Detecting Credit Card Fraud Using PySpark and ADBench Anomaly Detection Benchmark

Jay Chuwonganant, Enzo Wollmeister, Tyler Smith

Initial Project

- Attempted to use a Databricks Solution Accelerator
  - Credit Card Transactions Analytics
- Became more than our team could handle
  - Strange data with timeseries-based analysis
  - Required connection to AWS Server
    - Including fees beyond free-tier allocation
- Looked for alternative data
  - ADBench Anomaly Detection Benchmark

Introduction

- **Category:** Classification and Clustering
- **Topic:** Analyzing structured data, specifically credit card transactions, to detect anomalies.
- **Tasks:**
  - Identifying fraudulent transactions using anomaly detection techniques.
  - Comparing Deep SVDD and K-Means models for effective fraud detection.
- **Project Goal:**
  - Predict credit card fraud using machine learning models.
  - Identify fraudulent transactions by training and testing the data.
    - Fraudulent transactions seen as anomalies.
- We implemented Deep Support Vector Data Description (DeepSVDD) from the DeepOD library to identify anomalies in credit card fraud data.
- We used the ADBench Credit Card Fraud Dataset.

The Problem and Solution

- **Problem:**
  - Credit card fraud can cost financial systems billions of dollars per year.
  - High dimensional data makes manual fraud detection unrealistic and impractical.
- **Solution:**
  - Explore DeepSVDD for its ability to isolate anomalies and detect fraudulent transactions effectively.

Model Overview – Deep SVDD

- **A Deep SVDD model:**
  - Machine learning technique used to detect anomalies.
  - Maps normal data into a hypersphere with minimal volume ([2] Ruff et al)
    - Anything that falls outside of the hypersphere (outlier) is considered an anomaly.
- **Strengths of Deep SVDD:**
  - Handles high-dimensional data effectively.
  - Focuses specifically on anomaly detection.
  - Reduces false positives by isolating anomalies outside the hypersphere.
- **Weaknesses of Deep SVDD:**
  - Training may be sensitive to the data.
  - Relatively more intensive than certain models.

Demonstration of Model - Data

- Load in data with npz file
- Assign training & test data
- Split data for training & testing evaluation
  - 80/20 Train-Test Split
    - Standard procedure for model evaluation
  - Wanted to see how the reduced input data affected training process

Load in data

```
from numpy import load
data = load("13_fraud.npz")
#print(data.files)
X_train = data['X']
y_train = data['y']
```

```
from sklearn.model_selection import train_test_split

# Assuming 'X_train' and 'y_train' are your original data and labels
X_train2, X_test, y_train2, y_test = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42
)
```

Demonstration of Model - Training

- Use model.fit procedure after import
  - 10 Minute/Model Run with default epochs
- Takes data and creates hypersphere
- Runs X epochs to fit the hypersphere
- Returns a model to make decisions
  - Uses decision_function from DeepOD

```
Start Training...
ensemble size: 1
MLPnet(
  (network): Sequential(
    (0): LinearBlock(
      (linear): Linear(in_features=29, out_features=100, bias=False)
      (act_layer): ReLU()
    )
    (1): LinearBlock(
      (linear): Linear(in_features=100, out_features=50, bias=False)
      (act_layer): ReLU()
    )
    (2): LinearBlock(
      (linear): Linear(in_features=50, out_features=128, bias=False)
      (act_layer): Identity()
    )
  )
)
epoch  1, training loss: 0.001096, time: 7.4s
epoch 10, training loss: 0.000012, time: 7.4s
epoch 20, training loss: 0.000010, time: 7.7s
epoch 30, training loss: 0.000008, time: 7.7s
epoch 40, training loss: 0.000008, time: 7.4s
epoch 50, training loss: 0.000007, time: 7.5s
epoch 60, training loss: 0.000007, time: 7.9s
epoch 70, training loss: 0.000007, time: 7.9s
epoch 80, training loss: 0.000007, time: 7.9s
epoch 90, training loss: 0.000007, time: 10.4s
epoch100, training loss: 0.000006, time: 7.3s
Start Inference on the training data...
testing: 100%|██████████| 4451/4451 [00:02<00:00, 1573.74it/s]
testing: 100%|██████████| 4451/4451 [00:03<00:00, 1356.84it/s]
```

Methodology

- **Data Preparation:**
  - 80/20 split: 80% training, 20% testing.
    - Normalization applied for consistency.
- **Model Implementation:**
  - Tools Used: Python, DeepOD Library, Google Colab.
  - Settings for Deep SVDD:
    - Hypersphere approach, 1 epoch
    - Considered multiple epochs
  - Approaches:
    - Unsupervised Training throughout the entire dataset.
    - Unsupervised Training through a split dataset.
- Evaluation Metrics:
  - AUC, AUCPR, F1-score, ROC curve.
- Null Hypothesis: Deep SVDD model is not different than Kmeans at anomaly detection.
  - 1 degree of freedom
- 20% Holdout Instead of Cross-Validation
  - Ensures consistent comparison between K-Means and Deep SVDD.
  - CV was less feasible due to constraints and computational resources.

Finetuning - DeepSVDD

- The user can choose how many epochs to run
  - Fewer (1) provided the best results
  - More epochs should've improved the model's ability to learn
    - Overfitting
- The Hypersphere approach is unique
  - Difficult to change much – either in or out of the hypersphere
    - Not very flexible
- DeepOD Contamination Threshold
  - Doesn't apply to DeepSVDD

Results – Deep SVDD

- **Unsupervised Training:**
  - AUC: 0.924
- **Supervised Training:**
  - AUC: 0.941
- The Supervised Training yields a higher rate of true positives and a lower rate of false positives.

Model Overview – K-Means

- **A K-Means model:**
  - Groups data into clusters based upon similarities.
  - Fraud cases often do not align well with normal clusters.
    - Anomalies are the points furthest from the center of the cluster.
- **Strengths of K-Means:**
  - More simplistic and faster for low-dimensional data.
  - Works well when clusters are clearly defined.
- **Weaknesses of K-Means:**
  - Struggles with high-dimensional data typical in fraud cases.

Results – K-Means

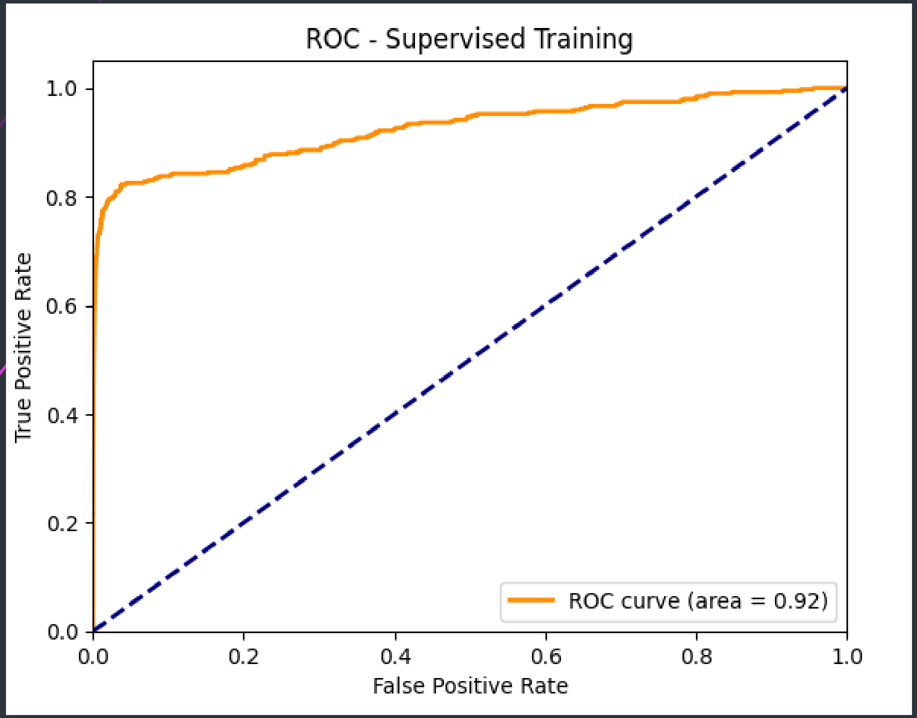
- **K-Means Training Metrics:**
  - AUC: 0.952
  - AUCPR: 0.147
  - F1: 0.226
- **K-Means Testing Metrics:**
  - AUC: 0.963
  - AUCPR: 0.172
  - F1: 0.225

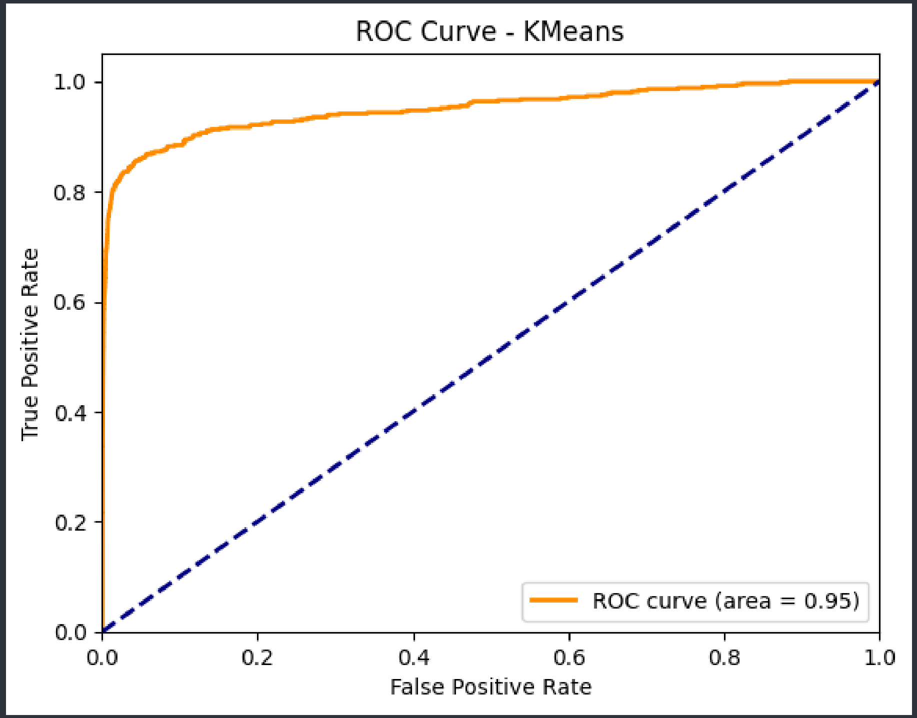
Comparison

Metric	DeepSVDD (Split)	K-Means
Training AUC	0.924	0.952
Testing AUC	0.941	0.963
F1-Score	0.541	0.254

- DeepSVDD performs better on F1-scores.
- K-Means achieves higher AUC.

ROCAUC Plots





Pairwise T-Test

To measure if the difference between Deep SVDD and K-Means is statistically significant.

- **Results:**
  - P-Value: 0.0 (< 0.05)
    - Deep SVDD provides a significantly different targeted fraud detection.

Future Work

- Increased Hyperparameter Tuning
- Sensitivity Analysis into Data Size & Variability
- Generalizability
- Robust Metric Analysis
- Model Evaluation
- Scale the model to handle larger datasets
- Incorporate various other models for comparison
  - Logistic Regression
  - Isolation Forest
  - K-Nearest Neighbors

Conclusion

- The most challenging aspect was learning how to navigate through the different mediums and tools and getting them to function correctly.
- The aspect where we learned the most involved gaining insight into unsupervised learning techniques and their application in anomaly detection.

References

- [1] Hongzuo Xu, Guansong Pang, Yijie Wang and Yongjun Wang, “Deep Isolation Forest for Anomaly Detection,” in IEEE Transactions on Knowledge and Data Engineering, doi: 10.1109/TKDE.2023.3270293.

- [2] Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S.A., Binder, A., Müller, E., Kloft, M.. (2018). Deep One-Class Classification. *Proceedings of the 35th International Conference on Machine Learning, in Proceedings of Machine Learning Research* 80:4393-4402 Available from https://proceedings.mlr.press/v80/ruff18a.html.

- [3] Songqiao Han, Xiyang Hu, Hailiang Huang, Mingqi Jiang, Yue Zhao (2022). ADBench: Anomaly Detection Benchmark. *Neural Information Processing Systems (NeurIPS).* Accessed 2024-11-24.