2023

DS 7333 | Quantifying the World

CASE STUDY 5 | FIREWALL CLASSIFICATION

JOEY HERNANDEZ | DANNY CHANG

Introduction

Background

In today's technologically advanced and interconnected world, the management of data traffic and ensuring the security of a network infrastructure is paramount. Large corporations, especially those heavily reliant on online platforms, witness a significant amount of firewall interactions daily. These firewalls act as gatekeepers, determining which requests should be granted access and which should be denied, to protect the integrity and confidentiality of a company's data and resources.

The company for our project is one large-scale entity that faces these challenges. Historically, their approach to managing these interactions has been manual, necessitating considerable manpower and resources. This traditional method not only leads to increased operational costs but also leaves room for human error, potentially compromising the security of the entire system. As the volume of requests and data interactions continues to rise, it is crucial to rethink and reimagine the process to ensure scalability and efficiency.

Objective and Scope

The primary objective of this project is to develop an intelligent, autonomous system that can accurately and efficiently classify incoming firewall requests and decide whether to allow, deny, drop, or reset both connections. This system is intended to replace the current manual method, which is resource-intensive and prone to human error, with a more advanced, automated, and reliable solution. To achieve this, the project will employ machine learning algorithms, particularly Support Vector Machines (SVC) and Stochastic Gradient Descent (SGD) Classifier, to learn from the historical data and make informed decisions in real-time.

Scope

Data Utilization:

- Utilize the available dataset of approximately 60,000 rows, which contains historical decisions on firewall requests.
- Address the challenge of class imbalance, particularly with the "reset-both" class, which has only about 50 occurrences, thereby posing a risk of model overfitting towards other classes.

Model Development and Tuning:

- Implement Support Vector Machine (SVC) models, renowned for their effectiveness in high-dimensional spaces and adaptability in creating complex decision boundaries.

    - Focus on tuning the 'C' parameter in SVC, which controls the trade-off between achieving a low training error and a low testing error that is crucial for the model's effectiveness in generalizing unseen data.
- Employ the Stochastic Gradient Descent (SGD) Classifier, known for its efficiency in large-scale data handling and versatility in optimizing different loss functions and penalties.
    - Fine-tune the 'alpha' parameter, which is a constant that multiplies the regularization term and inversely scales with the learning rate, influencing model convergence and accuracy.

Classification Strategy:

- Develop a model capable of classifying requests into one of the following categories: "allow," "deny," "drop," and "reset-both."
- Implement strategies to handle class imbalance, such as resampling the dataset or applying different weights during the training process, ensuring that the "reset-both" category is treated accurately.

Performance Evaluation:

- Upon development, the model will undergo rigorous testing to evaluate its accuracy, speed, and reliability in classifying and processing requests.
- Performance metrics will not only focus on overall accuracy but also on the precision, recall, and F1-score for each class, especially for the underrepresented "reset-both" category.

Data Inspection

Upon preliminary inspection of the dataset provided for our use case, it's evident that the data quality is fair. Some key observations include:

1. **Duplicates:** The dataset does not contain any duplicate entries, ensuring that each data point is unique and will contribute to the robustness of the model.
2. **Missing Values:** There are no missing values present within the dataset, suggesting that the data collection process has been thorough and systematic.
3. **Integrity:** The data appears to be intact, with no evident signs of corruption or anomalies that might jeopardize the model-building process.
4. **Class Imbalance:** One noteworthy aspect of the dataset is the class imbalance observed in the target variable, specifically in the "reset-both" category, which only registers about 50 occurrences. While the data is of high quality, this imbalance poses a challenge for model training, as it may lead to overfitting or underrepresentation of this specific class.

Feature Analysis and Treatment:

While the dataset encompasses various features, attention is required for a specific subset of features: Source port, destination port, NAT source port, and NAT Destination port. At a passing glance, these features appear numerical. However, based on domain knowledge and the context in which these ports operate, it's more apt to treat them as categorical variables.

To efficiently manage these features without inflating the feature space, a binning strategy has been employed. The binning criteria adopted are as follows:

- **Bins:** [0, 1023, 49151, 65535]

Corresponding to these bins, the data points are categorized into:

- **Labels:** ['well-known', 'registered', 'dynamic/private']

This binning strategy allows for meaningful categorization while keeping the dataset compact and interpretable. For instance, ports ranging from 0 to 1023 are typically "well-known" ports designated for standard services, whereas the ones from 1023 to 49151 are "registered" ports. Lastly, ports from 49151 to 65535 fall under the "dynamic/private" category.

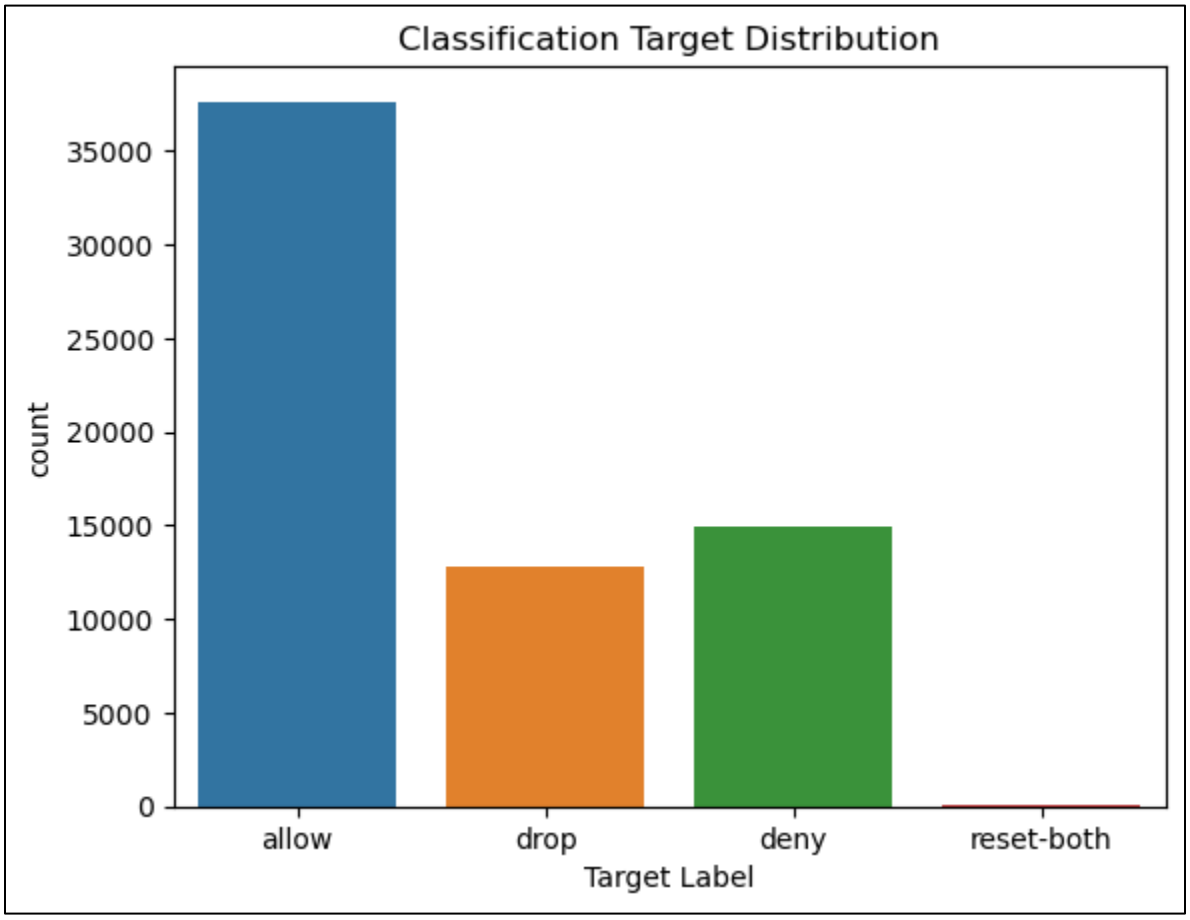
One Hot Encoding:

After performing the feature binning for the various ports, we then proceeded to one hot encode the features to represent them in a format suitable for machine learning models. One Hot Encoding (OHE) is a process where categorical variables are converted into a form that can be provided to ML algorithms to improve predictions.

Target Variable Inspection

As mentioned above, in our analysis spanning the entire dataset, we meticulously examined the distribution of our target variable, differentiating between "allow", "deny", "drop", and "reset-both" categories. The table highlights a pronounced class imbalance, with "reset-both" instances forming a minuscule fraction of the total data. To break it down: "allow" commands a dominant 62.73%, "deny" stands at 24.98%, "drop" at 21.42%, while "reset-both" barely touches 0.09% of the dataset. Such a stark disparity, especially in the "reset-both" category, emphasizes the need to craft our modeling approaches with precision to adeptly address this imbalance.

Figure 1: *Count plot illustrating the distribution of Classification targets.*



Description*: The count plot displays the class distribution for our target variable, highlighting the disparity between each of the four instances. This visualization aids in identifying class imbalances present in the dataset.*

Table 1: *Table detailing the distribution of target classes.*

Distribution of Target Variable		
Class	**Class Counts**	**Class Percentages**
Allow	37640	62.73%
Drop	14987	24.98%
Deny	12851	21.42%
Reset-Both	54	0.09%

Description*:* The table presents the distribution of occurrences of the data *target* variable. It succinctly highlights the number of instances for each class and the corresponding percentages.

Modeling

Support Vector Machines (SVC)

The Support Vector Machine (SVM) is a powerful and versatile supervised machine learning model capable of performing classification tasks. SVM works by finding the hyperplane that best divides the dataset into classes. In high dimensional spaces, this hyperplane can be thought of as a "line" that optimally separates data into respective categories.

For our analysis, we specifically employed the Linear Support Vector Classification (Linear SVC). This SVM variant is suitable for large datasets and performs linear classification, which means it seeks to classify data by finding the best linear separator.

Data Preparation and Splitting

Prior to model training, we partitioned our dataset using a train-test split, ensuring a training set that constitutes 80% of the entire dataset. Given the known class imbalances in our target variable, it was paramount to ensure that both training and test sets have proportional representations of each class. To achieve this, we utilized the stratify argument, which guarantees that the distribution of target classes is consistent across both training and test datasets.

Hyperparameter Tuning

A key hyperparameter for the SVM is the regularization parameter, commonly denoted as 'C'. The role of 'C' is to determine the trade-off between achieving a low classification error on the training dataset and maintaining a small margin. A smaller value of 'C' will look for a larger margin separating hyperplane, even if it misclassifies more points, whereas a larger 'C' prioritizes classifying all training examples correctly, potentially at the expense of a smaller margin.

For the optimal selection of the 'C' parameter, we employed a Randomized Search CV process. Instead of exhaustively exploring the entire parameter space, a randomized search samples a fixed number of parameter settings from the specified distributions. This approach offers a more efficient way to navigate the hyperparameter space, especially when dealing with a large dataset. For our model, we conducted 60 iterations to hone in on the most effective 'C' value.

Specific Configurations

Our implementation of the Linear SVC came with two additional noteworthy configurations:

1. **Dual = False**: By setting the dual parameter to False, we solved the primal optimization problem, which is more efficient for larger datasets since the number of samples often exceeds the number of features. This was selected based on the recommendations from the sklearn website: *If n_samples < n_features and optimizer supports chosen loss, multi_class and penalty, then dual will be set to True, otherwise it will be set to False*.
2. **Class Weight = 'Balanced'**: Due to class imbalances in our target variable, it was essential to weigh the classes so that the model does not exhibit bias toward the majority class. The 'balanced' option automatically adjusts weights inversely proportional to class frequencies, ensuring that each class contributes equally to the model's learning process.

In conclusion, our SVM-based approach, combined with meticulous data handling and hyperparameter tuning, aimed to derive meaningful patterns from the data while also addressing challenges such as class imbalance. The choice of Linear SVC catered to our need for efficiency and accuracy in handling a dataset of substantial size.

Results:

Confusion Matrix Analysis:

The confusion matrix is an essential tool for understanding the classification performance, as it presents a clear picture of true positives, false positives, true negatives, and false negatives for each class:

- **Allow**: Out of 7528 instances, the model correctly classified 7511 as 'allow' and misclassified 17 as 'deny'. This indicates a near-perfect prediction for this class.
- **Deny**: Of the 2998 'deny' instances, the model correctly predicted 2942. However, there were 48 instances wrongly classified as 'drop' and 6 as 'allow', along with 2 misclassifications as 'reset-both'.
- **Drop**: The model showcased excellent performance for the 'drop' category, correctly classifying all 2570 instances.
- **Reset-Both**: This was the most challenging class for our model, given its sparse representation in the dataset. Out of 11 instances, the model only correctly classified one, with 10 instances being erroneously labeled as 'deny'.

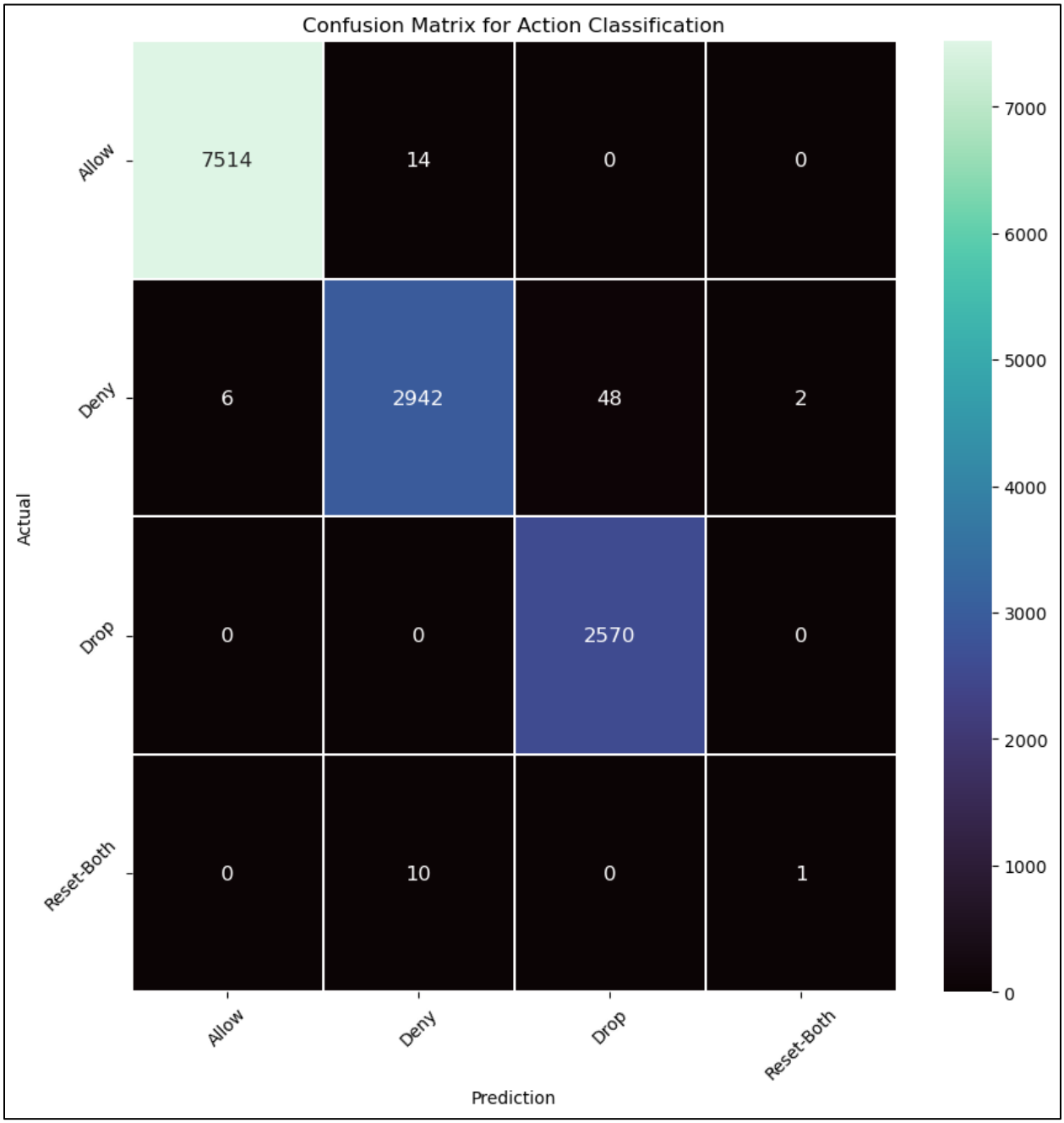
Classification Report:

The classification report provides a comprehensive overview of key metrics, including precision, recall, and F1-score for each class:

- **Allow**: The model achieved a perfect precision, recall, and F1-score of 1.00, demonstrating impeccable performance for this category.
- **Deny**: With a precision of 0.99, recall of 0.98, and an F1-score of 0.99, the model displayed robust performance, correctly identifying the vast majority of 'deny' instances.
- **Drop**: The model attained a precision of 0.98, matched with a perfect recall and an F1-score of 0.99, indicating a near-perfect classification.
- **Reset-Both**: Here, the model faced challenges, achieving a precision of 0.33, a recall of 0.09, and a subsequent F1-score of 0.14. This underscores the difficulty posed by the class imbalance inherent to the 'reset-both' category.

Furthermore, the model achieved an overall accuracy of 0.99, indicating its excellent ability to generalize. The weighted average precision, recall, and F1-score were also 0.99, reiterating the model's strong overall performance. However, the macro averages, which give equal weight to each class, highlight the challenges with the 'reset-both' category, with values of 0.83 for precision, 0.77 for recall, and 0.78 for the F1-score.

Figure 2: *A visual of the Linear SVC prediction matrix.*



Description*: The plot above illustrates the confusion matrix for the predictions made by the Linear SVC model. On the x-axis, we have the predicted labels, and on the y-axis, we have the true labels.*

Figure 3 : *Classification Report for the Linear SVC Classification Model*

Classification Report				
	Precision	**Recall**	**F1- Score**	**Support**
Allow	1.00	1.00	1.00	7528
Deny	0.99	0.98	0.99	2998
Drop	0.98	1.00	0.99	2570
Reset-Both	0.33	0.09	0.14	11
Accuracy			0.99	13107
Macro Avg	0.83	0.77	0.78	13107
Weighted Avg	0.99	0.99	0.99	13107

Description*: The classification report highlights the model's strong performance across most categories, with precision, recall, and F1-score exceeding 0.95 for "Allow", "Deny", and "Drop".*

Stochastic Gradient Descent Classifier (SGD)

The Stochastic Gradient Descent (SGD) classifier is a potent and adaptable supervised machine learning algorithm capable of executing classification tasks. At its core, SGD updates parameters iteratively in response to each training example, making it particularly efficient for large datasets. The goal of SGD is to minimize the loss function by adjusting weights based on the gradient of the loss with respect to each weight.

Data Preparation and Splitting

Prior to beginning the modeling process, we split our dataset via a train-test partitioning scheme, using 80% for training and the remaining 20% for validation. Given the inherent class imbalances in our dataset, maintaining a consistent representation of classes in both subsets was vital. To ensure this, the stratify argument was leveraged, preserving the distribution of the target classes across both the training and test datasets.

Hyperparameter Tuning

A crucial hyperparameter for the SGD Classifier is the learning rate, often denoted as alpha. This parameter regulates the step size at each iteration while moving toward a minimum of the loss function. The correct balance is imperative; too large an alpha might overshoot the minimum, while too small could lead to slow convergence or getting stuck in local minimums.

To optimize the selection of the alpha parameter, we deployed a Randomized Search CV strategy. Instead of a comprehensive exploration of all possible values, this approach samples from a predefined distribution of possible alpha values, providing a balance between accuracy and efficiency. Our tuning procedure involved 5-fold cross-validation, ensuring robustness by testing the model on various sub-samples of the training data.

Specific Configurations

Our instantiation of the SGD Classifier was characterized by a few distinct configurations:

1. **Loss = 'Log'**: Opting for the logarithmic loss function (or log loss) transforms our SGD Classifier into a logistic regression, allowing the model to estimate probabilities, thus furnishing more nuanced decisions.
2. **Penalty = 'L2'**: This selection applies L2 regularization to the model. Regularization penalizes complex models, providing a countermeasure against overfitting. The L2 penalty specifically targets the squared magnitudes of the coefficient, promoting feature weights to be small and smooth.
3. **Class Weight = 'Balanced'**: Given the uneven distribution of classes in our target variable, it was of paramount importance to address this disparity. The 'balanced' setting adjusts the weights automatically, in an inversely proportional manner relative to class frequencies. This ensures that the model does not inherently favor the majority class, promoting a more unbiased learning experience.
4. **Early Stopping = True**: This option enables the Early Stopping mechanism, which is a form of regularization aimed at preventing overfitting. By monitoring the model's performance on a validation set during training, Early Stopping halts the learning process once it detects that the performance has ceased to improve, or if it begins to degrade. This not only helps in avoiding overfitting, but also reduces the training time, as the model will stop learning as soon as it reaches an optimal state, rather than continuing to iterate until a predetermined number of epochs has been completed.
5. **Cross Validation:** The integration of the cross_val_score tool ensures a robust evaluation of the model's performance by splitting the dataset multiple times into different training and test sets. In essence, cross-validation divides the data into 'k' subsets, and the model is trained on 'k-1' of these subsets and tested on the remaining one. This process is repeated 'k' times, with each subset serving as the test set once. The benefit of this approach is that it reduces the variability associated with a single random train-test split, offering a more holistic perspective on how the model might perform on unseen data. The average score from all 'k' tests is generally considered a more reliable metric, mitigating the risk of overestimating the model's performance due to potential anomalies or biases in a single split.

Results:

Confusion Matrix Analysis: The confusion matrix provides critical insights into classification performance by detailing true positives, false positives, true negatives, and false negatives for each class:

- **Allow**: Out of 7,528 instances, the model accurately classified 7,414 as 'allow'. It misclassified 56 as 'deny', 58 as 'drop', and 0 as 'reset-both', showcasing an impressive prediction accuracy for this class.

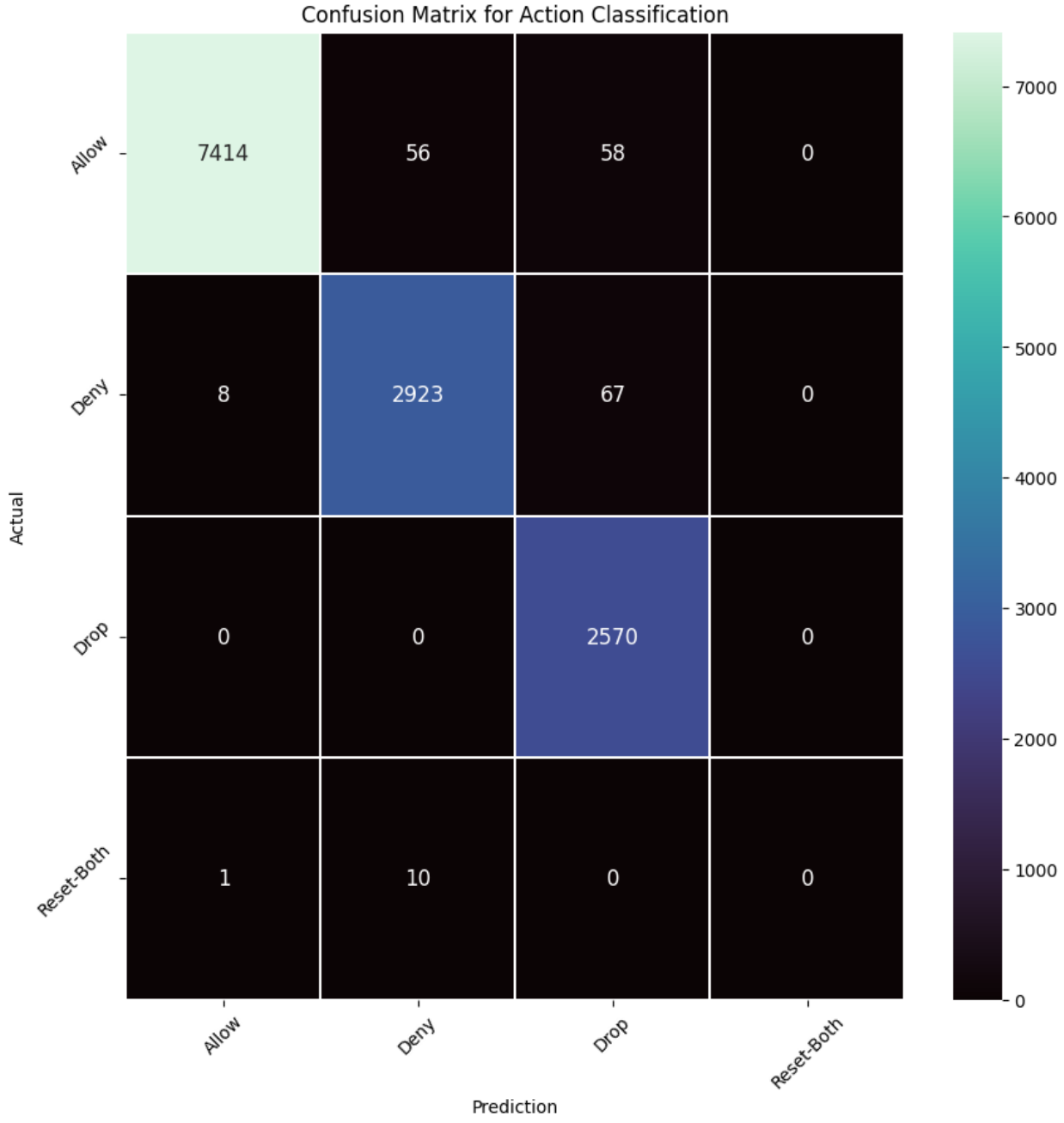
- **Deny**: From the 2,998 'deny' instances, the model rightly predicted 2,923. There were 67 instances mistakenly classified as 'drop', 8 as 'allow', and none as 'reset-both'.
- **Drop**: For the 'drop' category, the model exhibited outstanding performance by correctly classifying all 2,570 instances without any misclassifications.
- **Reset-Both**: The 'reset-both' category remained challenging for the model. Out of 11 instances, none were accurately classified, with 1 instance mislabeled as 'allow' and 10 as 'deny'.

Classification Report: The classification report provides a comprehensive analysis of key metrics, including precision, recall, and F1-score for each class across 5-fold split:

- Allow: Across the splits, the model consistently achieved high precision, recall, and F1-score values, typically around 1.00, 0.99, and 1.00 respectively, indicating excellent performance in identifying 'allow' instances.
- Deny: The model demonstrated strong precision and recall values, generally around 0.98 and 0.97-0.98 respectively, and an F1-score of 0.97-0.98, showcasing its effectiveness in categorizing 'deny' cases.
- Drop: The model maintained a high precision of around 0.94-0.97, paired with a perfect recall of 1.00 and an F1-score of 0.97-0.98, highlighting its proficiency in the 'drop' classification.
- Reset-Both: This class posed significant challenges for the model across all splits, yielding precision, recall, and F1-score values consistently at 0.00, reflecting the difficulties associated with this scarcely represented category.

Additionally, the model maintained a commendable overall accuracy of around 0.98-0.99 across the splits while our average comes out to 0.984. The weighted average metrics for precision, recall, and F1-score were consistently close to 0.98-0.99, emphasizing the model's overall effectiveness. However, the macro averages, which give equal importance to each class, revealed the model's struggles with the 'reset-both' class, typically resulting in values of 0.73-0.74 for precision, recall, and F1-score.

Figure 4: *A visual of the SGD Classification prediction matrix.*



Description*: The plot above illustrates the confusion matrix for the predictions made by the SGD model. On the x-axis, we have the predicted labels, and on the y-axis, we have the true labels.*

Figure 5: *Classification Report for the SGD Classification Model (Report 4 of the Stratified 5-fold split)*

Classification Report - SGD				
	Precision	**Recall**	**F1- Score**	**Support**
Allow	1.00	0.98	0.99	7528
Deny	0.97	0.97	0.97	2997
Drop	0.94	1.00	0.97	2570
Reset-Both	0.00	0.00	0.00	11
Accuracy			0.98	13106
Macro Avg	0.73	0.74	0.73	13106
Weighted Avg	0.98	0.98	0.98	13106

Description*: The classification report highlights the model's strong performance across most categories, with precision, recall, and F1-score exceeding 0.95 for "Allow", "Deny", and "Drop".*

Conclusion

In conclusion, as the digital landscape continues to expand, efficient and secure data traffic management is becoming ever more essential. Our project's intent was to transform a manual system into an intelligent, autonomous solution to meet the growing demands of today's interconnected world.

Employing both the Support Vector Machines (SVC) and the Stochastic Gradient Descent (SGD) Classifier, our endeavor yielded promising results. The SGD Classifier, for instance, showcased a remarkable accuracy rate, correctly classifying 7,414 out of 7,528 'allow' instances and all 2,570 'drop' instances. However, it struggled with the 'reset-both' category, misclassifying all 11 instances. On the other hand, the SVM-based approach displayed near-perfect predictions for the 'allow' class, with 7,514 correct out of 7,528. While it handled the 'deny' and 'drop' categories commendably, it faced challenges with the 'reset-both' category, accurately predicting none out of 11 instances.

These results underscore both the potential and the areas needing refinement. While our machine learning models exhibited significant advancements in automating firewall request classifications, the 'reset-both' category highlighted the necessity for ongoing model improvements.

Looking ahead, it is crucial for the company to engage in regular model evaluations and updates, ensuring that the system remains responsive to evolving data patterns. By successfully

integrating machine learning into their network management strategy, the company is not only poised to realize operational efficiencies but also enhance the overall robustness and security of its digital infrastructure.

Recommendations

1. **Ensemble Techniques:** Given that both SGD and SVM have their strengths and weaknesses, the company might consider deploying ensemble techniques. Combining the predictions of both models could harness the strengths of each and potentially yield a more robust overall performance.
2. **Hyperparameter Retuning:** As new data becomes available, consider running regular hyperparameter tuning sessions. Given that the optimal parameters may change with different datasets, this ensures that the models are always tuned to their best possible configurations.
3. **Scalability Concerns:** As the company grows and the volume of firewall interactions increases, it's vital to ensure that the infrastructure supporting these machine learning models can handle the increased load. This might involve scaling up computational resources or adopting cloud-based solutions.
4. **Address Class Imbalance:** Continuous monitoring of class distributions in incoming data is crucial. If significant shifts are noticed, the company might need to readdress the class weighting or even consider techniques like resampling to maintain model performance.
5. **Continuous Evaluation Metrics:** Beyond accuracy, other metrics like precision, recall, and the F1 score for each class should be continuously monitored. These metrics provide a more holistic view of model performance, especially when dealing with imbalanced datasets.

Appendix

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:
```
df = pd.read_csv('log2.csv')
```

In [ ]:
```
df.duplicated().sum()
```

Out[ ]:
```
8362
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(65532, 12)
```

In [ ]:
```
df.head()
```

Out[ ]:

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Action	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received
0	57222	53	54587	53	allow	177	94	83	2	30	1	1
1	56258	3389	56258	3389	allow	4768	1600	3168	19	17	10	9
2	6881	50321	43265	50321	allow	238	118	120	2	1199	1	1
3	50553	3389	50553	3389	allow	3327	1438	1889	15	17	8	7
4	50002	443	45848	443	allow	25358	6778	18580	31	16	13	18

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
Source Port             0
Destination Port        0
NAT Source Port         0
NAT Destination Port    0
Action                  0
Bytes                   0
Bytes Sent              0
Bytes Received          0
Packets                 0
Elapsed Time (sec)      0
pkts_sent               0
pkts_received           0
dtype: int64
```

Change to categorical type features

Source Port: bin into 3 categories well known, registerd, dynamic/private

Destination Port

NAT Source NAT Destination

Bin the columns into groups

In [ ]:
```
src_bins = [0,1023,49151,65535]
labels = ['well-known','registered','dynamic/private']

df['src_port_category'] = pd.cut(df['Source Port'], bins = src_bins,
                                 labels=labels, right = True, include_lowest=True)
```

In [ ]:
```
des_bins = [0,1023,49151,65535]
labels = ['well-known','registered','dynamic/private']

df['des_port_category'] = pd.cut(df['Destination Port'], bins = des_bins,
                                  labels=labels, right = True, include_lowest=True)
```

In [ ]:
```
nat_src_bins = [0,1023,49151,65535]
labels = ['well-known','registered','dynamic/private']

df['nat_src_category'] = pd.cut(df['NAT Source Port'], bins = nat_src_bins,
                                 labels=labels,include_lowest=True,right=True)
```

In [ ]:
```
nat_des_bins = [0,1023,49151,65535]
labels = ['well-known','registered','dynamic/private']

df['nat_des_category'] = pd.cut(df['NAT Destination Port'], bins=nat_des_bins,
```

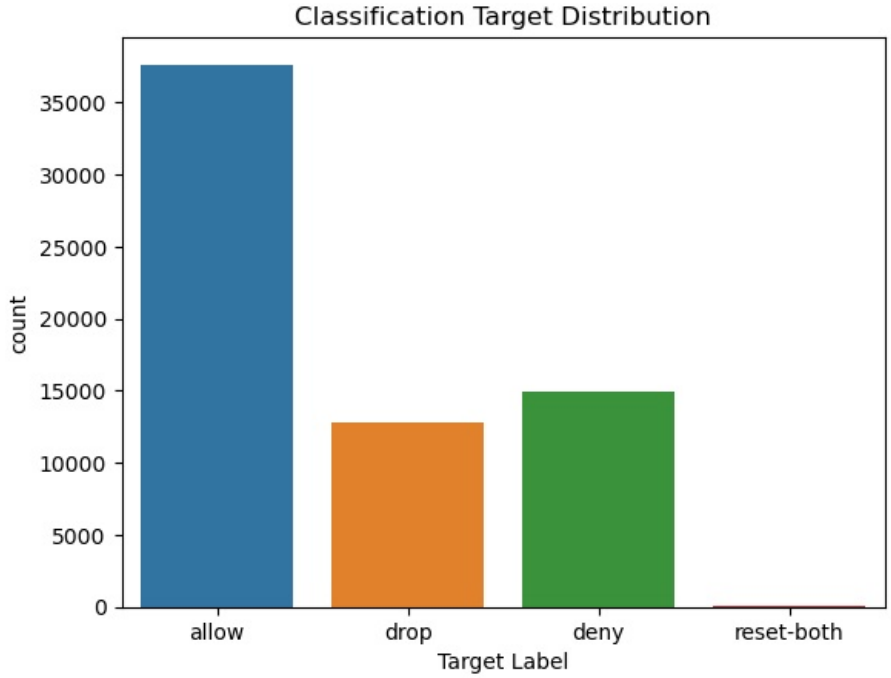
```
                                labels= labels, include_lowest=True,right = True)
```

Drop the original columns

In [ ]:
```
df.drop(['Source Port','Destination Port',
        'NAT Source Port','NAT Destination Port'], axis=1, inplace = True)
```

Look as target distribution

In [ ]:
```
sns.countplot(data = df, x ='Action')
plt.title('Classification Target Distribution')
plt.xlabel('Target Label')
plt.show()
```



In [ ]:
```
df['Action'].value_counts()
```

Out[ ]:
```
allow         37640
deny          14987
drop          12851
reset-both       54
Name: Action, dtype: int64
```

Dummy Data

In [ ]:
```
dummy_list = ['src_port_category',
              'des_port_category',
              'nat_src_category',
              'nat_des_category']

for col in dummy_list:
    dummies = pd.get_dummies(df[col],prefix=col, drop_first = True)
    df = pd.concat([df,dummies],axis =1)

df.drop(dummy_list,axis = 1, inplace=True)
```

Scaling Data

In [ ]:
```
from sklearn.preprocessing import StandardScaler

columns_for_scaling = ['Bytes','Bytes Sent','Bytes Received',
                       'Packets','Elapsed Time (sec)','pkts_sent','pkts_received']

subset_data = df[columns_for_scaling]

scaler = StandardScaler()
scaled_data = scaler.fit_transform(subset_data)

df[columns_for_scaling] = scaled_data
```

SVC

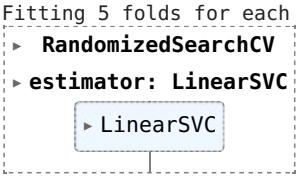
Train Test Split

In [ ]:
```
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split

X = df.drop('Action',axis=1)
y = df['Action']

X_train,X_test,y_train,y_test = train_test_split(X,y,
                                                 train_size = .8,
                                                 random_state=12,
                                                 stratify=y)
```

In [ ]:
```
from sklearn.model_selection import RandomizedSearchCV

param_dist = {'C': np.logspace(-3,2,1000)}



random_search = RandomizedSearchCV(LinearSVC(dual=False,class_weight='balanced'),
                                   param_distributions=param_dist,
                                   n_iter=60,
                                   cv=None,
                                   verbose=1,
                                   n_jobs=-1,
                                   random_state=12)

random_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

Out[ ]:

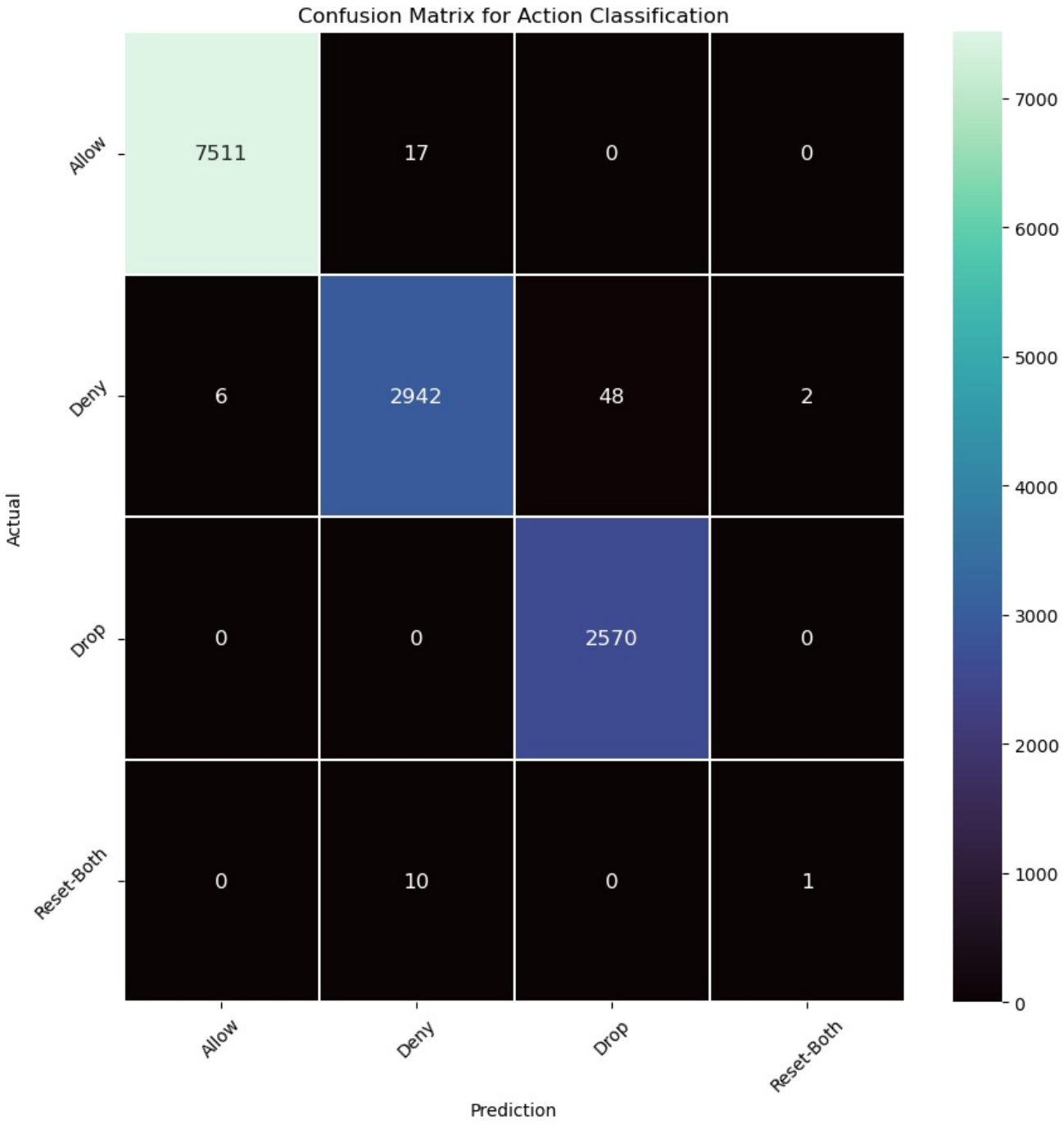


In [ ]:
```
from sklearn.metrics import confusion_matrix, classification_report

# Make predictions on the test data using the best estimator from RandomizedSearchCV
y_pred = random_search.best_estimator_.predict(X_test)

# Calculate the confusion matrix
confusion_mat = confusion_matrix(y_test, y_pred)

# Generate the classification report
class_report = classification_report(y_test, y_pred)

# Print the confusion matrix and classification report
print("Confusion Matrix:")
print(confusion_mat)

print("\nClassification Report:")
print(class_report)
```

```
Confusion Matrix:
[[7511   17    0    0]
 [   6 2942   48    2]
 [   0    0 2570    0]
 [   0   10    0    1]]

Classification Report:
              precision    recall  f1-score   support

       allow       1.00      1.00      1.00      7528
        deny       0.99      0.98      0.99      2998
        drop       0.98      1.00      0.99      2570
  reset-both       0.33      0.09      0.14        11

    accuracy                           0.99     13107
   macro avg       0.83      0.77      0.78     13107
weighted avg       0.99      0.99      0.99     13107
```

In [ ]:
```
plt.figure(figsize=(10,10))
sns.heatmap(confusion_mat,
            cmap= 'mako',
            annot=True,
            fmt='.0f',
            annot_kws={'size':12},
            linecolor='white',
```

```
            linewidths=.1,
            )
class_names = ['Allow','Deny','Drop','Reset-Both']
tick_position = [.5,1.5,2.5,3.5]
plt.xticks( ticks=tick_position,labels=class_names, rotation= 45)
plt.yticks(ticks=tick_position, labels = class_names, rotation = 45)
plt.title('Confusion Matrix for Action Classification')
plt.xlabel('Prediction')
plt.ylabel('Actual')
plt.show()
```



In [ ]:
```
from sklearn.model_selection import RandomizedSearchCV

param_dist = {'C': np.logspace(-3,2,1000)}

random_search = RandomizedSearchCV(LinearSVC(dual=False, class_weight='balanced'),
                                   param_distributions=param_dist,
                                   n_iter=60,
                                   cv=None,
                                   verbose=1,
                                   n_jobs=-1,
                                   random_state=12)

random_search.fit(X,y)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

Out[ ]:

▸ RandomizedSearchCV
▸ estimator: LinearSVC
▸ LinearSVC

In [ ]:
```
from sklearn.metrics import confusion_matrix, classification_report

# Make predictions on the test data using the best estimator from RandomizedSearchCV
y_pred = random_search.best_estimator_.predict(X)

# Calculate the confusion matrix
confusion_mat = confusion_matrix(y, y_pred)

# Generate the classification report
class_report = classification_report(y, y_pred)

# Print the confusion matrix and classification report
print("Confusion Matrix:")
print(confusion_mat)

print("\nClassification Report:")
print(class_report)
```

```
Confusion Matrix:
[[37541    95     2     2]
 [    8 14712   263     4]
 [    0     0 12851     0]
 [    0    42     0    12]]

Classification Report:
              precision    recall  f1-score   support

       allow       1.00      1.00      1.00     37640
        deny       0.99      0.98      0.99     14987
        drop       0.98      1.00      0.99     12851
  reset-both       0.67      0.22      0.33        54

    accuracy                           0.99     65532
   macro avg       0.91      0.80      0.83     65532
weighted avg       0.99      0.99      0.99     65532
```

Part 2 - Stochastic Gradient Descent

In [ ]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import SGDClassifier

param_dist = {'alpha': np.logspace(-3,2,1000)}

random_search = RandomizedSearchCV(SGDClassifier(class_weight='balanced', loss='log_loss', penalty='l2', early_
                                   param_distributions=param_dist,
                                   n_iter=60,
                                   cv=5,
                                   verbose=1,
                                   n_jobs=-1,
                                   random_state=12)

random_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

Out[ ]:

▸ RandomizedSearchCV
▸ estimator: SGDClassifier
▸ SGDClassifier

In [ ]:
```
from sklearn.metrics import confusion_matrix, classification_report

# Make predictions on the test data using the best estimator from RandomizedSearchCV
y_pred = random_search.best_estimator_.predict(X_test)

# Calculate the confusion matrix
confusion_mat = confusion_matrix(y_test, y_pred)
```

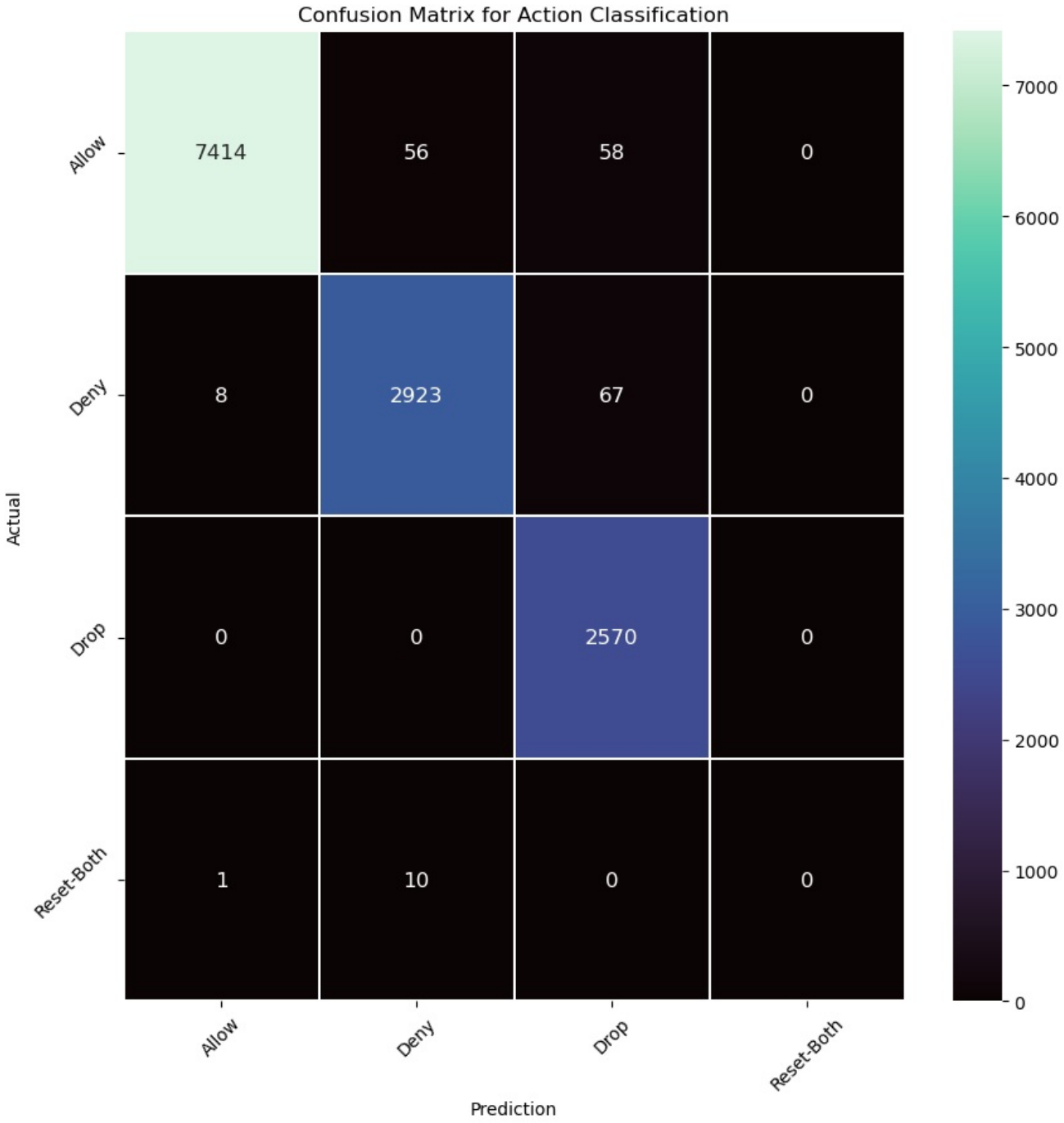
```python
# Generate the classification report
class_report = classification_report(y_test, y_pred)

# Print the confusion matrix and classification report
print("Confusion Matrix:")
print(confusion_mat)

print("\nClassification Report:")
print(class_report)
```

```
Confusion Matrix:
[[7414   56   58    0]
 [   8 2923   67    0]
 [   0    0 2570    0]
 [   1   10    0    0]]

Classification Report:
              precision    recall  f1-score   support

       allow       1.00      0.98      0.99      7528
        deny       0.98      0.97      0.98      2998
        drop       0.95      1.00      0.98      2570
  reset-both       0.00      0.00      0.00        11

    accuracy                           0.98     13107
   macro avg       0.73      0.74      0.74     13107
weighted avg       0.98      0.98      0.98     13107
```

```
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]:

```python
plt.figure(figsize=(10,10))
sns.heatmap(confusion_mat,
            cmap= 'mako',
            annot=True,
            fmt='.0f',
            annot_kws={'size':12},
            linecolor='white',
            linewidths=.1,
            )
class_names = ['Allow','Deny','Drop','Reset-Both']
tick_position = [.5,1.5,2.5,3.5]
plt.xticks( ticks=tick_position,labels=class_names, rotation= 45)
plt.yticks(ticks=tick_position, labels = class_names, rotation = 45)
plt.title('Confusion Matrix for Action Classification')
plt.xlabel('Prediction')
plt.ylabel('Actual')
plt.show()
```



Stochastic Gradient Descent with Early Stopping and CV

In [ ]:
```
random_search = RandomizedSearchCV(
    SGDClassifier(
        class_weight='balanced',
        loss='log_loss',
        penalty='l2',
        early_stopping=True,
        n_iter_no_change= 10
    ),
    param_distributions=param_dist,
    n_iter=60,
    cv=5,
    verbose=1,
    n_jobs=-1,
    random_state=12
)

random_search.fit(X, y)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

Out[ ]:

▸ **RandomizedSearchCV**
▸ **estimator: SGDClassifier**
▸ SGDClassifier

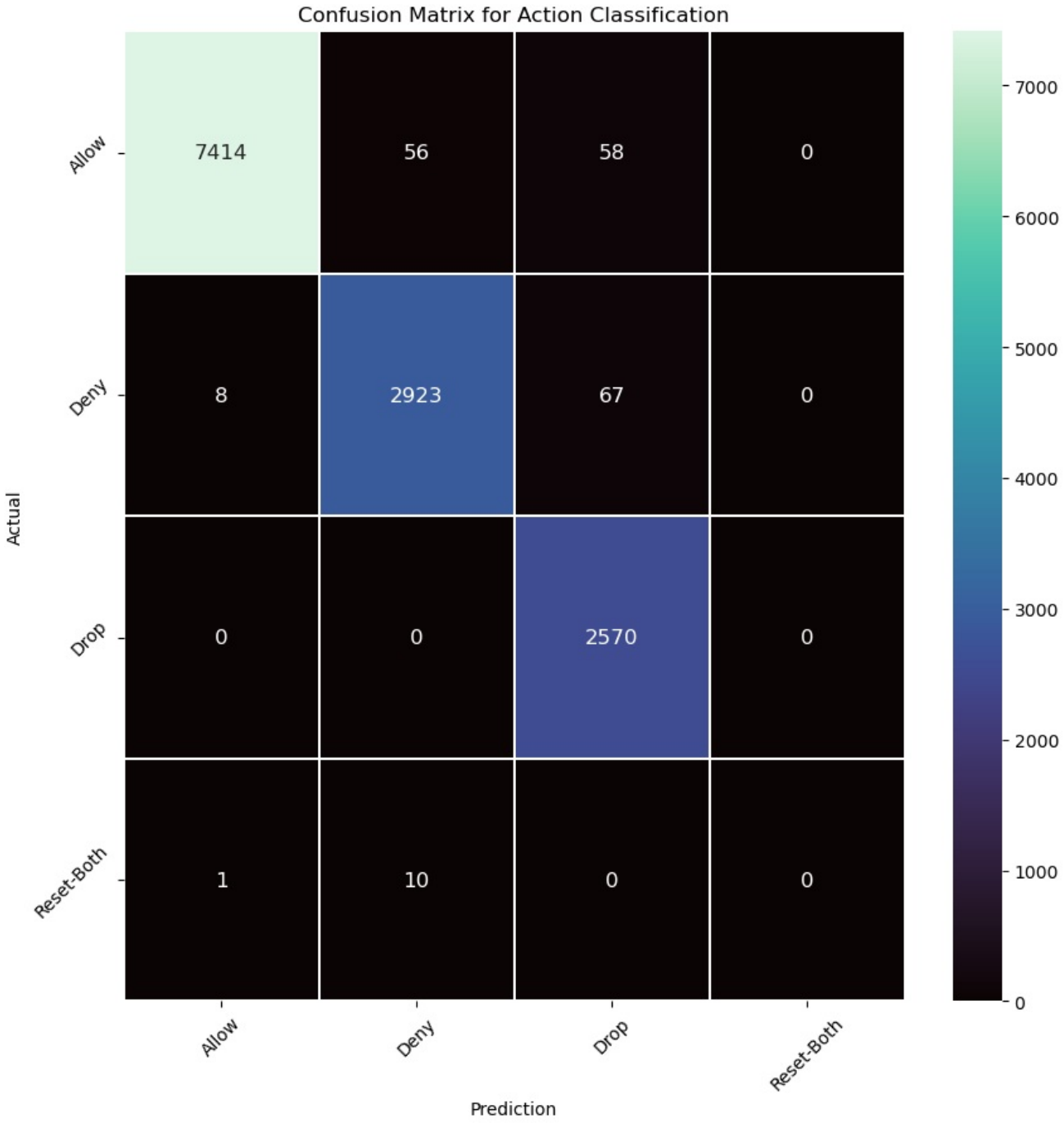
In [ ]:
```
from sklearn.model_selection import cross_val_score
```

```
from sklearn.metrics import make_scorer, accuracy_score

best_estimator = random_search.best_estimator_

cross_val_scores = cross_val_score(
    best_estimator,
    X,
    y,
    cv=5,
    scoring=make_scorer(accuracy_score)
)
```

In [ ]:
```
print("Cross-validated scores:", cross_val_scores)
print("Mean accuracy score:", cross_val_scores.mean())
print("Standard deviation of accuracy scores:", cross_val_scores.std())
```

```
Cross-validated scores: [0.98939498 0.98687724 0.98657104 0.97916985 0.98130627]
Mean accuracy score: 0.9846638749851 83
Standard deviation of accuracy scores: 0.003804667143368835
```

In [ ]:
```
plt.figure(figsize=(10,10))
sns.heatmap(confusion_mat,
            cmap= 'mako',
            annot=True,
            fmt='.0f',
            annot_kws={'size':12},
            linecolor='white',
            linewidths=.1,
            )
class_names = ['Allow','Deny','Drop','Reset-Both']
tick_position = [.5,1.5,2.5,3.5]
plt.xticks( ticks=tick_position,labels=class_names, rotation= 45)
plt.yticks(ticks=tick_position, labels = class_names, rotation = 45)
plt.title('Confusion Matrix for Action Classification')
plt.xlabel('Prediction')
plt.ylabel('Actual')
plt.show()
```



Extra

In [ ]:

```
from sklearn.model_selection import StratifiedKFold, cross_val_score

best_estimator = random_search.best_estimator_

cv = StratifiedKFold(n_splits=5)

cross_val_scores = cross_val_score(
    best_estimator,
    X,
    y,
    cv=cv,
    scoring=make_scorer(accuracy_score)
)


for i, (train_index, test_index) in enumerate(cv.split(X, y), start=1):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    best_estimator.fit(X_train, y_train)

    y_pred = best_estimator.predict(X_test)

    report = classification_report(y_test, y_pred, digits=2)
    print(f"Classification Report for Split {i}:\n", report)
```

```
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Classification Report for Split 1:
              precision    recall  f1-score   support

       allow       1.00      0.99      1.00      7528
        deny       0.98      0.98      0.98      2997
        drop       0.97      1.00      0.98      2571
  reset-both       0.00      0.00      0.00        11

    accuracy                           0.99     13107
   macro avg       0.74      0.74      0.74     13107
weighted avg       0.99      0.99      0.99     13107

c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Classification Report for Split 2:
              precision    recall  f1-score   support

       allow       1.00      0.99      0.99      7528
        deny       0.98      0.98      0.98      2998
        drop       0.96      1.00      0.98      2570
  reset-both       0.00      0.00      0.00        11

    accuracy                           0.99     13107
   macro avg       0.73      0.74      0.74     13107
weighted avg       0.99      0.99      0.99     13107

c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Classification Report for Split 3:
              precision    recall  f1-score   support

       allow       1.00      0.99      0.99      7528
        deny       0.98      0.97      0.98      2998
        drop       0.96      1.00      0.98      2570
  reset-both       0.00      0.00      0.00        10

    accuracy                           0.99     13106
   macro avg       0.73      0.74      0.74     13106
weighted avg       0.99      0.99      0.99     13106

c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Classification Report for Split 4:
               precision    recall  f1-score   support

       allow       1.00      0.98      0.99      7528
        deny       0.97      0.97      0.97      2997
        drop       0.94      1.00      0.97      2570
  reset-both       0.00      0.00      0.00        11

    accuracy                           0.98     13106
   macro avg       0.73      0.74      0.73     13106
weighted avg       0.98      0.98      0.98     13106

Classification Report for Split 5:
               precision    recall  f1-score   support

       allow       1.00      0.98      0.99      7528
        deny       0.98      0.97      0.97      2997
        drop       0.94      1.00      0.97      2570
  reset-both       0.00      0.00      0.00        11

    accuracy                           0.98     13106
   macro avg       0.73      0.74      0.73     13106
weighted avg       0.98      0.98      0.98     13106
```

```
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]:

```
cross_val_scores = np.array(cross_val_scores)
print("Cross-validation accuracy scores:", cross_val_scores)
print("Mean accuracy score:", cross_val_scores.mean())
```

```
Cross-validation accuracy scores: [0.98939498 0.98687724 0.98657104 0.97916985 0.98130627]
Mean accuracy score: 0.984663874985183
```

In [ ]:

```
from sklearn.model_selection import KFold

def classification_report_with_accuracy_score(y_true, y_pred):

    print (classification_report(y_true, y_pred)) # print classification report
    return accuracy_score(y_true, y_pred) # return accuracy score

cv = KFold(n_splits=4, shuffle=True, random_state=i)

nested_score = cross_val_score(best_estimator, X, y, cv=cv, \
               scoring=make_scorer(classification_report_with_accuracy_score))
print(nested_score)
```

```
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

       allow       1.00      0.99      0.99      9425
        deny       0.98      0.97      0.98      3726
        drop       0.95      1.00      0.98      3218
  reset-both       0.00      0.00      0.00        14

    accuracy                           0.99     16383
   macro avg       0.73      0.74      0.74     16383
weighted avg       0.98      0.99      0.99     16383
```

```
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

       allow       1.00      0.98      0.99      9397
        deny       0.98      0.97      0.98      3760
        drop       0.95      1.00      0.97      3214
  reset-both       0.00      0.00      0.00        12

    accuracy                           0.98     16383
   macro avg       0.73      0.74      0.74     16383
weighted avg       0.98      0.98      0.98     16383

c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Joey\anaconda3\envs\ML\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

       allow       1.00      0.98      0.99      9392
        deny       0.98      0.98      0.98      3776
        drop       0.95      1.00      0.98      3203
  reset-both       0.00      0.00      0.00        12

    accuracy                           0.99     16383
   macro avg       0.73      0.74      0.74     16383
weighted avg       0.98      0.99      0.98     16383

              precision    recall  f1-score   support

       allow       1.00      0.98      0.99      9426
        deny       0.98      0.98      0.98      3725
        drop       0.95      1.00      0.98      3216
  reset-both       0.00      0.00      0.00        16

    accuracy                           0.98     16383
   macro avg       0.73      0.74      0.74     16383
weighted avg       0.98      0.98      0.98     16383

[0.98541171 0.98376366 0.98504547 0.98412989]
```