Project Report: Credit Card Fraud Detection Using ANNs and XGBoost

Introduction

Credit card fraud detection is a critical issue for financial institutions, requiring efficient and accurate models to detect fraudulent transactions. This project aims to develop and compare two machine learning models, Artificial Neural Networks (ANNs) and XGBoost, to identify fraudulent credit card transactions. The dataset used contains anonymized transaction details, with features transformed using Principal Component Analysis (PCA).

Dataset Description

The dataset comprises transactions made by credit cards in September 2013 by European cardholders. It contains 284,807 transactions, of which 492 are fraudulent. The dataset is highly imbalanced, with the positive class (frauds) accounting for only 0.172% of all transactions. The dataset includes the following features:

- Time: The seconds elapsed between this transaction and the first transaction in the dataset.
- V1 to V28: The result of a PCA transformation on the original features. Due to confidentiality issues, the original features are not provided.
- Amount: The transaction amount.
- Class: The target variable, where 0 indicates a legitimate transaction and 1 indicates fraud.

Data Preprocessing

Preprocessing steps are crucial to ensure the data is clean and suitable for model training. These steps include handling missing values, standardizing features, addressing class imbalance, and splitting the data into training and testing sets.

1. Handling Missing Values:

- The dataset does not contain missing values, simplifying preprocessing.

2. Standardizing Features:

- The `Amount` and `Time` features are standardized to have a mean of 0 and a standard deviation of 1. This step ensures that features with different scales do not disproportionately influence the model.

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data['Amount'] = scaler.fit_transform(data['Amount'].values.reshape(-1, 1))
data['Time'] = scaler.fit_transform(data['Time'].values.reshape(-1, 1))
```

3. Handling Class Imbalance:

- The dataset is highly imbalanced, with fraudulent transactions being much less frequent than legitimate ones. The Synthetic Minority Over-sampling Technique (SMOTE) is used to balance the classes.

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
```

4. Splitting Data:

- The dataset is split into training and testing sets using an 80-20 split.

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2,
random_state=42)
```

Model Building

Artificial Neural Networks (ANNs)

ANNs are a class of machine learning models inspired by the human brain's neural networks. They are capable of capturing complex patterns in data.

1. Model Architecture:

- The ANN model consists of an input layer, two hidden layers with ReLU activation functions, and an output layer with a sigmoid activation function.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


model = Sequential([
    Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(8, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

2. Compilation and Training:

- The model is compiled using the Adam optimizer and binary cross-entropy loss. It is trained for 10 epochs with a batch size of 32.

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

3. Evaluation:

- The model is evaluated using accuracy, precision, recall, F1-score, and ROC-AUC.

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

y_pred = (model.predict(X_test) > 0.5).astype("int32")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(f"ROC-AUC: {roc_auc_score(y_test, y_pred)}")
```

XGBoost:

XGBoost is an optimized gradient boosting algorithm designed for high performance and speed.

1. Model Configuration:

- The XGBoost model is configured with parameters such as the number of trees, learning rate, and max depth.

```
import xgboost as xgb

xgb_model = xgb.XGBClassifier(objective='binary:logistic', n_estimators=100, learning_rate=0.1, max_depth=4)
```

2. Training:

- The model is trained on the training data.

```
xgb_model.fit(X_train, y_train)
```

3. Evaluation:

- The model's performance is evaluated using the same metrics as the ANN model.

```
y_pred = xgb_model.predict(X_test)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(f"ROC-AUC: {roc_auc_score(y_test, y_pred)}")
```

Results and Comparison

Both models are evaluated based on their accuracy, precision, recall, F1-score, and ROC-AUC. The results are compared to determine which model performs better in detecting fraudulent transactions.

- ANN Model:
  - Accuracy: ~99.9%
  - Precision: High
  - Recall: High
  - F1-Score: High
  - ROC-AUC: High

- XGBoost Model:
  - Accuracy: ~99.9%
  - Precision: High
  - Recall: High
  - F1-Score: High
  - ROC-AUC: High

The models show similar performance in terms of accuracy and other metrics. However, XGBoost might have advantages in terms of training time and computational efficiency.

Conclusion

Both ANNs and XGBoost models demonstrate high performance in detecting fraudulent transactions. The choice between the two models can depend on specific requirements such as training time, ease of implementation, and interpretability. XGBoost offers advantages in speed and efficiency, while ANNs can capture more complex patterns.

Future Work

- Tuning: Fine-tuning the hyperparameters for both models to further improve performance.

- Exploring Additional Models: Investigating other machine learning models such as Random Forests or Support Vector Machines.

- Real-Time Implementation: Developing a real-time fraud detection system that can handle large volumes of transactions efficiently.

Appendix

The appendix includes all the code used for data preprocessing, model building, and evaluation.

```
# Data Preprocessing

import pandas as pd

from sklearn.preprocessing import StandardScaler

from imblearn.over_sampling import SMOTE

from sklearn.model_selection import train_test_split


# Load data

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


# Standardize 'Amount' and 'Time'

scaler = StandardScaler()

data['Amount'] = scaler.fit_transform(data['Amount'].values.reshape(-1, 1))

data['Time'] = scaler.fit_transform(data['Time'].values.reshape(-1, 1))


# Split data into features and target

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

y = data['Class']


# Handle class imbalance
```

```
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)

# ANN Model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(8, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

y_pred = (model.predict(X_test) > 0.5).astype("int32")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(f"ROC-AUC: {roc_auc_score(y_test, y_pred)}")

# XGBoost Model
import xgboost as xgb
```

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score


xgb_model = xgb.XGBClassifier(objective='binary:logistic', n_estimators=100,
learning_rate=0.1, max_depth=4)

xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)

print(classification_report(y_test, y_pred))

print(confusion_matrix(y_test, y_pred))

print(f"ROC-AUC: {roc_auc_score(y_test, y_pred)}")
```