Mount Zion college of Engineering and Technology

Pudukkottai, Tamil Nadu

IBM Naan Muthalavan

Applied Data Science

Credit Card Fraud Detection

By

Sriramnehru.M

Varatharajan.P.S

Srisaran.M

Nandhakumar.S

SivaGuru.R

Credit Crad Fraud Detection

Phase 4:

In this technology you will continue building our project by performing feature engineering, model training and evaluation. Perform different analysis as needed.

Problem statement :

The project aims to develop the machine learning-based system that analyzes transaction data in real time,effectively detecting credit card fraud while minimizing false positives.This solution will help financial institutions protect against fraudulent transactions,reducing financial losses and ensuring customer trust.

Introduction:

Credit card fraud poses a significant threat in today's digital age, where financial transactions occur seamlessly across the globe. Detecting and preventing fraudulent activities is paramount for both financial institutions and consumers. In this context, the development and implementation of robust credit card fraud detection systems have become crucial. This introduction will delve into the pressing need for such systems, the methods employed to identify and combat fraud, and the ever-evolving landscape of fraudulent activities that necessitate continuous innovation in this field. in the project.

Dataset: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Training and evaluating on a dataset involves several steps:

1. Data Preparation:

Load your dataset using libraries like Pandas.Preprocess the data by handling missing values, encoding categorical variables, and scaling/normalizing numerical features.

```
python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

data = pd.read_csv('your_dataset.csv')

# Data preprocessing
X = data.drop('target_column', axis=1)
y = data['target_column']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Standardize/normalize data (if needed)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

2. Select a Machine Learning Algorithm:

Choose an appropriate algorithm for your task.

python

from sklearn.linear_model import LinearRegression

model = LinearRegression() # Example for regression

3.Training the Model:

Fit the model to the training data.

python

model.fit(X_train, y_train)

4.Evaluating the Model:

Use the testing data to evaluate the model's performance.

python

For regression

from sklearn.metrics import mean_squared_error, r2_score

```python
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# For classification
from sklearn.metrics import accuracy_score, classification_report

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(report)
```

5.Cross-Validation:

To get a more reliable estimate of your model's performance, you can perform k-fold cross-validation.

Python

```python
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
print("Cross-Validation Scores:", scores)
print("Mean CV Score:", scores.mean())
```

6.Hyperparameter Tuning:

If your model has hyperparameters, you can optimize them using techniques like Grid Search or Random Search.

```python
from sklearn.model_selection import GridSearchCV

param_grid = {'parameter_name': [value1, value2, ...]}
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X, y)

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
```

7.Deployment:

Once you are satisfied with the model's performance, you can deploy it for predictions on new data.

Different analysis :

1. Descriptive Statistics:

Use Pandas for basic summary statistics like mean, median, standard deviation.

```Python
import pandas as pd
data = pd.read_csv('your_dataset.csv')
summary_stats = data.describe()
```

2. Data Visualization :

Matplotlib and Seaborn are popular libraries for creating data visualizations like histograms, scatter plots, and box plots.

```
Python
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")
sns.scatterplot(x="x_column", y="y_column", data=data)
plt.show()
```

3. Hypothesis Testing:

Use libraries like SciPy for t-tests, ANOVA, and other statistical tests.

```
Python
from scipy import stats
t_stat, p_value = stats.ttest_ind(data_group1, data_group2)
```

4. Regression Analysis:

Scikit-Learn is a powerful library for linear regression, logistic regression, and other regression models.

```
python
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
model.fit(X, y)
predictions = model.predict(X)
```

5. Classification Analysis:

Scikit-Learn is also suitable for classification tasks. You can use classifiers like Decision Trees, Random Forests, and Support Vector Machines.

```
Python
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier()
classifier.fit(X_train, y_train)
```

6. Clustering Analysis:

Scikit-Learn provides clustering algorithms like K-Means and DBSCAN for unsupervised analysis.

```
python
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

7. Principal Component Analysis (PCA):

Scikit-Learn can be used for dimensionality reduction using PCA.

```
python
from sklearn.decomposition import PCA
```

```
  pca = PCA(n_components=2)
  pca.fit(X)
  reduced_data = pca.transform(X)
```

Program:

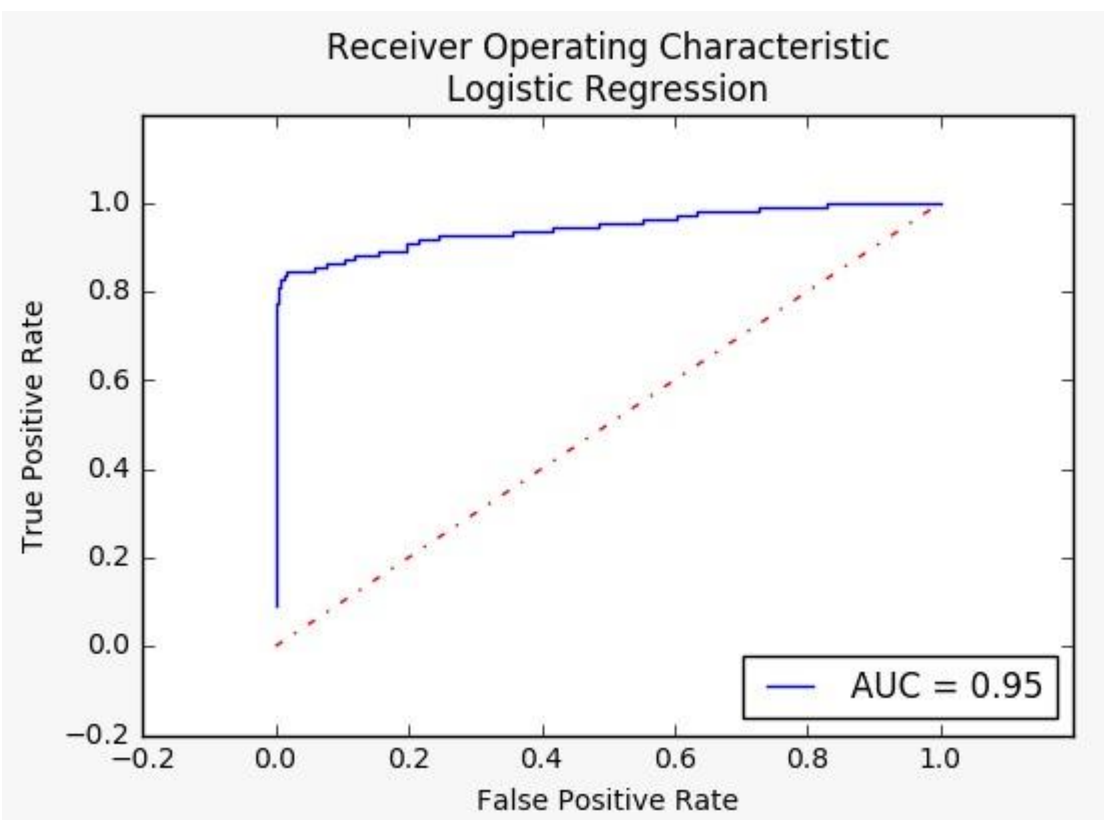
```
import pandas as pd
import numpy as np

#importing the data set
df=pd.read_csv("Kaggle/Credit Card Fraud/Data set.csv")

#creating target series
target=df['Class']
target

#dropping the target variable from the data set
df.drop('Class',axis=1,inplace=True)
df.shape

#converting them to numpy arrays
X=np.array(df)
y=np.array(target)
X.shape
y.shape

#distribution of the target variable
len(y[y==1])
```

```
len(y[y==0])

#splitting the data set into train and test (75:25)
from sklearn.cross_validation import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=1)
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)

#applyting SMOTE to oversample the minority class
from imblearn.over_sampling import SMOTE
sm=SMOTE(random_state=2)
X_sm,y_sm=sm.fit_sample(X_train,y_train)
print(X_sm.shape,y_sm.shape)
print(len(y_sm[y_sm==1]),len(y_sm[y_sm==0]))

from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
from sklearn import metrics

#Logistic Regression
logreg=LogisticRegression()
logreg.fit(X_sm,y_sm)
y_logreg=logreg.predict(X_test)
y_logreg_prob=logreg.predict_proba(X_test)[:,1]

#Performance metrics evaluation
print("Confusion Matrix:\n",metrics.confusion_matrix(y_test,y_logreg))
print("Accuracy:\n",metrics.accuracy_score(y_test,y_logreg))
print("Precision:\n",metrics.precision_score(y_test,y_logreg))
```

```
print("Recall:\n",metrics.recall_score(y_test,y_logreg))
print("AUC:\n",metrics.roc_auc_score(y_test,y_logreg_prob))
auc=metrics.roc_auc_score(y_test,y_logreg_prob)

#plotting the ROC curve
fpr,tpr,thresholds=metrics.roc_curve(y_test,y_logreg_prob)
plt.plot(fpr,tpr,'b', label='AUC = %0.2f'% auc)
plt.plot([0,1],[0,1],'r-.')
plt.xlim([-0.2,1.2])
plt.ylim([-0.2,1.2])
plt.title('Receiver Operating Characteristic\nLogistic Regression')
plt.legend(loc='lower right')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

Output:

```
(213605, 30) (71202, 30) (213605,) (71202,)
(426448, 30) (426448,)
213224 213224
[[70155   936]
 [   19    92]]
0.986587455409
0.0894941634241
0.828828828829
0.945549423331
```



Receiver Operating Characteristic
Logistic Regression
True Positive Rate
False Positive Rate
AUC = 0.95
−0.2
0.0
0.2
0.4
0.6
0.8
1.0