DeepOD – Deep SVDD Implementation

Introduction

Our team comprised of Jay Chuwonganant, Enzo Wollmeister and Tyler Smith looked for a model to run a credit card dataset predicting fraud for our final project. We were able to find Hongzuo Xu et al's [1] implementation of the DeepOD python library. This library assisted in finding Ruff's [2] Deep Support Vector Data Description (DeepSVDD) model that specializes in anomaly detection from 2018. We will run this model within a Google Colab instance, gathering various metrics for comparative purposes with secondary models of interest with the main task to detect anomalies through some unsupervised or supervised processes.

Background and Related Work

There are many instances of anomaly detection models available. Our initial project idea included using similar training datasets regarding banking data through a Databricks instance, but this unfortunately required additional resources not provided by Databricks community edition. Our team worked through Databricks & Amazon Web Services (AWS) and were able to successfully run pieces of this model, but unfortunately between the AWS resources and model output, we determined it was in our best interest to find an alternative model to focus on while still looking into credit card fraud data.

This search led us to the DeepOD library and the various models within. Code was provided in the DeepOD Github repository that could run the standardized datasets provided by Songqiao et al [3] in the ADBench repository. This data includes the credit card

fraud dataset that aligns with our project scope. The ADBench benchmark data is widely considered the standard checkpoint to test anomaly detection models. The ADBench datasets do not outright split data into train or test, but our team decided to move forward with an 80/20 train-test split through SKLearn to see how that impacted the model training results.

Our team searched through the various models DeepOD has access to. Scale Learning-based deep Anomaly Detection (SLAD) was one such model that the team considered, but ultimately the team decided that the DeepSVDD model was the most intriguing due to the better understanding we were able to grasp regarding the model after reading through some documentation.

The DeepSVDD model has the main objective of accurately describing normal ([2] Ruff et al). This type of model is especially important since other simpler implementations of anomaly detection have issues parsing through high-dimensional data with poor computational scalability as well. DeepSVDD's major claim is that it is trained by optimizing an anomaly detection-based objective function and other heuristics ([2] Ruff et al). The team was unable initially to identify the hyperparameters given the limited availability of information outside of the paper, but eventually found two candidates: contamination and epochs.

Methodology

DeepSVDD is shown as the example model within the DeepOD documentation initially making it a strong candidate for our team to work with. After the initial delays, the

model can be easily run in an unsupervised manner for a zero-shot implementation. The code to run this model is rather simple. First, the user will call a new model, fit the model with the training data, and then use the decision function based on the fitted model's results. The model requires a NumPy vector as its inputs which are stored in .npz files from the ADBench repository. Each training instance takes around 10 minutes to complete using the default number of epochs.

Moving into how DeepSVDD returns predictions, the documentation states that the model finds common factors of variation through a neural network, eventually resulting in a hypershere rather than error reconstruction ([2] Ruff et al). Ruff et al continues about how the compression of autoencoders becomes a hyperparameter that needs tuning, and the hypershere approach provides the minimized volume to help the model focus on the anomaly detection task. The model is optimized through stochastic gradient descent, converging to a local minimum.

Utilizing DeepODs tabular_metrics function, we can produce the AUC score, accuracy and F1 metrics. Through a separate method, we can fit other metrics including the ROC plots, and eventually run the pairwise t-test through the scipy library. This will allow us to compare to a baseline result provided by a simple k-means algorithm. After an 80/20 train-test split, both the training and testing data had 0.1% rate of fraudulent claims, matching the overall dataset rate among 285,000 overall rows. There are 590 total fraudulent claims to be identified. This split was done at first in an exploratory nature, but the results ended up performing better with the split, so the team stuck with this method.

Experiment Design and Results

To test the effectiveness of these models, we successfully implemented and evaluated the DeepSVDD anomaly detection model using the credit card fraud dataset. Its performance was compared to a k-means clustering model. Our team's primary objective was to see if this complex modeling technique would significantly outperform a simple algorithm like k-means.

The first training instance was an unsupervised approach using the full available data and the default number of epochs. Due to the nature of training the hypershere with DeepSVDD, the amount of training data fed into it does significantly change the results, and it's recommended in the documentation to put as much data as possible to allow for proper training. This run returned an AUC score of 0.83 which was lower than expected. The corresponding ROC plot is shown in exhibit A, and displays a trend seen in many of the DeepSVDD models with true positive rate flatlining around 0.7. This behavior indicates good performance predicting true positives without many false positives up until a point. This is common with data containing few false positives like this credit card fraud dataset and is something to consider as our team compares between models.

After splitting the data with an 80/20 split, the AUC score improved on both training and testing datasets, with similar results of 0.90 for both. Each instance was trained in an unsupervised manner with the default epochs, and the results are rather odd to see a lower AUC score when training with the full data, but it remains a possibility given the nature of the DeepSVDD training and having very few actual fraudulent claims in the

dataset. On the test data as part of the 80/20 split, the model achieved an f1 score of 0.418 which does outperform the full model run that returned a 0.254 f1 score.

Following work preparing this final project and presentation, the code with DeepOD presented the option to limit the epochs which sparked additional curiosity within our team. After playing around with the epochs ran, we determined the best results actually came with a single epoch. This follows the initial intuition with the hyperspheric approach in DeepSVDD as it needs to understand the structure of the data to construct the hypersphere and then it is ready to make predictions. Running the single epoch resulted in our final model displayed in Exhibit C, presenting the highest AUC scores of 0.924 for training and 0.941 for testing. The f1 scores similar displayed marked improvement with scores of 0.459 for training and 0.541 for testing.

Exhibit A: Full Training AUC – Unsupervised DeepSVDD

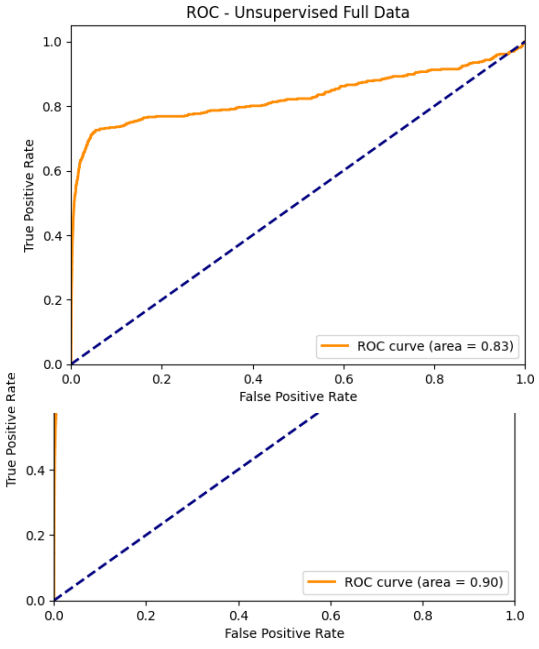


Exhibit B: 80/20 Split, Unsupervised DeepSVDD
Exhibit B-1: Training Data
Exhibit B-2: Test Data

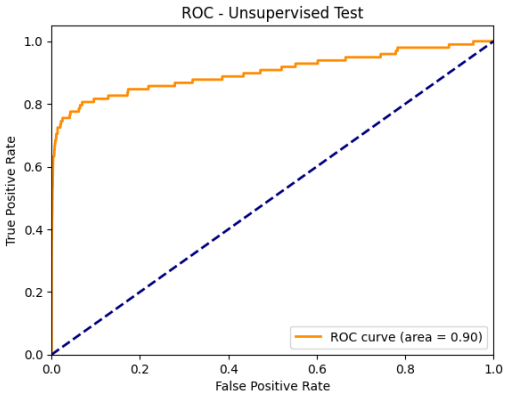
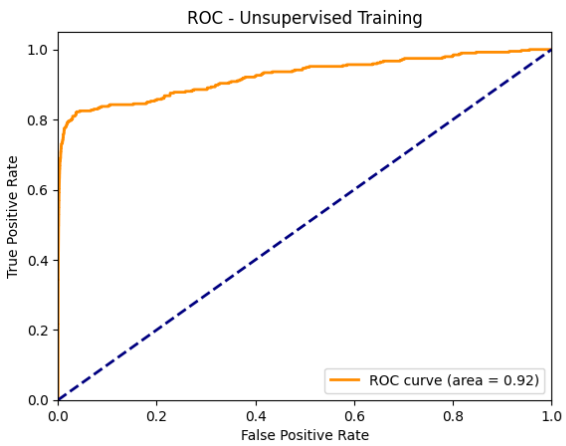
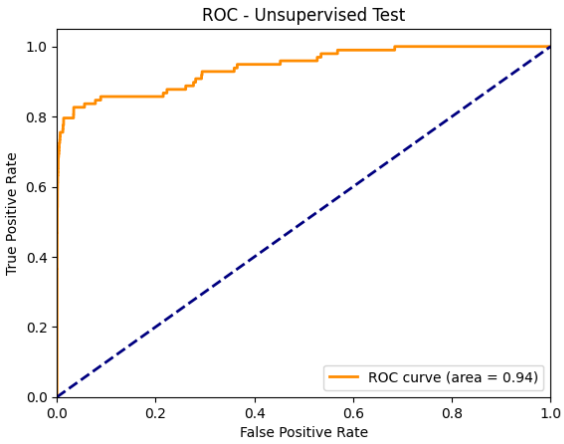


Exhibit C: 80/20 Split, Unsupervised DeepSVDD, 1 Epoch
Exhibit C-1: Training Data

Exhibit C-2: Test Data

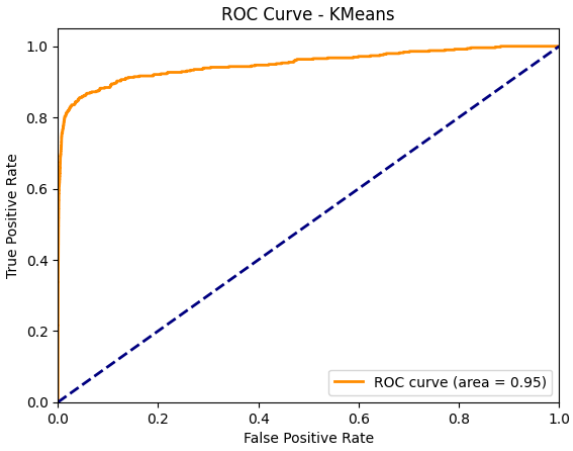




The code provided in Appendix A displays the results taken for a K-Means model simulation. The results show a high AUC for both training (0.952) and testing (0.963) which indicates that the model is quite effective in distinguishing between normal and fraudulent transactions. This was counter-intuitive to our initial expectations since we would hope to see an improved model outcome from a more advanced model like DeepSVDD.

Exhibit D: 80/20 Split, K-Means

Exhibit D-1: Training Data



For the K-Means model, the area under precision-recall curve (AUCPR) score is relatively low. The AUCPR for training is 0.1466 and for testing is 0.1716. This indicates that the K-Means model may be good at ranking positive examples, but it still struggles to capture the anomalies properly in regards to our dataset. These scores in particular are lower than the DeepSVDD model from Exhibit C which returned a training score of 0.373 and test score of 0.504. The f1 score is also relatively low in comparison with the K-Means f1 score for training at 0.226 compared to a 0.459, and for testing is 0.255 from K-Means compared to 0.541 from DeepSVDD. K-Means struggles at identifying fraudulent transactions in an overall basis, and we see marked improvement with the more advanced DeepSVDD model, which makes sense given the low number of actual fraudulent transactions present in our dataset.

Comparing DeepSVDD and K-Means, both models demonstrate strong performance with high AUC scores. This indicates that both models should be capable of distinguishing fraudulent transactions from normal transactions effectively, but each still struggles with overall accuracy due to false positive rates shown in the AUC plots.

The final test was to run a pairwise t-test on the resulting data to see if DeepSVDD provided significantly different predictions than a K-Means model. The result of this test can be found in Appendix B, and shows that the two sets of predictions are significantly different with a very small p-value (< 0.05). Based on these results, we see unique predictions from DeepSVDD, as well as marked improved in the f1 score, leading us to conclude that DeepSVDD is a better option for anomaly detection than the baseline K-Means model.

Summary and Future Work

This project successfully implemented and evaluated the DeepSVDD anomaly detection model using a credit card fraud dataset. Our primary objective was to assess the effectiveness of DeepSVDD compared to a simpler algorithm, K-Means, for anomaly detection. The DeepSVDD model showed notable performance advantages in several areas, particularly in F1 scores and AUCPR metrics, despite exhibiting unexpected AUC results when trained on the full dataset. The comparison revealed that while K-Means provides strong AUC scores, it struggles with identifying fraudulent transactions accurately, as indicated by its relatively lower F1 and AUCPR scores.

DeepSVDD demonstrated its capability to process high-dimensional data and detect anomalies effectively, leveraging a hypersphere approach to minimize volume and optimize for anomalies. The pairwise t-test further confirmed that DeepSVDD produced significantly different predictions compared to K-Means, reinforcing its value as an advanced model for anomaly detection. However, results also highlighted potential issues with tuning the model's hyperparameters and the challenges of working with smaller datasets or unsupervised training settings.

Future work on the DeepSVDD model could yield improved results both on the dataset our team worked with, and in other applications of the model. The performance of DeepSVDD, particularly its AUC scores, suggests that more optimization of the hypersphere parameters could improve its anomaly detection capabilities. For the scope of this project, our team felt good enough about the results to draw conclusions, but

conducting more thorough parameter tuning could yield better results. The results we looked at also imply that further investigation of the model's sensitivity to data size and variability is necessary. The different full-data run compared to the train-test split run shows an impact in the data training. The model isn’t necessarily trained, but does need to understand the scope of the data to tune the hypersphere. Conducting experiments with varying dataset splits and different sizes could provide a clearer understanding of its behavior.

An increased scope of the DeepSVDD model would also be an interesting topic to explore. While outside the scope of this project, applying DeepSVDD to other real-world datasets would validate its generalizability and practical utility across multiple. That work could emphasize a more robust metric analysis, including exploring the relationship between AUC, AUCPR, and F1 scores across training scenarios. This additional analysis would provide deeper insights into overall performance.

Appendix A

```python
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from deepod.metrics import tabular_metrics

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train2)
X_test_scaled = scaler.transform(X_test)

# Fit KMeans model
kmeans = KMeans(n_clusters=2, random_state=42)  # Assuming 2 clusters for anomaly detection
kmeans.fit(X_train_scaled)

# Calculate anomaly scores (distance to the nearest cluster center)
kmeans_distances_train = kmeans.transform(X_train_scaled)  # Distances to cluster centers for training data
kmeans_distances_test = kmeans.transform(X_test_scaled)  # Distances to cluster centers for test data

# Anomaly scores: minimum distance to the nearest cluster center
scores_train_kmeans = kmeans_distances_train.min(axis=1)
scores_test_kmeans = kmeans_distances_test.min(axis=1)

# Evaluate using AUC, AP, and F1 scores
auc_train_kmeans, ap_train_kmeans, f1_train_kmeans = tabular_metrics(y_train2, scores_train_kmeans)
auc_test_kmeans, ap_test_kmeans, f1_test_kmeans = tabular_metrics(y_test, scores_test_kmeans)

# Print results
print(f"KMeans Training Metrics - AUC: {auc_train_kmeans}, AP: {ap_train_kmeans}, F1: {f1_train_kmeans}")
print(f"KMeans Test Metrics - AUC: {auc_test_kmeans}, AP: {ap_test_kmeans}, F1: {f1_test_kmeans}")
```

```
KMeans Training Metrics - AUC: 0.9521533694756636, AP: 0.1466135615782124, F1: 0.22588832487309646
KMeans Test Metrics - AUC: 0.9631415593812088, AP: 0.17158117139102222, F1: 0.25510204081632654
```

Appendix B

```python
# DeepSVDD scores
scores_deepsvdd = clf2.decision_function(X_test)

# KMeans scores
kmeans_distances = kmeans.transform(X_test)
scores_kmeans = kmeans_distances.min(axis=1)

### Pair-wise T-Test
from scipy import stats

# Assuming 'scores_deepsvdd' and 'scores_other_model' are anomaly scores from both models
t_statistic, p_value = stats.ttest_rel(scores_deepsvdd, scores_kmeans)

print("T-statistic:", t_statistic)
print("P-value:", p_value)
```

```
testing: 100%|██████████| 891/891 [00:02<00:00, 389.36it/s]
T-statistic: -20408.778530205596
P-value: 0.0
```

Bibliography

[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.