Support Vector Machine (SVM) for Anomaly Detection

Support Vector Machines (SVMs) are powerful supervised learning models that can also be used for anomaly detection. They can be effective for anomaly detection because they find the hyperplane that best separates the normal data points from the anomalies.

Mainly, the one-class support vector machine is an unsupervised model for anomaly or outlier detection. In this article, we will discuss how we can use support vector machines for anomaly detection.

What is an anomaly?

- An anomaly is something that differs from what is typical, normal, or expected. It can be an irregularity or an outlier that stands out from the usual pattern. Anomalies are important because they often signal unusual or unexpected events, such as errors, fraud, or rare incidents.
- Anomaly detection involves identifying these uncommon patterns or outliers within a dataset. This process finds applications in various fields, including fraud detection, network security, healthcare, manufacturing, and more.

Why do we use Support Vector Machines for Anomaly Detection?

We use Support Vector Machine for anomaly detection because of the following reasons:

1. **Effective for High-Dimensional Data**: SVMs perform well in high-dimensional spaces, making them suitable for datasets with many features, such as those commonly encountered in anomaly detection tasks.
2. **Robust to Overfitting**: SVMs are less prone to overfitting, which is crucial in anomaly detection where the goal is to generalize well to unseen anomalies.
3. **Optimal Separation**: SVMs aim to find the hyperplane that maximally separates the normal data points from the anomalies, making them effective in identifying outliers.
4. **One-Class SVM**: The One-Class SVM variant is specifically designed for anomaly detection, learning to distinguish normal data points from outliers without the need for labeled anomalies.
5. **Kernel Trick**: SVMs can use kernel functions to map data into a higher-dimensional space, allowing for non-linear separation of anomalies from normal data.
6. **Handling Imbalanced Data**: Anomaly detection datasets are often highly imbalanced, with normal data points outnumbering anomalies. SVMs can handle this imbalance well.

7. **Interpretability**: SVMs provide clear decision boundaries, which can help in interpreting why a particular data point is classified as an anomaly

In this article we will seeks answers to the questions:

- How to train a one-class support vector machine (SVM) model.
- How to predict anomalies from a one-class SVM model.
- How to change the default threshold for anomaly prediction.
- How to visualize the prediction results.

Implementation of using Support Vector Machines for anomaly detection

To demonstrate how to use Support Vector Machines for anomaly detection we will use a sample dataset.

Step 1: Import Libraries

The code imports necessary libraries for data processing, visualization, model training, and evaluation.

Python` ```

Synthetic dataset

from sklearn.datasets import make_classification

#Data processing import pandas as pd import numpy as np from collections import Counter

#Visualization import matplotlib.pyplot as plt

#Model and performance from sklearn.model_selection import train_test_split from sklearn.svm import OneClassSVM from sklearn.metrics import classification_report

```
    `

### Step 2: Create Imbalanced Dataset

The code generates an imbalanced synthetic dataset with two features (`feature1` and `feature2

Python`   ```
# Create an imbalanced dataset
X, y = make_classification(n_samples=100000, n_features=2, n_informative=2,
```

```
                                    n_redundant=0, n_repeated=0, n_classes=2,
                                    n_clusters_per_class=1,
                                    weights=[0.995,0.005],
                                    class_sep=0.5, random_state=0)

  #Convert the data from numpy array to a pandas dataframe
  df = pd.DataFrame({'feature1': X[:, 0], 'feature2': X[:, 1], 'target': y})

  #Check the target distribution
  df['target'].value_counts(normalize = True)
```

`

Output:

```
target
0    0.9897
1    0.0103
Name: proportion, dtype: float64
```

The output shows that we have about 1% of the data in the minority class and 99% in the majority class.

Step 3: Train Test Split

In this step, we split the dataset into 80% training data and 20% validation data. random_state ensures that we have the same train test split every time. The seed number for random_state does not have to be 42, and it can be any number.

Python` ```

Train test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Check the number of records print('The number of records in the training dataset is', X_train.shape[0]) print('The number of records in the test dataset is', X_test.shape[0])

Analyze class distribution in the training set

class_counts = Counter(y_train) majority_class, majority_count = sorted(class_counts.items())[0] minority_class, minority_count = sorted(class_counts.items())[-1]

print(f"The training dataset has records for the majority class ()") print(f"and records for the minority class ()")

`

****Output:****

The number of records in the training dataset is 80000 The number of records in the test dataset is 20000 The training dataset has 79183 records for the majority class (0) and 817 records for the minority class (1)

```
The train test split gives us 80,000 records for the training dataset and 20,000 for the valid

### Step 4: Train One Class Support Vector Machine (SVM) Model

When training the one-class SVM, there are a few critical hyperparameters:

*   ****nu**** is to specify the percentage of anomalies. nu=0.01 means that we have found 1%
*   ****Kernel**** specifies the kernel type. The radial basis function (rbf) kernel is a comm
*   ****gamma**** is a kernel coefficient, and it is for ‘rbf’ , ‘poly’ , and ‘sigmoid’ kernel

Python`   ```
# Train the one support vector machine (SVM) model
one_class_svm = OneClassSVM(nu = 0.01, kernel = 'rbf', gamma = 'auto').fit(X_train)
```

`

Step 5: Predict Anomalies

- After training the one-class SVM model on the training dataset, we make predictions on the testing dataset. By default, one-class SVM labels the normal data points as 1s and anomalies as -1s.
- To compare the labels with the ground truth in the testing dataset, we changed the anomalies’ label from -1 to 1, and the normal labels from 1 to 0.

Python` ```

Predict the anomalies

prediction = one_class_svm.predict(X_test)

#Change the anomalies' values and to make it consistent with the true values prediction = [1 if i==-1 else 0 for i in prediction]

#Check the model performance print(classification_report(y_test, prediction))

`

****Output:****

	precision	recall	f1-score	support
0	0.99	0.99	0.99	19787
1	0.06	0.06	0.06	213

The model has a [recall](https://www.geeksforgeeks.org/precision-and-recall-in-information-ret

Step 6: Customize Predictions Using Scores

Instead of using the default threshold to identify outliers, ****we can customize the threshol

Python` ```

```
# Get the scores for the testing dataset
score = one_class_svm.score_samples(X_test)

#Check the score for 2% of outliers
score_threshold = np.percentile(score, 2)
print(f'The customized score threshold for 2% of outliers is {score_threshold: .2f}')
# Check the model performance at 2% threshold
customized_prediction = [1 if i < score_threshold else 0 for i in score]
#Check the prediction performance
print(classification_report(y_test, customized_prediction))
```

```
```

Output:

```
The customized score threshold for 2% of outliers is  182.62
              precision    recall  f1-score   support
           0       0.99      0.98      0.99     19787
           1       0.06      0.10      0.07       213
    accuracy                           0.97     20000
   macro avg       0.52      0.54      0.53     20000
weighted avg       0.98      0.97      0.98     20000
```

The recall value increased from 6% to 10% because we increased the threshold for anomalies.

Step 7: Putting test and predictions in the same data frame

Python` ```

Put the testing dataset and predictions in the same dataframe

df_test = pd.DataFrame(X_test, columns=['feature1', 'feature2']) df_test['y_test'] = y_test df_test['one_class_svm_prediction'] = prediction df_test['one_class_svm_prediction_customized'] = customized_prediction

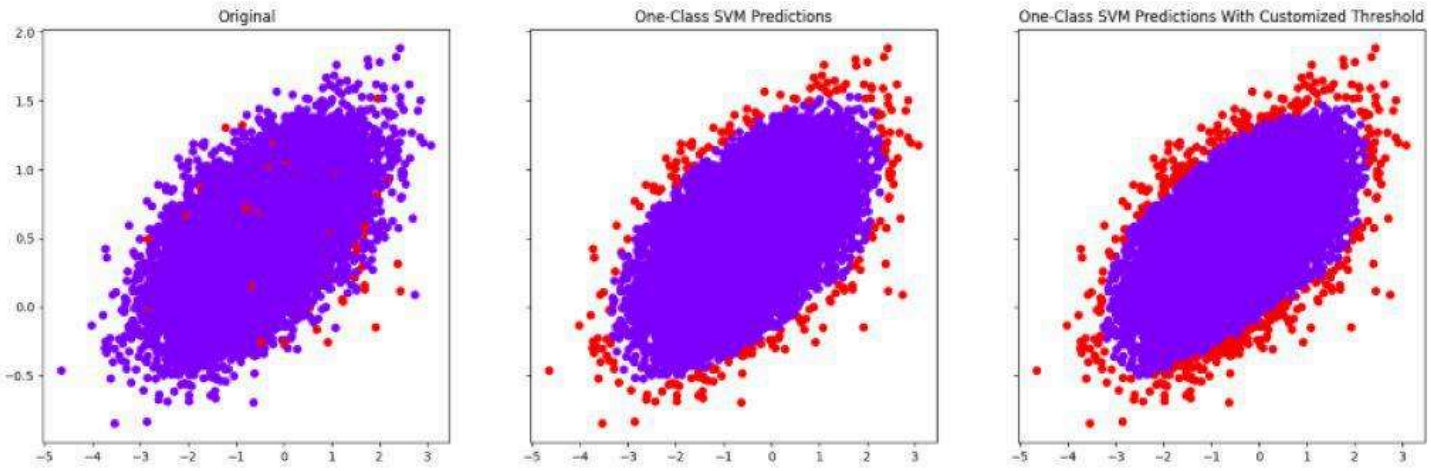
```
```

```
### Step 8: Visualization

This step will plot the data points and check the differences between actual, one-class SVM pr

Python`   ```
# Visualize the actual and predicted anomalies
fig, (ax0, ax1, ax2)=plt.subplots(1,3, sharey=True, figsize=(20,6))

#Ground truth
ax0.set_title('Original')
ax0.scatter(df_test['feature1'], df_test['feature2'], c=df_test['y_test'], cmap='rainbow')

#One-Class SVM Predictions
ax1.set_title('One-Class SVM Predictions')
ax1.scatter(df_test['feature1'], df_test['feature2'], c=df_test['one_class_svm_prediction'], c
```

```
#One-Class SVM Predictions With Customized Threshold
ax2.set_title('One-Class SVM Predictions With Customized Threshold')
ax2.scatter(df_test['feature1'], df_test['feature2'], c=df_test['one_class_svm_prediction_cust
```

`

Output:



We can see that one-class SVM has a clear boundary and labeled the data points out of the boundary to be anomalies. When we increase the threshold for the score, more data points are labeled as anomalies.

Conclusion

In conclusion, One-Class SVMs are highly effective for anomaly detection due to their ability to handle high-dimensional and imbalanced data while providing robust decision boundaries. By tuning hyperparameters and decision thresholds, these models can accurately identify and separate anomalies from normal data points, making them invaluable in fields such as fraud detection, network security, and healthcare. This approach demonstrates how SVMs can be leveraged to detect rare and unusual events within a dataset.