factor and Isolation Forest algorithm on the PCA transformed Credit Card Transaction data.

About dataset

Worked on Kaggle credit card fraud detection dataset which contained 31 parameters (amount, time, class, V1-V28). The datasets contain 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. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, the original features and more background information about the data is not provided. 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.

Machine Learning-Based Approaches

Density-Based Anomaly Detection

Density-based outlier detection method investigates the density of an object and that of its neighbors. Here, an object is identified as an outlier if its density is relatively much lower than that of its neighbors. The nearest set of data points are evaluated using a score, which could be Euclidian distance or a similar measure dependent on the type of the data (categorical or numerical). They could be broadly classified into two algorithms:

K-nearest neighbor: k-NN is a simple, non-parametric lazy learning technique used to classify data based on similarities in distance metrics such as Euclidian, Manhattan, Minkowski, or Hamming distance.

Relative density of data: This is better known as local outlier factor (LOF). This concept is based on a distance metric called reachability distance.

Clustering-Based Anomaly Detection

Clustering is one of the most popular concepts in the domain of unsupervised learning.

Assumption: Data points that are similar tend to belong to similar groups or clusters, as determined by their distance from local centroids.

K-means is a widely used clustering algorithm. It creates 'k' similar clusters of data points. Data instances that fall outside of these groups could potentially be marked as anomalies.

Support Vector Machine-Based Anomaly Detection

- A support vector machine is an effective technique for detecting anomalies which is usually associated with supervised learning, but sometimes unsupervised problems can also be detected (OneClassCVM).
- The algorithm learns a soft boundary in order to cluster the normal data instances using the training set, and then, using the testing instance, it tunes itself to identify the abnormalities that fall outside the learned region.
- Depending on the use case, the output of an anomaly detector could be numeric scalar values for filtering on domain-specific thresholds or textual labels (such as binary/multi labels).

In PyCharm we are going to take the credit card fraud detection as the case study for understanding this concept in detail using the following Anomaly Detection Techniques name

- **Isolation Forest Anomaly Detection Algorithm.**
- **Density-Based Anomaly Detection (Local Outlier Factor) Algorithm.**

Methodology

- First, we obtained our dataset from Kaggle, a data analysis website which provides datasets.
- Platform used in this project was PyCharm.
- Importing of the necessary libraries with the packages

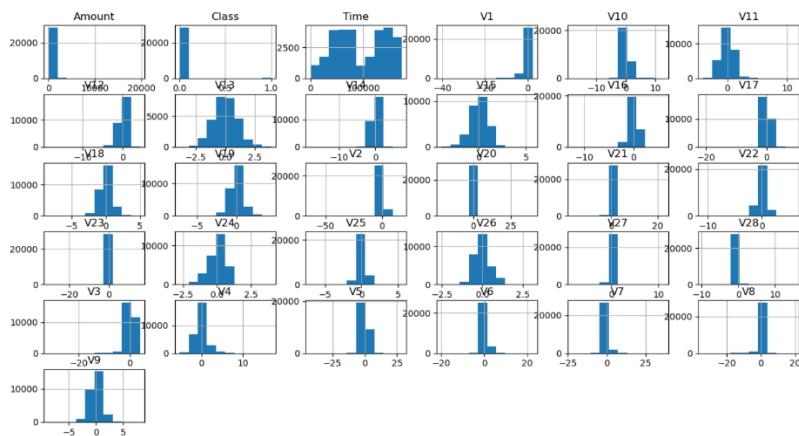
```
import pickle
import sys
import numpy
import matplotlib
import pandas
import scipy
import seaborn
import sklearn
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

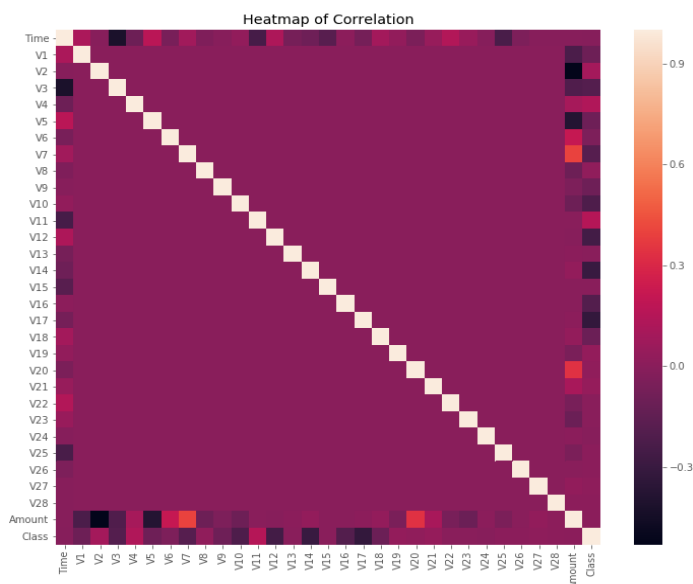
- loading the dataset

```
data = pd.read_csv('creditcard.csv')
```

- After checking this dataset, a histogram is plotted for every column. This is done to get a graphical representation of the dataset which can be used to verify that there are no missing any values in the dataset. This is done to ensure that we don't require any missing value imputation and the machine learning algorithms can process the dataset smoothly.



- After this analysis, a heatmap is plotted to get a colored representation of the data and to study the correlation between out predicting variables and the class variable. Here the darker ones denote positive correlation while lighter denotes negative. This heatmap is shown below:



- No of fraud cases and valid cases were identified

```
Fraud = data[data['Class'] == 1]
Valid = data[data['Class'] == 0]
```

- An outlier fraction was used to calculate the fraudulent cases to the valid cases.

```
outlier_fraction = len(Fraud) / float(len(Valid))
print(outlier_fraction)
```

- getting all the columns from the dataset

```
columns = data.columns.tolist()
```

- filter the columns to remove the data we do not need.

```
columns = [c for c in columns if c not in ["Class"]]
```

- store the variable we'll be predicting on

```
target = "Class"
```

Applying the algorithm to the project

The data set has been preprocessed and is ready to be trained. Imported several packages using sklearn for isolation forest and local outlier factor.

Built two anomaly detection models :- Isolation Forest and Local Outlier Factor. . A comparison is made between 2 models.

Isolation forest: Isolation forest is a machine learning algorithm for anomaly detection. It's an unsupervised learning algorithm that identifies anomaly by isolating outliers in the data.

Local outlier factor: The Local Outlier Factor (LOF) algorithm is an unsupervised anomaly detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers as outliers the samples that have a substantially lower density than their neighbors.

The two algorithms are put into a dictionary of classifiers

```
classifiers = {
    "Isolation Forest": IsolationForest(max_samples=len(X),
                                         contamination=outlier_fraction,
                                         random_state=state),
    "Local Outlier Factor": LocalOutlierFactor(
```

```

n_neighbors=20,
contamination=outlier_fraction)
}

```

- performance of both algorithms was considered isolation forest was identified as better.

Total number of errors = 107. Accuracy = 99.62%. the precision and recall are lower in local outlier factor.

Total number of errors = 77. Accuracy = 99.73%. precision and recall are bigger in isolation forest than that of local outlier factor.

Storing the model

- pickle was used to store the model. Pickle is used for serializing and deserializing python object structures.

```

•pickle.dump(clf, open('model.pkl', 'wb'))
model = pickle.load(open('model.pkl', 'rb'))

```

- need to use 'wb' ('b' for binary) during file writing and 'rb' during file opening.

Building the web application using flask

A html page was created with the style sheet. colorlib was used to download a free html template with the style sheet.

In order to demonstrate the use of POST method in URL routing, i created an HTML form and use the POST method to send form data to a URL.

To build the web page the framework used was flask. flask makes the process of designing a web application simpler. Flask lets us focus on what the users are requesting and what sort of response to give back.

```

@app.route('/')
def home():
    return render_template('index.html')

```

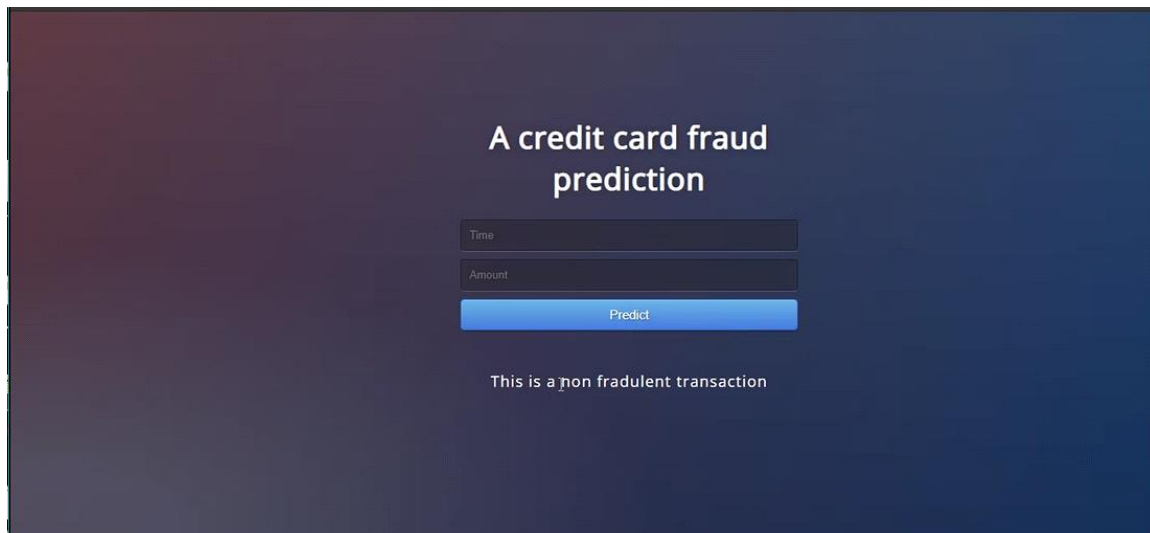
This is the default function which gets triggered whenever the url is open.

```
if __name__ == "__main__":  
    app.run(debug=True)
```

Once you run this you should see the basic app in action on <http://localhost:5000/>.

Next form data (amount, time) is gathered and is sent through an if else statement to return the output (transaction valid or fraud).

Finally the result will be as below:



The screenshot shows a web application interface with a dark blue gradient background. At the top, the text "A credit card fraud prediction" is displayed in white. Below this, there are two input fields: "Time" and "Amount", both with light blue borders. A blue button with the text "Predict" is positioned below the input fields. At the bottom, the text "This is a non fraudulent transaction" is displayed in white.