Mount Zion college of Engineering and Technology

Pudukkottai, Tamil Nadu

IBM Naan Muthalavan

Applied Data Science

Credit Crad Fraud Detection

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 pressingneed 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.

Design Thinking

Data Source: Utilize a dataset containing transaction data, including features such as transaction amount, timestamp, merchant information, and card details.

Data Preprocessing: Clean and preprocess the data, handle missing values, and normalize features.

Feature Engineering: Create additional features that could enhance fraud detection, such as transaction frequency and amount deviations

Model Selection: Choose suitable machine learning algorithms (e.g., Logistic Regression, Random Forest, Gradient Boosting) for fraud detection.

Model Training: Train the selected model using the preprocessed data.

Evaluation: Evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.

The key components of credit card fraud detection systems include:

1. Data Collection: Gathering transaction data, including details like the transaction amount, merchant information, cardholder details, and timestamps, is the foundational step.

2. Data Preprocessing: Cleaning and formatting the data to ensure consistency and accuracy is crucial. This may involve handling missing values, normalizing data, and removing outliers.

3. Feature Engineering: Creating relevant features from the raw data to help machine learning models identify patterns and anomalies. Examples include calculating transaction frequency, time of day, or geographical distance between transactions.

4. Machine Learning Models: Utilizing various machine learning algorithms, such as logistic regression, decision trees, random forests, and neural networks, to analyze transaction data and identify fraudulent patterns.

5. Anomaly Detection: Employing anomaly detection techniques, like clustering or statistical methods, to flag transactions that deviate significantly from the norm.

6. Behavioral Analysis: Examining cardholder behavior and establishing a baseline for what constitutes normal spending patterns. Deviations from this baseline can indicate potential fraud.

7. Rules-Based Systems: Implementing predefined rules or heuristics to detect suspicious transactions based on specific criteria, such as transaction amount thresholds or geographic location rules.

8. Real-time Monitoring: Continuously monitoring transactions in real-time to identify and respond to suspicious activity as it occurs.

9. Scoring and Risk Assessment: Assigning risk scores to transactions or cardholders based on the likelihood of fraud. High-risk transactions trigger further investigation or denial.

10. Machine Learning Model Training and Updates: Regularly training machine learning models with new data to adapt to evolving fraud patterns and improve accuracy.

11. Geolocation and IP Analysis: Analyzing the location of transactions and IP addresses to detect potential discrepancies that might indicate fraud.

12. Customer Authentication: Employing multi-factor authentication methods, such as OTPs (one-time passwords) or biometrics, to verify the identity of cardholders during transactions.

13. Historical Data Analysis: Studying historical fraud data to identify trends and patterns that can inform the development of fraud detection models.

14. Communication and Alerts: Notifying cardholders and fraud prevention teams of suspicious activity through alerts, emails, or SMS messages.

15. Fraud Investigation Unit: Establishing a dedicated team to investigate and respond to suspected cases of fraud, ensuring prompt action when needed.

16. Continuous Improvement: Regularly reviewing and updating the fraud detection system to adapt to new fraud techniques and improve its accuracy.

Loading & Preprocessing of dataset

1. Import Necessary Libraries: Start by importing libraries such as NumPy, Pandas, and any machine learning frameworks like TensorFlow or PyTorch that you'll need.

2. Load the Dataset: Use functions provided by the libraries to load our dataset. For example, if you have a CSV file, can use `pandas.read_csv()` to load it into a DataFrame.

python

```
import pandas as pd

# Load a dataset from a CSV file
data = pd.read_csv('your_dataset.csv')
```

3. Exploratory Data Analysis (EDA):Perform some initial data exploration to understand the dataset's structure, missing values, and basic statistics.

python

```
# Display the first few rows of the dataset
print(data.head())

# Check for missing values
print(data.isnull().sum())

# Basic statistics
print(data.describe())
```

4. Data Cleaning: Handle missing data by either removing or imputing missing values.Can also handle outliers and data inconsistencies during this step.

5. Feature Selection/Engineering: Decide which features are relevant for your task. Can create new features or transform existing ones if needed.

6. Data Splitting: Split the dataset into training, validation, and test sets. This is important for model evaluation.

python

```
from sklearn.model_selection import train_test_split


X = data.drop('target_column', axis=1)

y = data['target_column']


X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)

X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

Feature engineering for credit card fraud detection

1. Transaction Amount: Create features based on the transaction amount, such as average transaction amount for a specific cardholder, standard deviation, or percentiles.

2. Time-Based Features: Extract time-related features, like the hour of the day, day of the week, or time since the last transaction.

3. Transaction Frequency: Calculate transaction frequency, such as the number of transactions in a certain time window.

4. Geographic Features: Incorporate location-based features, such as the country or city where the transaction occurred.

5. Cardholder Behavior: Analyze a cardholder's historical behavior, like the average transaction amount, frequency, or categories of transactions.

6. Merchant Information: Include features related to the merchant, such as the type of business, location, or frequency of transactions at that merchant.

7. Anomaly Detection Features: Apply anomaly detection techniques to identify unusual patterns or outliers in transaction data.

8. Aggregated Features: Create features by aggregating data over time periods, like daily, weekly, or monthly statistics for a cardholder.

9. User-Agent Data: Consider the user-agent string if available, which can provide information about the device used for the transaction.

10. Social Network Analysis: Analyze relationships between cardholders based on shared contact information or transaction history.

11. Cardholder Segmentation: Segment cardholders into groups based on their behavior and create features for each segment.

12. Time since Previous Fraud: Calculate the time elapsed since the last known fraudulent transaction for a cardholder.

13. Payment History: Explore features related to a cardholder's payment history, like missed payments or changes in spending patterns.

Training and Evaluating on a dataset

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

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

```
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 boxvisualization

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)
```

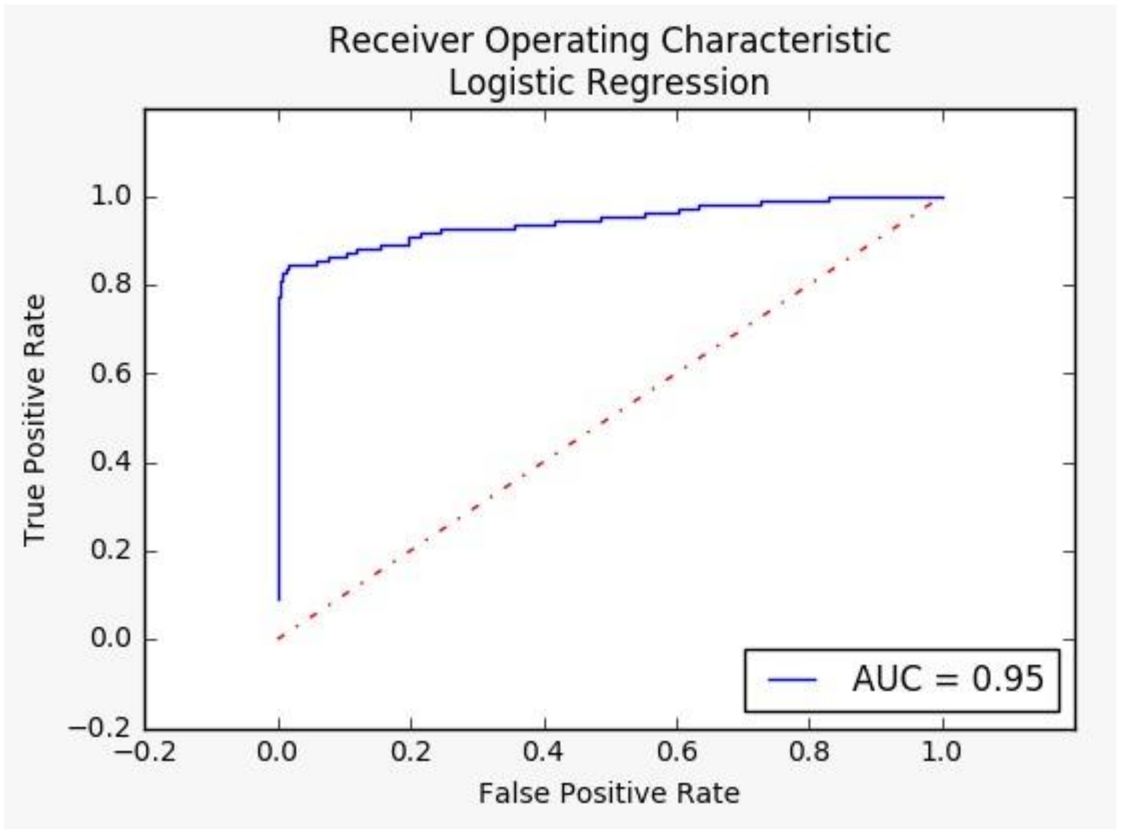
Program

```
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
```



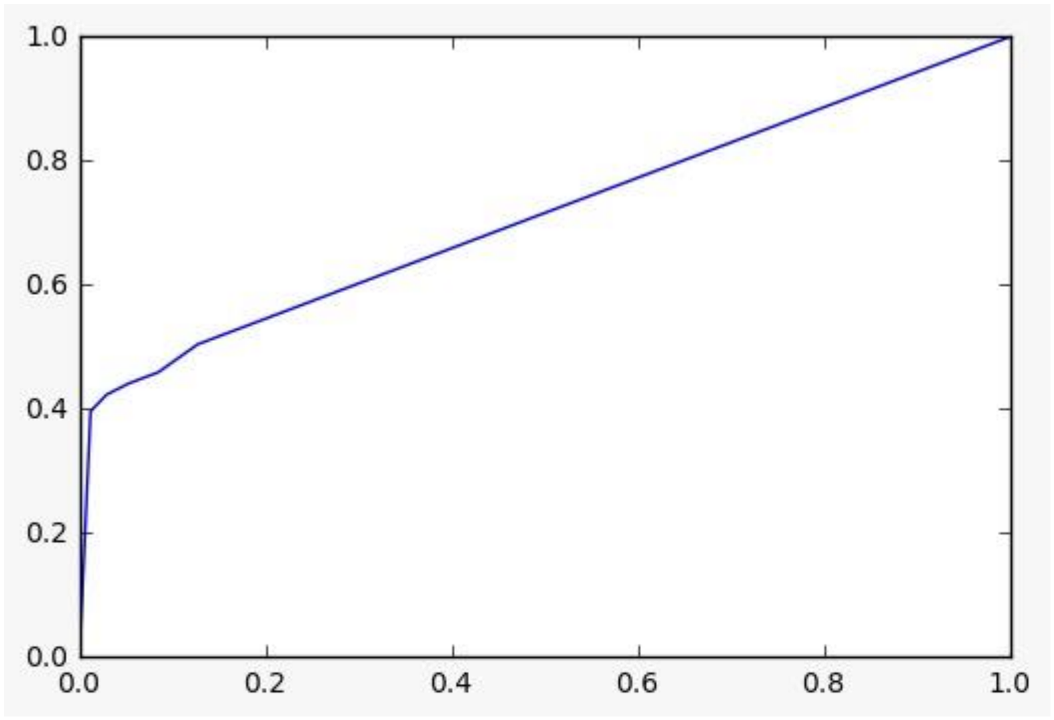
Program:

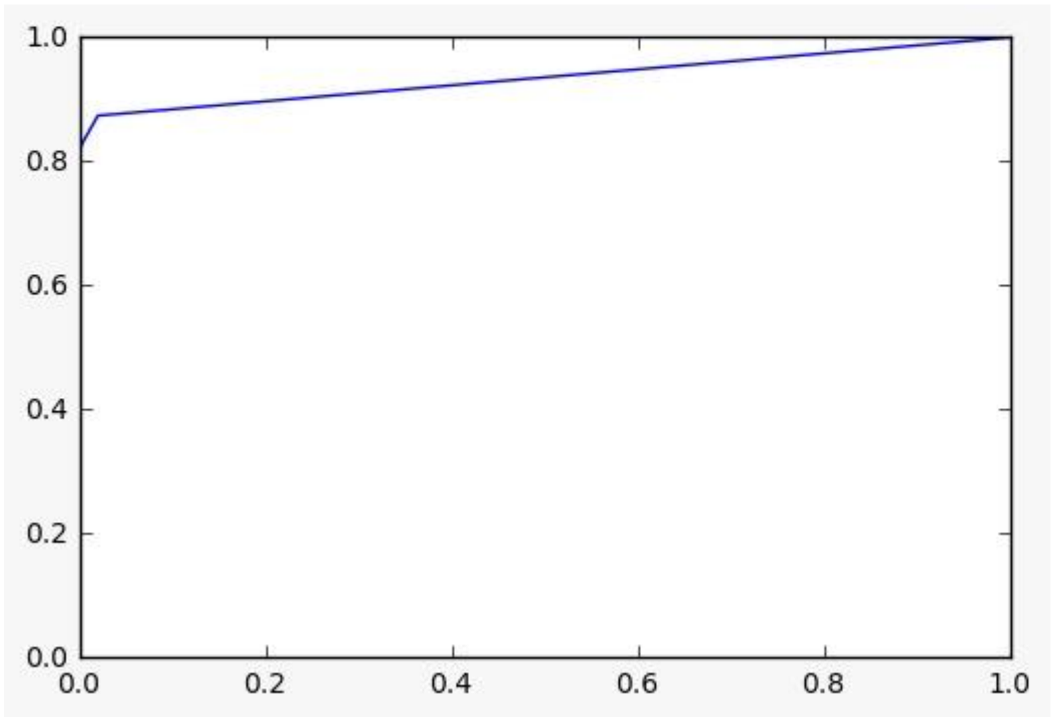
```
#K Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_sm,y_sm)
y_knn=knn.predict(X_test)
y_knn_prob=knn.predict_proba(X_test)[:,1]

#metrics evaluation
print(metrics.confusion_matrix(y_test,y_knn))
print(metrics.accuracy_score(y_test,y_knn))

print(metrics.precision_score(y_test,y_knn))

print(metrics.recall_score(y_test,y_knn))

print(metrics.roc_auc_score(y_test,y_knn_prob))


#plotting the ROC curve

fpr,tpr,thresholds=metrics.roc_curve(y_test,y_knn_prob)

plt.plot(fpr,tpr)

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.0])

plt.show()

[[67323  3768]

 [   62    49]]

0.946209376141

0.0128373067854

0.441441441441

0.7110521713
```

Output

```
[[71076    15]

 [   23    88]]

0.999466307126

0.854368932039

0.792792792793
```

```
0.935089856282
```

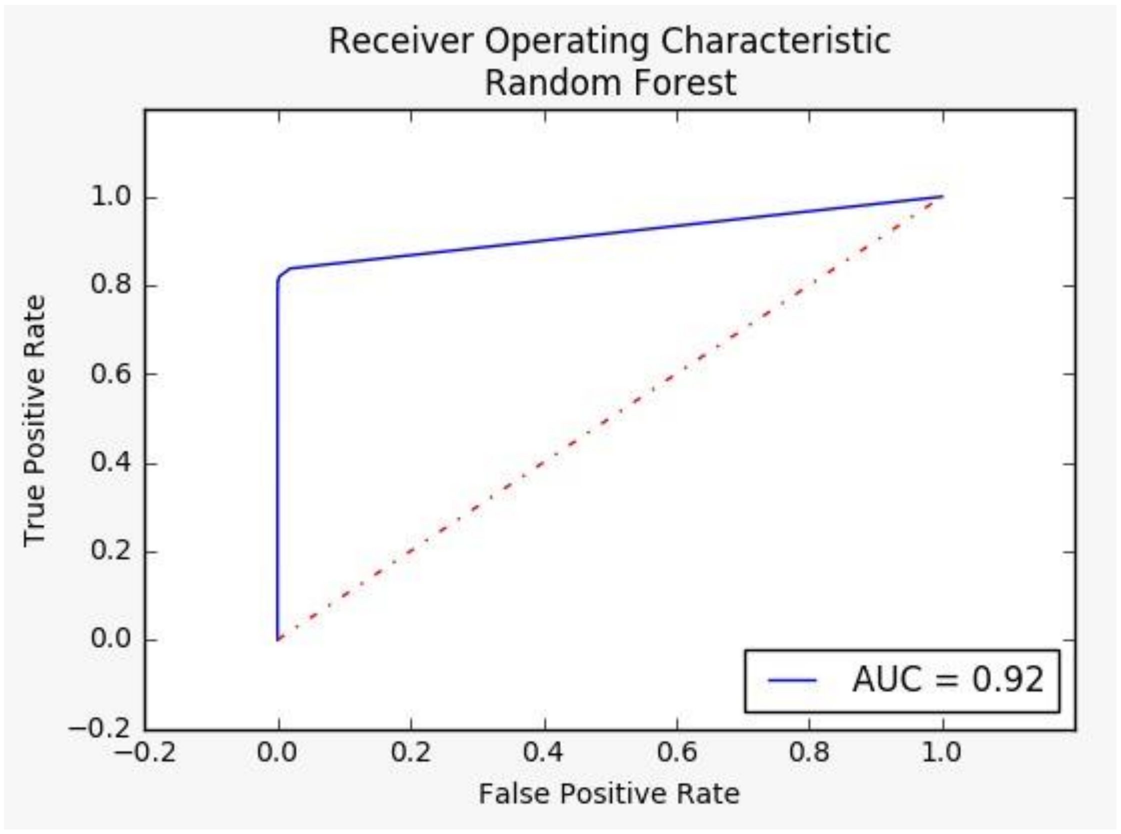




Program:

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(random_state=3)
rf.fit(X_sm,y_sm)
y_rf=rf.predict(X_test)
y_rf_prob=rf.predict_proba(X_test)[:,1]

#Performance metrics evaluation
print("Confusion Matrix:\n",metrics.confusion_matrix(y_test,y_rf))
print("Accuracy:\n",metrics.accuracy_score(y_test,y_rf))
print("Precision:\n",metrics.precision_score(y_test,y_rf))
print("Recall:\n",metrics.recall_score(y_test,y_rf))
print("AUC:\n",metrics.roc_auc_score(y_test,y_rf_prob))
auc=metrics.roc_auc_score(y_test,y_rf_prob)

#plotting the ROC curve
fpr,tpr,thresholds=metrics.roc_curve(y_test,y_rf_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\nRandom Forest')
plt.legend(loc='lower right')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
Output:

Confusion Matrix:

 [[71076    15]

 [   23    88]]

Accuracy:

 0.999466307126

Precision:

 0.854368932039

Recall:

 0.792792792793

AUC:

0.91715547678
```



One innovative technique for credit card fraud detection is using machine learning and artificial intelligence. These techniques can analyze a wide range of data, including transaction history, user behavior, and more, to identify patterns and anomalies that might indicate fraudulent activity. Additionally, advanced algorithms like deep learning and neural networks can be used to enhance the accuracy of fraud detection systems. Other approaches include behavioral biometrics, which use unique patterns of user behavior, and blockchain technology for secure and transparent transactions. Continuous research and development in this field are crucial to stay ahead of evolving fraud techniques.

Conclusion:

credit card fraud detection is an ongoing process, and be prepared to adapt to new fraud techniques and data challenges over.

By

Sriramnehru.M

Varatharajan.P.S

Srisaran.M

Nandhakumar.S

SivaGuru.R