Thaddeus Backus, Austin Webb

# Predicting the Class of an Internet Connection Request Using SVM and SGD

## Introduction

In the domain of internet traffic classification, Support Vector Machine (SVM) and Stochastic Gradient Descent (SGD) models have emerged as invaluable tools, celebrated for their exceptional predictive accuracy and adaptability, especially to non-linear problems. These machine learning algorithms excel in the realm of network security, effectively navigating the complexities of high-dimensional network traffic datasets. They adeptly uncover intricate patterns and relationships hidden within the data. The significance of SVM and SGD lies in their capacity to empower network administrators and security analysts to proactively identify and respond to potential threats or anomalies in internet traffic, thereby bolstering network defenses and ensuring the uninterrupted flow of online operations.

The primary objective is to present a comprehensive approach to internet traffic classification, highlighting the application of state-of-the-art machine learning models and techniques to address real-world challenges in network security. By leveraging the predictive capabilities of SVM and SGD, this endeavor contributes to the advancement of internet traffic security, leading to a new era of more robust and proactive cybersecurity practices in the ever-evolving landscape of online connectivity.

## Methods

The foundation of our analysis was a dataset consisting of internet connection requests. The given data contained 65,532 rows or separate connection requests, 11 variables or feature, and “Action”, the target variable. There was no missing data present in this dataset. Figure 1 shows the distribution of the target variables. The classes contained in this dataset were highly imbalanced. The majority of internet connection requests, 37,640, were labeled ‘allow’. This was followed by ‘deny’ with 14,987, ‘drop’ had 12,851 cases, and only 54 occurrences of ‘reset-both’. The pronounced imbalance, particularly the ‘reset-both’ class, can pose challenges during the model training phase. The scarcity of this class may hinder the classifiers’ ability to accurately predict this class, potentially leading to skewed performance evaluation.

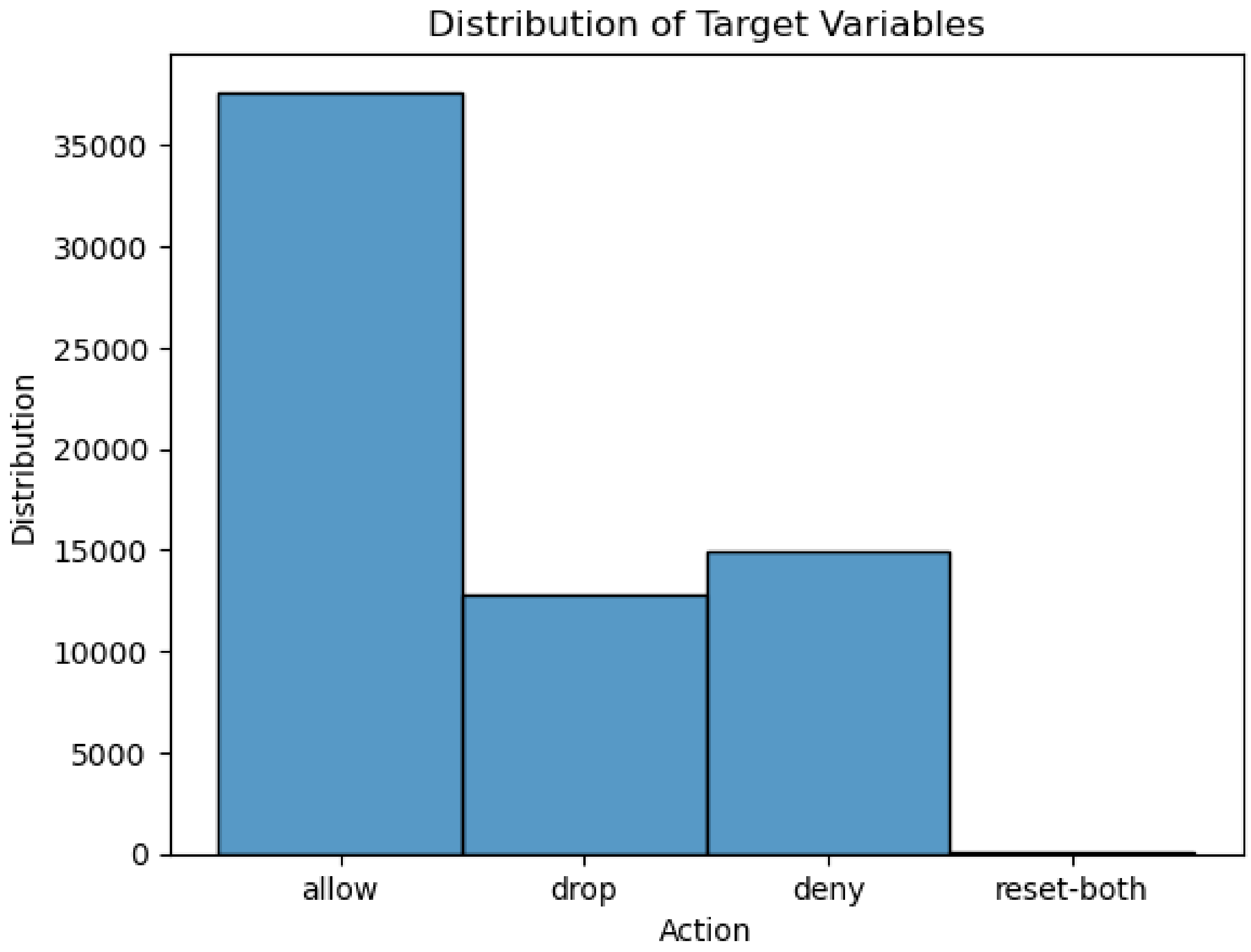

*Figure 1: Distribution of target classes in "Action" column*

The remaining features that were used to predict “Action” were Source, Destination, NAT Source, and NAT Destination ports, bytes, bytes sent and received, packets, elapsed time, packets sent, and packets received. Other than the Port features, all of these are continuous numeric features.

Data preprocessing was one of the most critical aspects of this project, for it can significantly impact the performance of classification models. To prepare the data the Port features were One-Hot Encoded, and the “Action” was re-encoded to replace the class names with numerical values. The main motivation behind this label encoding was to simplify the classification process, due to most machine learning models requiring numerical values.

The nest step in preprocessing came from the categorical nature of the Port features. These features were one-hot encoded to allow for each unique category, or in this case port number, to have its own column with a 1 representing a connection that utilized that specific port. This expanded the dataset from an initial 11 features to 57,960 columns. This approach allowed for the preservation of the categorical nature of these attributes while also making them suitable for SVM and SGD classifiers.

To further enhance the efficiency and speed of the models, the dataset was converted into a sparse matrix. The conversion involved making the data Compressed Sparse Row (CSR) matrix. A CSR matrix stores the data in a memory efficient way, which is particularly advantageous for datasets with a lot of zero values, which in this case came from the one-hot encoded features. This matrix reduces memory consumption and speeds up matrix operations during the model training.

The dataset was then divided into two distinct sets: the training and the test set. The training set consisted of 80% of the data, and served as the foundation upon which the models were built. This set allowed the models to learn and understand the patterns in the internet traffic features, enabling them to make unbiased predictions. The remaining 20% of the data was assigned to the testing set and played a critical role in the evaluating the models' predictive performance on data they had not seen yet.

The Linear SVC model, which in this case is an SVM model, used balanced class weights with dual optimization enabled.  A small set of values for C, the regularization parameter were tested to optimize performance. The C parameter, which controls the trade-off between maximizing the margin and minimizing the classification error, is a crucial hyperparameter for the SVM model. The values of C tested were 0.001, 0.01, 0.1, 1, 10, and 100 and the value with the overall best performance on predicting the test set was chosen. A C value of 10 had slightly better performance and was chosen for the final model.

For the SGD model, a grid search with cross validation was performed to help select the optimal parameters. The grid search tested both hinge and log-loss for the loss function and alpha values of 0.0001, 0.001, 0.01, 0.1, 1, 10, and 100. The alpha parameter controls the regularization term where higher is stronger. It was also used in this model to compute the learning rate. The results of the grid search had the best parameters as the loss function being hinge and an alpha value of 0.0001. This model also utilized early stopping to save time if the model's performance stopped increasing after 5 iterations and balanced class weights.

## Results

The SVM model had over 99% accuracy, precision, and recall when predicting 'allow', 'deny', and 'drop'. However, the model had 0% for all scores when the class was 'reset-both'. This was expected due to the small number of this class in the dataset. Figure 2 shows the confusion matrix for the predictions from SVM on the test set highlighting the small number of misclassifications. The SVM model left very little room for false positives or false negatives for the three main classes, which is evident in its impressive scores.

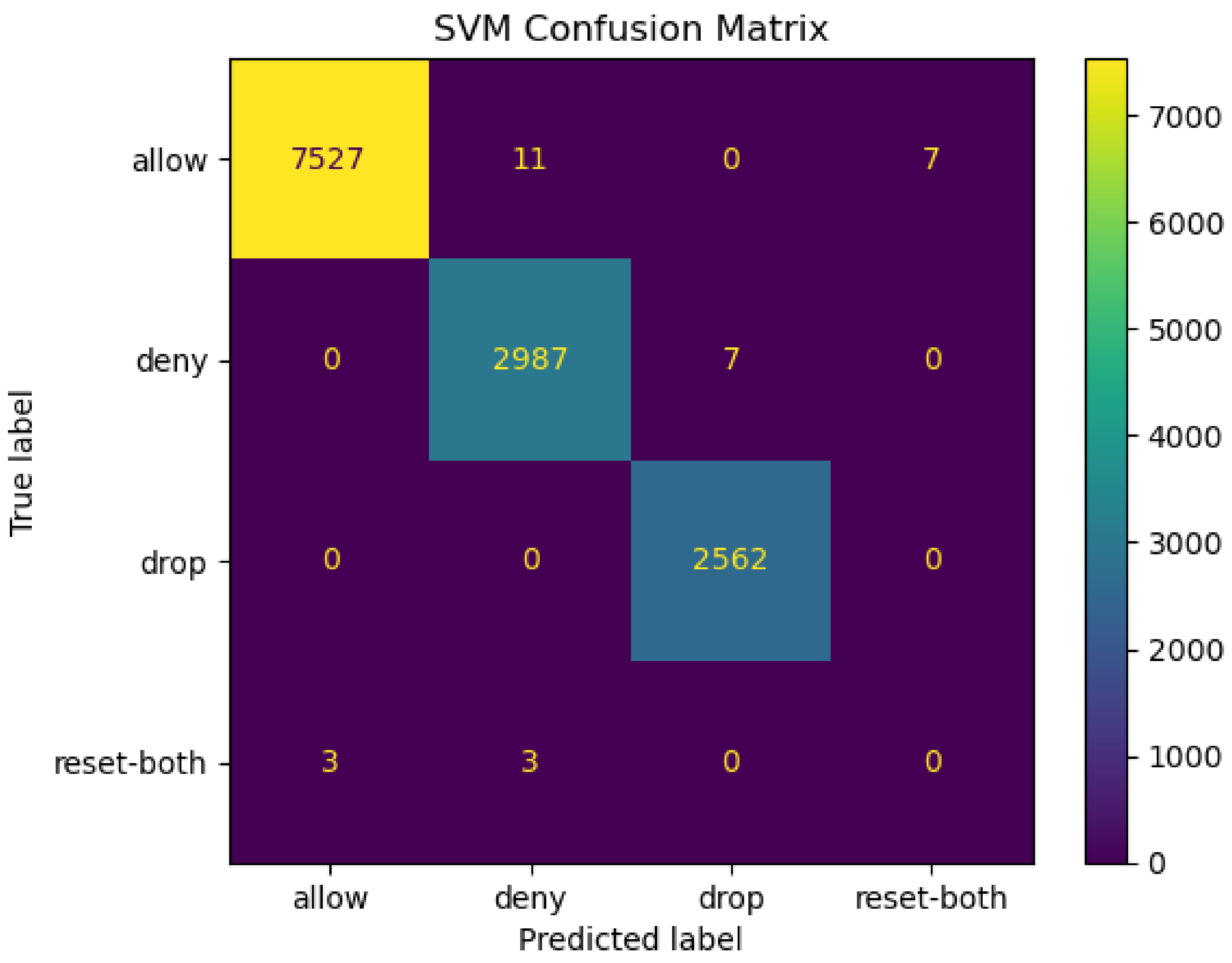

*Figure 2: SVM Confusion Matrix for Predictions on Test Set*

The SGD model also had over 99% accuracy, precision, and recall when predicting ‘allow’, ‘deny’, and ‘drop’. Once again, the model did not correctly predict any of the ‘reset-both’ classes. Figure 3 shows the confusion matrix for the predictions from SGD on the test set.

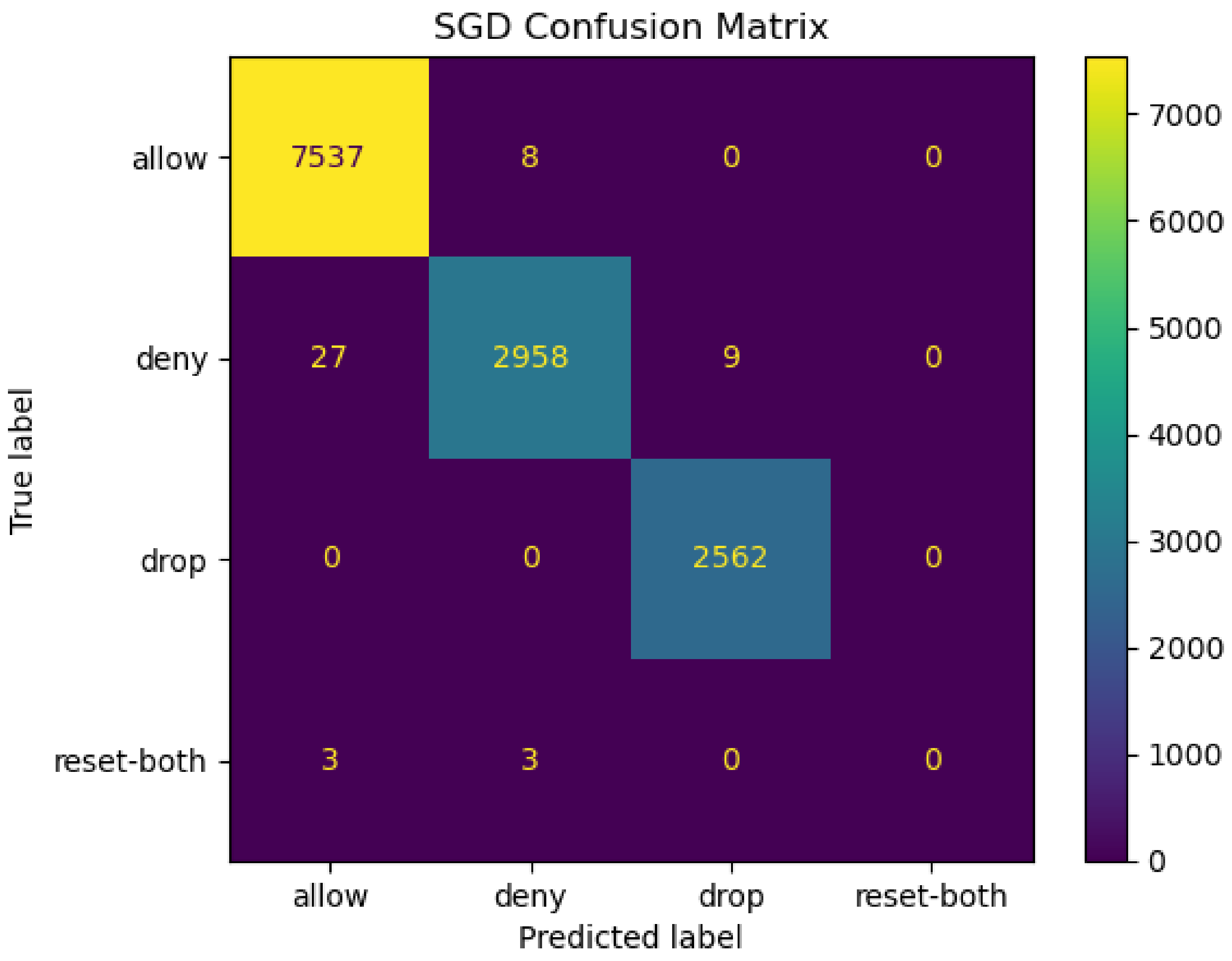

*Figure 3: SGD Confusion Matrix for Predictions on Test Set*

Table 1 shows the resulting accuracy, precision and recall for both models. For precision and recall, the weighted average of the scores for each class was used. Both the SVM and SGD models exhibited excellent accuracy, precision, and recall with very little difference between them. However, the SVM slightly outperformed the SGD model by about .15% in every metric looked at. When looking at the confusion matrices in Figure 2 and Figure 3 this is likely due to SGD missing 27 predictions that were 'deny' but the model predicted as 'allow'. Both models faced considerable challenges when classifying the 'reset-both' class, resulting in 0 scores across all the metrics. These challenges can primarily be attributed to the extreme class imbalance within this category, which affected both models' ability to classify correctly.

| **Scores** | **LinearSVC (SVM)** | **SGD** |
|---|---|---|
| Accuracy | 99.76% | 99.62% |
| Precision | 99.76% | 99.62% |
| Recall | 99.77% | 99.57% |

*Table 1: Prediction Scores for SVM and SGD*

After both models were trained, built, and evaluated on various scoring metrics, a look into feature importance was conducted. The feature importance analysis for both the SVM and SGD models plays a pivotal role in understanding the factors that significantly influence their predictions of connection requests. These models assign importance scores to each input feature, reflecting their contribution to the overall decision-making process. The analysis highlights which variables are the most crucial in determining which class the connection request will fall into.

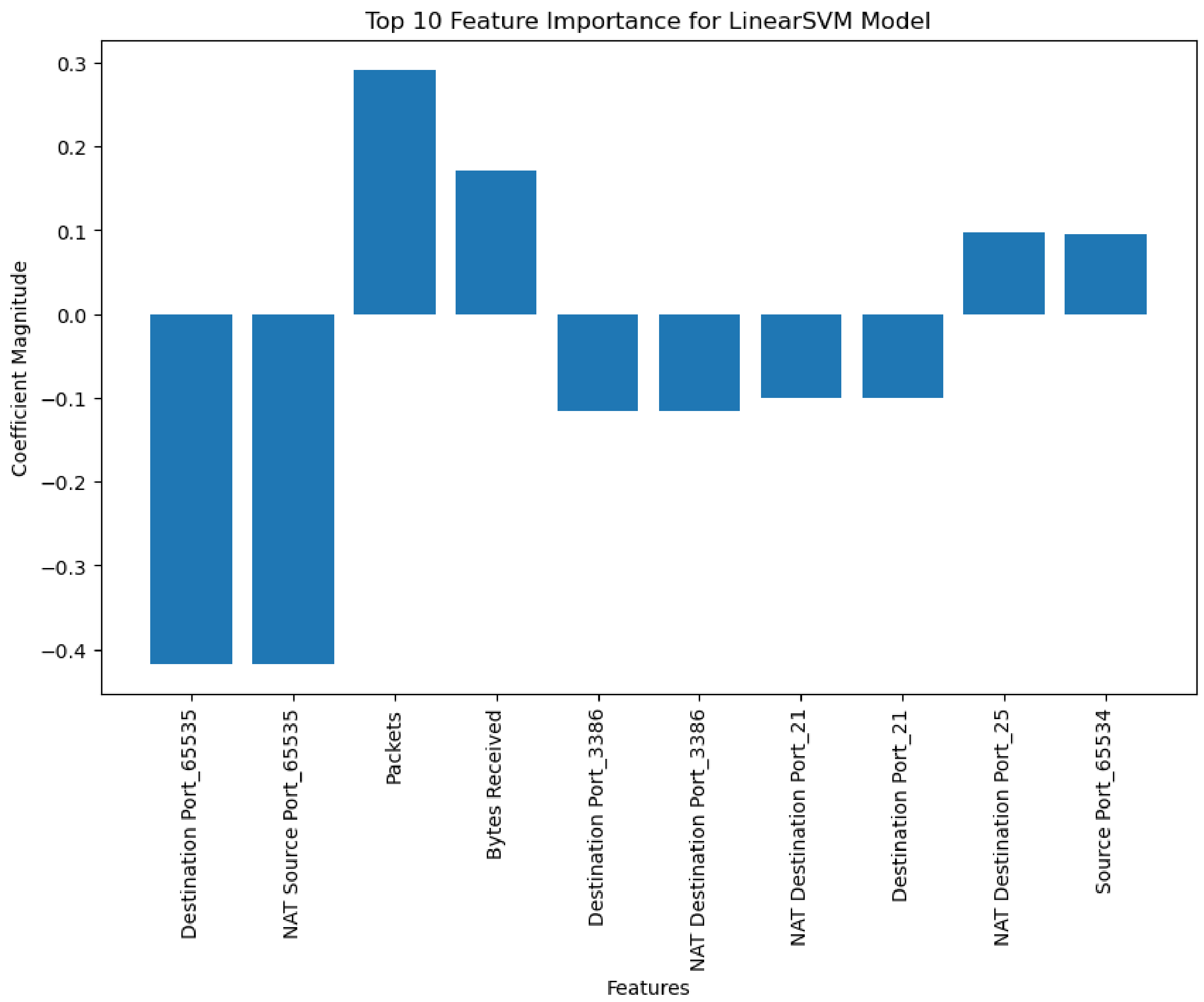

*Figure 4: Bar Plot to Visualize the Feature Importance of the SVM Model*

Several key features emerged as crucial contributors to the SVM model's ability to accurately predict internet connection request actions shown in Figure 4. Notably, the NAT Source Port_65535 and Destination Port_65535 along with packets proved to be significant features. The high importance of these features emphasizes the relevance of NAT ports in classifying connection requests. More details about this specific port would likely need to be investigated to identify what it entails as well as what action it is most commonly associated with.

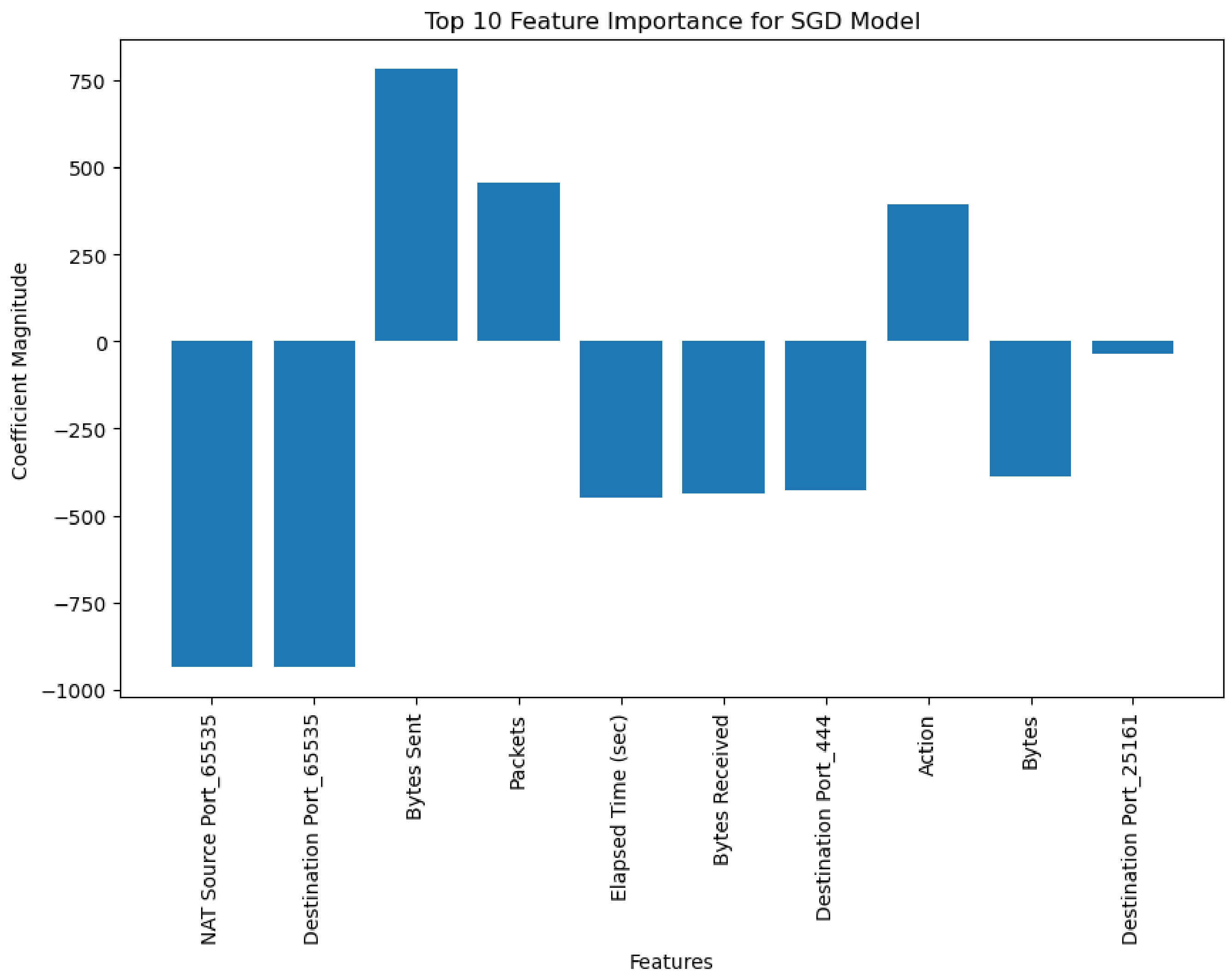

*Figure 5: Bar Plot to Visualize the Feature Importance of the SGD Model*

Among the top features identified in the SGD model were the same NAT ports from the SVM model but with Bytes sent being the third most influential feature. This can be seen in Figure 5. Interestingly, bytes sent was more important than either the total number of bytes or the bytes received.

## Conclusion

The results of this analysis demonstrate the formidable predictive capabilities of both the SVM and SGD models in identifying internet traffic classification. These models exhibit remarkable high levels of accuracy, precision, and recall, underlining their effectiveness in making precise classifications and minimizing false alarms when assessing internet traffic. These models performed extremely well on the three classes making up most of the data but struggled with the much smaller fourth class of 'reset-both'. This is likely due to the 'reset-both' class making up 0.08% of the dataset. The SVM model had an accuracy of 99.76% signifying its proficiency in accurate internet class predictions. However, the SGD model performs almost identically, with an accuracy of 99.62%.

While SVM emerges as the marginally better performer, it is essential to emphasize that both models are exceptionally close in terms of their predictive capabilities. The differences in performance metrics are relatively small and both models could be considered for this task. The choice between SVM and SGD should also consider factors beyond performance, such as model interpretability, computational efficiency, and ease of implementation.

## Appendix A

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import numpy as np
from scipy.sparse import csr_matrix

from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import LinearSVC
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score, confusion_matrix,
ConfusionMatrixDisplay

df = pd.read_csv('/Users/taddbackus/School/fall23/qtw/cs5/log2.csv')

df.isna().sum()

for column in df.columns:
    unique_values = len(df[column].unique())
    print(column,'=',unique_values)

print(df['Action'].value_counts())
sns.histplot(data=df, x='Action')
plt.xlabel('Action')
plt.ylabel('Distribution')
plt.show()

sns.histplot(data=df, x='Source Port')
plt.xlabel('Source Port')
plt.ylabel('Distribution')
plt.show()

sns.histplot(data=df, x='Destination Port')
plt.xlabel('Destination Port')
plt.ylabel('Distribution')
plt.show()
```

```
sns.histplot(data=df, x='NAT Source Port')
plt.xlabel('NAT Source Port')
plt.ylabel('Distribution')
plt.show()

sns.histplot(data=df, x='NAT Destination Port')
plt.xlabel('NAT Destination Port')
plt.ylabel('Distribution')
plt.show()

le = LabelEncoder()
df['Action'] = le.fit_transform(df['Action'])
mapping = dict(zip(le.classes_, le.transform(le.classes_)))
print('Mapping:\n',mapping)

columns_to_encode = ['Source Port', 'Destination Port', 'NAT Source Port', 'NAT Destination Port']
df_encoded = pd.get_dummies(df, columns=columns_to_encode, prefix=columns_to_encode)
print(df_encoded)

df_encoded = df_encoded.apply(lambda col: col.astype(int))
print(df_encoded)

X = df_encoded.drop(columns=['Action'])
y = df_encoded['Action']
X_sparse = csr_matrix(X)
X_train, X_test, y_train, y_test = train_test_split(X_sparse,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=42)

cValues = [0.001, 0.01, 0.1, 1, 10, 100]
for c in cValues:
    svmModel = LinearSVC(dual=True,C=c)
    svmModel.fit(X_train,y_train)
    svmPred = svmModel.predict(X_test)
    print('=============================')
    print('C',c)
    print(classification_report(y_test,svmPred))
    print(confusion_matrix(y_test,svmPred))
    print('=============================')

svmModel = LinearSVC(dual=True,class_weight='balanced',C=10)

svmModel.fit(X_train,y_train)
svmPred = svmModel.predict(X_test)
svmAcc = accuracy_score(y_test, svmPred)
svmRecall = recall_score(y_test, svmPred, average='weighted')
svmPrec = precision_score(y_test, svmPred, average='weighted')
print('SVM Accuracy:', svmAcc)
print('SVM Recall:', svmRecall)
print('SVM Precision:', svmPrec)
```

```
svmConfMatrix = confusion_matrix(y_test,svmPred)
print(classification_report(y_test,svmPred))
disp = ConfusionMatrixDisplay(confusion_matrix=svmConfMatrix,display_labels=le.classes_)
disp.plot()
plt.title('SVM Confusion Matrix')
plt.show()

coefficients = svmModel.coef_[0]

feature_names = df_encoded.columns


non_zero_indices = np.where(coefficients != 0)[0]
non_zero_coefficients = coefficients[non_zero_indices]

sorted_indices = np.argsort(np.abs(non_zero_coefficients))[::-1][:10]
top_coefficients = non_zero_coefficients[sorted_indices]
top_feature_indices = non_zero_indices[sorted_indices]

top_feature_names = [feature_names[i] for i in top_feature_indices]

plt.figure(figsize = (10, 6))
plt.bar(range(len(top_feature_names)),
    top_coefficients,
    tick_label = top_feature_names)

plt.title('Top 10 Feature Importance for LinearSVM Model')
plt.xlabel('Features')
plt.ylabel('Coefficient Magnitude')

plt.xticks(rotation = 90)

plt.show()

sgdModel = SGDClassifier(early_stopping=True)

params = {'loss':['hinge','log_loss'],
     'alpha':[0.0001,0.001,0.01,0.1,1,10,100],
     }
sgdGS = GridSearchCV(estimator=sgdModel,
          param_grid=params,
          cv=5,
          scoring='accuracy')
sgdGS.fit(X_train,y_train)
bestModel = sgdGS.best_estimator_
bestParams = sgdGS.best_params_
bestScore = sgdGS.best_score_

print(bestScore)
print(bestParams)
print(cross_val_score(bestModel,X_sparse,y,cv=5))

sgdModel2 = SGDClassifier(early_stopping=True, alpha=0.0001, loss='hinge')
```

```
sgdModel2.fit(X_train,y_train)
sgdPred = sgdModel2.predict(X_test)
sgdAcc = accuracy_score(y_test, sgdPred)
sgdRecall = recall_score(y_test, sgdPred, average='weighted')
sgdPrec = precision_score(y_test, sgdPred, average='weighted')
print('SGD Accuracy:', sgdAcc)
print('SGD Recall:', sgdRecall)
print('SGD Precision:', sgdPrec)

print(classification_report(y_test, sgdPred))
sgdConfMatrix = confusion_matrix(y_test, sgdPred)
disp = ConfusionMatrixDisplay(confusion_matrix=sgdConfMatrix,display_labels=le.classes_)
disp.plot()
plt.title('SGD Confusion Matrix')
plt.show()

coefficients = sgdModel2.coef_[0]

your_feature_names = df_encoded.columns


non_zero_indices = np.where(coefficients != 0)[0]
non_zero_coefficients = coefficients[non_zero_indices]

sorted_indices = np.argsort(np.abs(non_zero_coefficients))[::-1][:10]
top_coefficients = non_zero_coefficients[sorted_indices]
top_feature_indices = non_zero_indices[sorted_indices]

top_feature_names = [feature_names[i] for i in top_feature_indices]

plt.figure(figsize = (10, 6))
plt.bar(range(len(top_feature_names)),
    top_coefficients,
    tick_label = top_feature_names)

plt.title('Top 10 Feature Importance for SGD Model')
plt.xlabel('Features')
plt.ylabel('Coefficient Magnitude')

plt.xticks(rotation = 90)

plt.show()
```