Project Title: OUTLIER DETECTION APPROACHES ON STREAMING DATASETS

**PROPOSED WORK:** To take KDDCup'99 Network Intrusion Detection dataset, with its rich set of network traffic records, serves as an excellent benchmark for evaluating various outlier detection techniques. We explore and evaluate multiple approaches to identify the most effective method for detecting anomalies.

The approaches include Statistical method, proximity-based method, clustering based method,

classification-based method. Each technique has its own strengths and limitations. We aim to

determine the best approach based on accuracy and computational efficiency.We would

implement and evaluate the following outlier detection techniques on a subset of the KDDCup'99

dataset using Python:

Local Outlier Factor (LOF)

K-Nearest Neighbors (KNN)

One-Class SVM (OCSVM)

Isolation Forest (IForest)

K-Means Clustering

DBSCAN

The accuracy of each method will be measured by comparing its precision, recall, its F1 score and be compared its detected outliers against the known labels in the dataset.

**How to execute:**

Step 1: Unzip the attached zip file.

Step 2: Place the dataset file it in your local location.

Step 3: Open the python source code in a code editor and modify the line #25 to add the correct path of the dataset from the local location.

Step 4: Execute the python code (sourcecode.py)

Step 5: Close any intermediate results that pop up as images to move to the next section of the code.

Also, find the link to the google colab:

<https://colab.research.google.com/drive/1djqyoPQ_u67nKLoVDiltIjhpbFh7adsi?usp=sharing>

you can see the how the code is run in the google colab link above.

Code details below with results:

**Let us Start:** First Install scikit-learn and install pandas numpy matplotlib seaborn

We are using python code

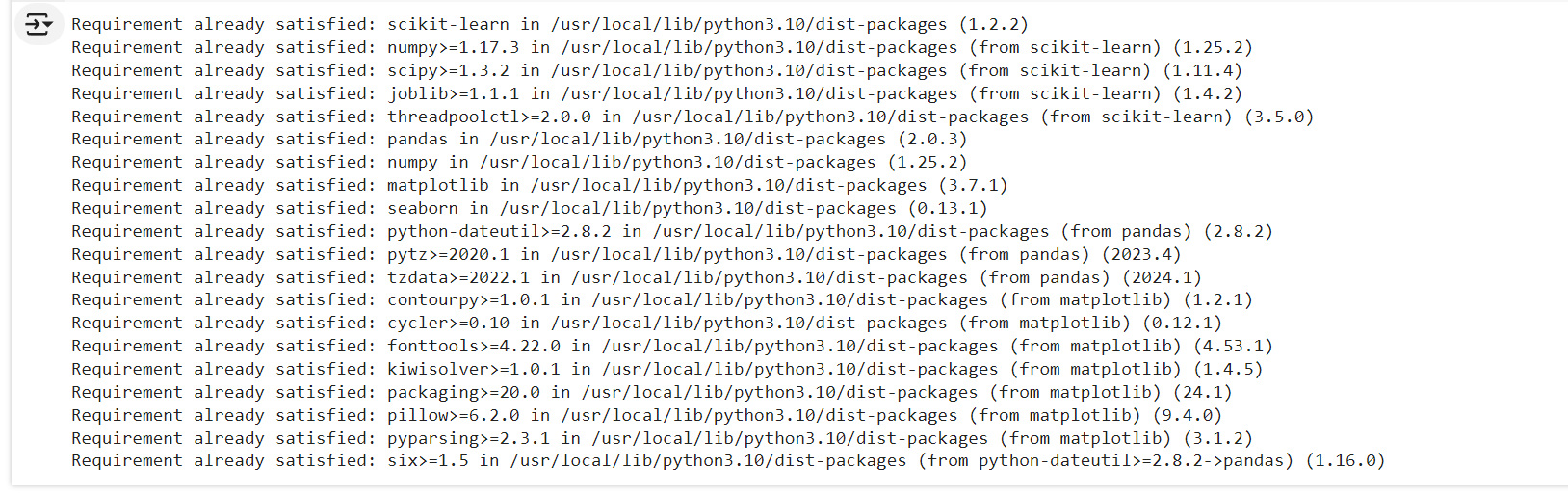
Step- 1:

# Install necessary libraries

!pip install scikit-learn

!pip install pandas numpy matplotlib seaborn

When we run it we will get:



Step-2:Importing the functions and cpmmands

import pandas as pd

import numpy as np

from sklearn.neighbors import LocalOutlierFactor, KNeighborsClassifier

from sklearn.svm import OneClassSVM

from sklearn.ensemble import IsolationForest

from sklearn.cluster import KMeans, DBSCAN

from sklearn.metrics import accuracy\_score

from sklearn.impute import SimpleImputer

import matplotlib.pyplot as plt

import seaborn as sns

Step-3: defining the schemea for the data set

column\_names = ['duration', 'protocol\_type', 'service', 'flag', 'src\_bytes', 'dst\_bytes',

'land', 'wrong\_fragment', 'urgent', 'hot', 'num\_failed\_logins', 'logged\_in',

'num\_compromised', 'root\_shell', 'su\_attempted', 'num\_root', 'num\_file\_creations',

'num\_shells', 'num\_access\_files', 'num\_outbound\_cmds', 'is\_host\_login', 'is\_guest\_login',

'count', 'srv\_count', 'serror\_rate', 'srv\_serror\_rate', 'rerror\_rate', 'srv\_rerror\_rate',

'same\_srv\_rate', 'diff\_srv\_rate', 'srv\_diff\_host\_rate', 'dst\_host\_count', 'dst\_host\_srv\_count',

'dst\_host\_same\_srv\_rate', 'dst\_host\_diff\_srv\_rate', 'dst\_host\_same\_src\_port\_rate',

'dst\_host\_srv\_diff\_host\_rate', 'dst\_host\_serror\_rate', 'dst\_host\_srv\_serror\_rate',

'dst\_host\_rerror\_rate', 'dst\_host\_srv\_rerror\_rate', 'label']

Step-4:

We need to download the zipped file that has uploaded in the document and unzip the dataset file and place it in your local location. And

Here we are going to copy paste your pathway of the zip file that has been downloaded in your Pc.

data = pd.read\_csv('/content/kddcup.data\_10\_percent/kddcup.data\_10\_percent', header=None, names=column\_names)

# Now Data Cleansing and Preparation:

# Now data will be preprocessed here by removing the duplicates , handling the missing values and normalizing the data.

Step –1:

# Remove duplicates

data.drop\_duplicates(inplace=True)

Step-2:

# Encode categorical features

categorical\_features = ['protocol\_type', 'service', 'flag']

data = pd.get\_dummies(data, columns=categorical\_features)

Step-3:

# Convert labels to binary (0 for normal, 1 for anomaly)

data['label'] = data['label'].apply(lambda x: 0 if x == 'normal.' else 1)

Step-4:

# For simplicity, we will use only numerical features

data\_features = data.drop(columns=['label'])

labels = data['label']

Step-5:

# Handle missing values using SimpleImputer

imputer = SimpleImputer(strategy='mean')

data\_features\_imputed = imputer.fit\_transform(data\_features)

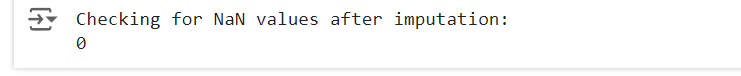
data\_features\_imputed = pd.DataFrame(data\_features\_imputed, columns=data\_features.columns)

Step-6:

print("Checking for NaN values after imputation:")

print(data\_features\_imputed.isna().sum().sum())

When we run it we get the output as :



Step-7:

# Normalize data

data\_features\_imputed = (data\_features\_imputed - data\_features\_imputed.mean()) / data\_features\_imputed.std()

Step-8:

# Ensure no NaN values after normalization

data\_features\_imputed.fillna(0, inplace=True)

Step-9:

# Reset index to ensure alignment

data\_features\_imputed.reset\_index(drop=True, inplace=True)

labels.reset\_index(drop=True, inplace=True)

Step-10:

# Let's randomly sample a subset for visualization purposes

sample\_indices = np.random.choice(data\_features\_imputed.index, size=1000, replace=False)

data\_sampled = data\_features\_imputed.loc[sample\_indices]

labels\_sampled = labels.loc[sample\_indices]

# Now Plotting the outliers:

# Step-1:

# # Function to plot data with outliers

def plot\_outliers(data, outliers, title, x\_label='Feature 1', y\_label='Feature 2'):

plt.figure(figsize=(10, 6))

plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c='blue', label='Normal')

plt.scatter(data.iloc[outliers, 0], data.iloc[outliers, 1], c='red', label='Outliers')

plt.title(title)

plt.xlabel(x\_label)

plt.ylabel(y\_label)

plt.legend()

plt.show()

Now we can perform the operations : Local Outlier Factor (LOF)

K-Nearest Neighbors (KNN)

One-Class SVM (OCSVM)

Isolation Forest (IForest)

K-Means Clustering

DBSCAN

# 1. Local Outlier Factor

Step-1:

lof = LocalOutlierFactor(n\_neighbors=20)

outliers\_lof = lof.fit\_predict(data\_sampled)

outliers\_lof = np.where(outliers\_lof == -1, 1, 0)

accuracy\_lof = accuracy\_score(labels\_sampled, outliers\_lof)

precision\_lof = precision\_score(labels\_sampled, outliers\_lof)

recall\_lof = recall\_score(labels\_sampled, outliers\_lof)

f1\_lof = f1\_score(labels\_sampled, outliers\_lof)

plot\_outliers(data\_sampled, np.where(outliers\_lof == 1)[0], f'Local Outlier Factor (LOF), Accuracy: {accuracy\_lof:.2f}')

When we run it we get output as:

A graph with red and blue dots

Description automatically generated

# 2. K-Nearest Neighbors (KNN)

Step-1:

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(data\_sampled, labels\_sampled)

distances, \_ = knn.kneighbors(data\_sampled)

outliers\_knn = np.where(distances.mean(axis=1) > np.percentile(distances.mean(axis=1), 95), 1, 0)

accuracy\_knn = accuracy\_score(labels\_sampled, outliers\_knn)

precision\_knn = precision\_score(labels\_sampled, outliers\_knn)

recall\_knn = recall\_score(labels\_sampled, outliers\_knn)

f1\_knn = f1\_score(labels\_sampled, outliers\_knn)

plot\_outliers(data\_sampled, np.where(outliers\_knn == 1)[0], f'K-Nearest Neighbors (KNN), Accuracy: {accuracy\_knn:.2f}')

When we run it, we get output as:

A graph with numbers and a line of numbers

Description automatically generated with medium confidence

# 3. One-Class SVM

Step-1:

ocsvm = OneClassSVM(gamma='auto')

outliers\_ocsvm = ocsvm.fit\_predict(data\_sampled)

outliers\_ocsvm = np.where(outliers\_ocsvm == -1, 1, 0)

accuracy\_ocsvm = accuracy\_score(labels\_sampled, outliers\_ocsvm)

precision\_ocsvm = precision\_score(labels\_sampled, outliers\_ocsvm)

recall\_ocsvm = recall\_score(labels\_sampled, outliers\_ocsvm)

f1\_ocsvm = f1\_score(labels\_sampled, outliers\_ocsvm)

plot\_outliers(data\_sampled, np.where(outliers\_ocsvm == 1)[0], f'One-Class SVM, Accuracy: {accuracy\_ocsvm:.2f}')

When we run it we get the output as:

A graph with red dots

Description automatically generated

# 4. Isolation Forest:

iforest = IsolationForest(contamination=0.1)

outliers\_iforest = iforest.fit\_predict(data\_sampled)

outliers\_iforest = np.where(outliers\_iforest == -1, 1, 0)

accuracy\_iforest = accuracy\_score(labels\_sampled, outliers\_iforest)

precision\_iforest = precision\_score(labels\_sampled, outliers\_iforest)

recall\_iforest = recall\_score(labels\_sampled, outliers\_iforest)

f1\_iforest = f1\_score(labels\_sampled, outliers\_iforest)

plot\_outliers(data\_sampled, np.where(outliers\_iforest == 1)[0], f'Isolation Forest, Accuracy: {accuracy\_iforest:.2f}')

When we run it, we get output as:

A graph with red and blue dots

Description automatically generated

# 5. K-Means Clustering:

kmeans = KMeans(n\_clusters=10)

clusters = kmeans.fit\_predict(data\_sampled)

distances = kmeans.transform(data\_sampled).min(axis=1)

outliers\_kmeans = np.where(distances > np.percentile(distances, 95), 1, 0)

accuracy\_kmeans = accuracy\_score(labels\_sampled, outliers\_kmeans)

precision\_kmeans = precision\_score(labels\_sampled, outliers\_kmeans)

recall\_kmeans = recall\_score(labels\_sampled, outliers\_kmeans)

f1\_kmeans = f1\_score(labels\_sampled, outliers\_kmeans)

plot\_outliers(data\_sampled, np.where(outliers\_kmeans == 1)[0], f'K-Means Clustering, Accuracy: {accuracy\_kmeans:.2f}')

When we run it we get an output as:

A screenshot of a computer

Description automatically generated

# 6. DBSCAN:

dbscan = DBSCAN(eps=0.5, min\_samples=5)

clusters = dbscan.fit\_predict(data\_sampled)

outliers\_dbscan = np.where(clusters == -1, 1, 0)

accuracy\_dbscan = accuracy\_score(labels\_sampled, outliers\_dbscan)

precision\_dbscan = precision\_score(labels\_sampled, outliers\_dbscan)

recall\_dbscan = recall\_score(labels\_sampled, outliers\_dbscan)

f1\_dbscan = f1\_score(labels\_sampled, outliers\_dbscan)

plot\_outliers(data\_sampled, np.where(outliers\_dbscan == 1)[0], f'DBSCAN, Accuracy: {accuracy\_dbscan:.2f}')

When we run it we get output as:

A graph with red dots

Description automatically generated

# Print accuracy scores:

print(f'Accuracy Scores:')

print(f'Local Outlier Factor (LOF): {accuracy\_lof:.2f}')

print(f'K-Nearest Neighbors (KNN): {accuracy\_knn:.2f}')

print(f'One-Class SVM: {accuracy\_ocsvm:.2f}')

print(f'Isolation Forest: {accuracy\_iforest:.2f}')

print(f'K-Means Clustering: {accuracy\_kmeans:.2f}')

print(f'DBSCAN: {accuracy\_dbscan:.2f}')

When we run it we will get our final output as:

A screenshot of a computer

Description automatically generated

# Print precision, recall, F1 scores

print(f'Precision Scores:')

print(f'Local Outlier Factor (LOF): {precision\_lof:.2f}')

print(f'K-Nearest Neighbors (KNN): {precision\_knn:.2f}')

print(f'One-Class SVM: {precision\_ocsvm:.2f}')

print(f'Isolation Forest: {precision\_iforest:.2f}')

print(f'K-Means Clustering: {precision\_kmeans:.2f}')

print(f'DBSCAN: {precision\_dbscan:.2f}')

we get out put as :

A screenshot of a computer

Description automatically generated

print(f'Recall Scores:')

print(f'Local Outlier Factor (LOF): {recall\_lof:.2f}')

print(f'K-Nearest Neighbors (KNN): {recall\_knn:.2f}')

print(f'One-Class SVM: {recall\_ocsvm:.2f}')

print(f'Isolation Forest: {recall\_iforest:.2f}')

print(f'K-Means Clustering: {recall\_kmeans:.2f}')

print(f'DBSCAN: {recall\_dbscan:.2f}')

we get output as :

A screenshot of a computer

Description automatically generated

print(f'F1 Scores:')

print(f'Local Outlier Factor (LOF): {f1\_lof:.2f}')

print(f'K-Nearest Neighbors (KNN): {f1\_knn:.2f}')

print(f'One-Class SVM: {f1\_ocsvm:.2f}')

print(f'Isolation Forest: {f1\_iforest:.2f}')

print(f'K-Means Clustering: {f1\_kmeans:.2f}')

print(f'DBSCAN: {f1\_dbscan:.2f}')

we get out put as:

A screenshot of a computer

Description automatically generated

# Plot comparison of the metrics

metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']

lof\_scores = [accuracy\_lof, precision\_lof, recall\_lof, f1\_lof]

knn\_scores = [accuracy\_knn, precision\_knn, recall\_knn, f1\_knn]

ocsvm\_scores = [accuracy\_ocsvm, precision\_ocsvm, recall\_ocsvm, f1\_ocsvm]

iforest\_scores = [accuracy\_iforest, precision\_iforest, recall\_iforest, f1\_iforest]

kmeans\_scores = [accuracy\_kmeans, precision\_kmeans, recall\_kmeans, f1\_kmeans]

dbscan\_scores = [accuracy\_dbscan, precision\_dbscan, recall\_dbscan, f1\_dbscan]

Run it after that:

scores = pd.DataFrame({

    'Metric': metrics,

    'LOF': lof\_scores,

    'KNN': knn\_scores,

    'One-Class SVM': ocsvm\_scores,

    'Isolation Forest': iforest\_scores,

    'K-Means': kmeans\_scores,

    'DBSCAN': dbscan\_scores

})

Run it , after that:

scores.set\_index('Metric', inplace=True)

after that:

scores.plot(kind='bar', figsize=(14, 8), colormap='viridis')

plt.title('Comparison of Outlier Detection Methods')

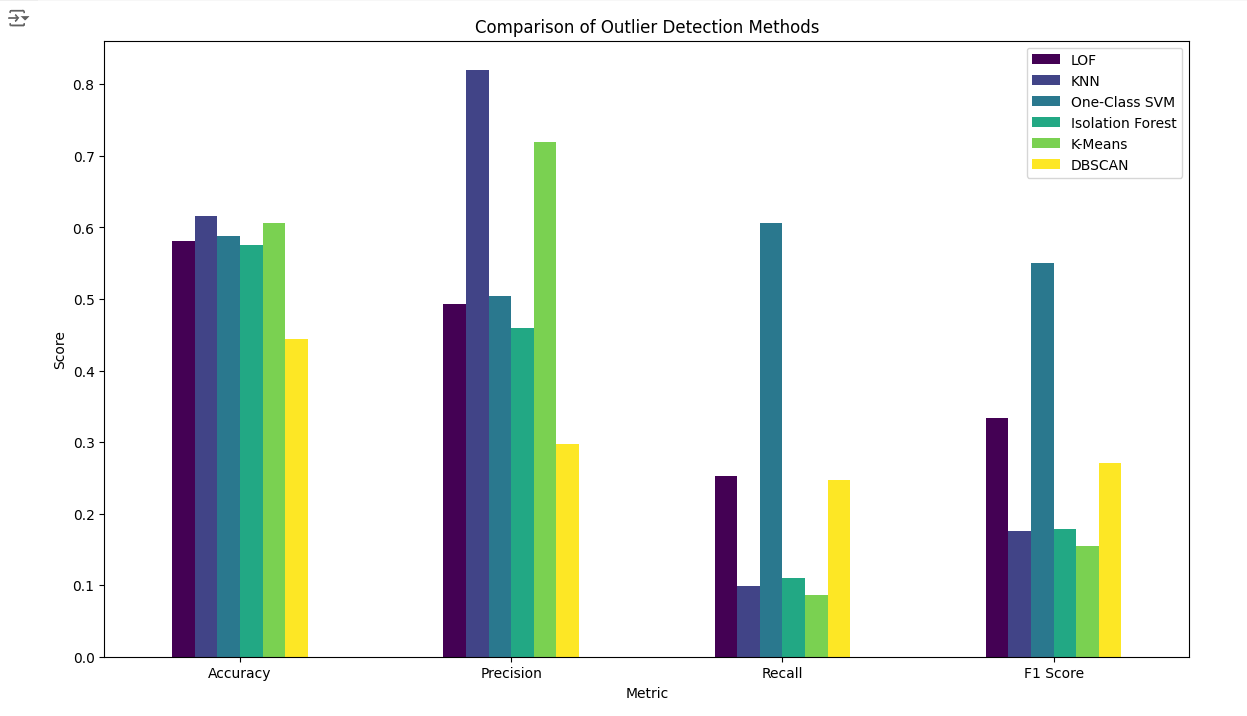
plt.ylabel('Score')

plt.xticks(rotation=0)

plt.legend(loc='best')

plt.show()

Run it, we will get our final output:



This is the final result that we get. Here we can see that in :

Accuracy: KNN method is highest and DBSCAN is lowest

Precision: KNN is highest and DBSCAN is lowest

Recall: -Class SVM is highest and K-Means is lowest

F1 score: One-Class SVM is highest, and K-Means is lowest

From this observation we can say that KNN is the best approach among all for outlier detection, because it has highest in both Accuracy and Precision. The other best approach we can consider One-Class SVM because it stands high in both Recall and F-1 score.

**Criteria for Selecting the Best Approach:**

**Performance Metrics:**

Choosing the method with the highest values in the metrics that are most important for your application (e.g., high precision if false positives are costly, high recall if missing outliers is costly, or high F1 score for a balance).

**Computational Efficiency:**

Some methods may be more computationally intensive than others. If working with large datasets, consider the scalability and computational cost.

**Interpretability:** Some methods are easier to understand and explain than others. For example, K-Means is easier to interpret than Isolation Forest.

**Robustness**: The method should be robust to different types of anomalies and not overly sensitive to the parameters.

**Potential Limitations of the Proposed Approaches :**

* **Local Outlier Factor (LOF):**
* **Limitations:** Sensitive to the choice of the number of neighbors (n\_neighbors). Performance can degrade with high-dimensional data due to the curse of dimensionality.
* **Scalability:** Computationally expensive for large datasets.
* **One-Class SVM:**
* **Limitations:** Sensitive to the choice of kernel and hyperparameters. Can be less effective with large-scale datasets.
* **Scalability:** High computational cost for training, especially with non-linear kernels.
* **Isolation Forest:**
* **Limitations**: Requires setting contamination parameter which might not be known a priori. Assumes anomalies are few and different from normal instances.
* **Scalability:** Generally efficient and scalable to large datasets.
* **K-Means Clustering:**
* **Limitations:** Assumes clusters are spherical and equally sized. Requires setting the number of clusters (k). Not specifically designed for anomaly detection.
* **Scalability:** Efficient for large datasets but can struggle with high-dimensional data.
* **DBSCAN:**
* **Limitations:** Sensitive to the choice of parameters (eps and min\_samples). Performance can degrade in high-dimensional data.
* **Scalability:** Can be efficient but may struggle with very large datasets depending on parameter settings.

**Overall Conclusion**:

* **KNN** is the best performer in terms of precision, making it suitable for applications where minimizing false positives is crucial.
* **KMeans** provides the best balance between precision and recall, evidenced by its highest F1 score, making it a strong general-purpose choice for outlier detection.
* **DBSCAN** performs well in recall and F1 score but has lower precision, indicating it might be better at finding outliers but also includes more false positives.
* **One-Class SVM** and **Isolation Forest** offer a balanced performance in accuracy and precision but might not be the best choices when high recall is required.

The choice of the best method depends on the specific requirements of the application, such as the importance of precision versus recall.