

```
1  ###
2  # Cell 0: This Cell will serve to load any lybraries I will need throughout my
   project. This Helps me keep everything neat.
3
4  # Basic data handling and computation
5  import pandas as pd
6  import numpy as np
7
8  # Data visualization
9  import matplotlib.pyplot as plt
10 import seaborn as sns
11
12 # Preprocessing
13 from sklearn.model_selection import train_test_split
14 from sklearn.preprocessing import StandardScaler, LabelEncoder
15 from sklearn.metrics import classification_report, confusion_matrix,
   accuracy_score
16
17 # PyTorch for model building and training
18 import torch
19 import torch.nn as nn
20 import torch.nn.functional as F
21 from torch.utils.data import DataLoader, TensorDataset
22 import torch.optim as optim
23
24 # Additional tools
25 import os # For directory and file operations
26 import sys # For system-specific parameters and functions
27
28 ###
29 #Cell 1: Import new cleaned CSV
30
31 # Replace the file path with your specific file location
32 file_path = r'C:/Users/gsmi/OneDrive/Desktop/CS691 Project Codename
   Prayer/UDPLag.csv'
33 df = pd.read_csv(file_path)
34
35 # Display the first few rows to ensure it's loaded correctly
36 print(df.head())
37 print(df.info())
38
39 ###
40 print(df.shape)
41 ###
42 # Cell 2: Check if CUDA is available and set the device accordingly
```

```

43 # Specify the GPU device
44 if torch.cuda.is_available():
45     print("Available CUDA devices:")
46     for i in range(torch.cuda.device_count()):
47         print(f'Device {i}: {torch.cuda.get_device_name(i)} with {torch.cuda.
get_device_properties(i).total_memory / 1e9} GB')
48 else:
49     print("No CUDA devices available.")
50
51 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
52 print(f'Using {device} device')
53
54 #%%
55 # Cell 3: Focus on Benign and UDPLag Data
56
57 # Filtering the DataFrame to include only Benign and UDPLag data
58 benign_udplag_df = df[df['Label'].isin(['BENIGN', 'UDPLag'])]
59
60 # Display the first few rows to check the filtered data
61 print(benign_udplag_df.head())
62 print(f'Counts for Benign and UDPLag:\n{benign_udplag_df['Label'].
value_counts()}\n')
63
64 # Plot the distribution of the Benign and UDPLag classes
65 plt.figure(figsize=(8, 6))
66 sns.barplot(x=benign_udplag_df['Label'].value_counts().index,
67             y=benign_udplag_df['Label'].value_counts().values,
68             palette='pastel')
69 plt.title('Distribution of Benign and UDPLag Traffic Types')
70 plt.xlabel('Traffic Type')
71 plt.ylabel('Number of Instances')
72 plt.show()
73
74 # Select two continuous variables to compare
75 x_var = 'Flow Duration'
76 y_var = 'Total Fwd Packets'
77
78 # Create the scatter plot
79 plt.figure(figsize=(10, 6))
80 sns.scatterplot(data=benign_udplag_df, x=x_var, y=y_var, hue='Label', style='
Label', alpha=0.7)
81 plt.title('Scatter Plot for Benign vs UDPLag')
82 plt.xlabel('Flow Duration')
83 plt.ylabel('Total Forward Packets')
84 plt.legend(title='Label')

```

```

85 plt.show()
86
87 #%%
88 # Cell 5: Filter the data for 'Benign' and 'UDPLag'
89 filtered_df = benign_udplag_df.copy()
90
91 # Select a few features for the pair plot
92 selected_features = [
93     'Flow Duration',
94     'Total Fwd Packets',
95     'Total Length of Fwd Packets',
96     'Fwd Packet Length Max',
97     'Flow IAT Mean',
98     'Fwd IAT Total',
99     'Down/Up Ratio',
100    'Average Packet Size'
101 ]
102
103 # Sample the data to make it more manageable for plotting
104 sampled_df = filtered_df.sample(frac=0.01, random_state=42) # Adjust the
fraction as needed
105
106 # Create a pair plot
107 sns.pairplot(sampled_df[selected_features + ['Label']], hue='Label', plot_kws
    ={'alpha': 0.5})
108
109 plt.show()
110 #%%
111 # Now using benign_udplag_df to ensure consistency.
112 print("Missing values per column:")
113 print(benign_udplag_df.isnull().sum())
114
115 #%%
116 # Cell 7: List of columns to keep
117 columns_to_keep = [
118     'Source Port', 'Destination Port', 'Protocol', 'Flow Duration',
119     'Total Fwd Packets', 'Total Backward Packets', 'Total Length of Fwd
    Packets',
120     'Total Length of Bwd Packets', 'Fwd Packet Length Max', 'Fwd Packet
    Length Min',
121     'Fwd Packet Length Mean', 'Fwd Packet Length Std', 'Bwd Packet
    Length Max',
122     'Bwd Packet Length Min', 'Bwd Packet Length Mean', 'Bwd Packet
    Length Std',
123     'Flow Bytes/s', 'Flow Packets/s', 'Flow IAT Mean', 'Flow IAT Std',

```

```

124 ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean',
125 ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total',
126 ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min',
127 'FIN Flag Count', ' SYN Flag Count', ' RST Flag Count', ' PSH Flag
    Count',
128 ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count', ' ECE Flag
    Count',
129 ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size',
130 ' Avg Bwd Segment Size', ' Label'
131 ]
132 df_cleaned = filtered_df[columns_to_keep].copy()
133 print("DataFrame shape after removing erroneous entries:", df_cleaned.
    shape)
134
135 ###
136 # Cell 8: Ensure no negative values in 'Flow Duration' and ' Total Fwd Packets'
137 df_cleaned = df_cleaned[(df_cleaned[' Flow Duration'] >= 0) & (df_cleaned['
    Total Fwd Packets'] >= 0)]
138 print("DataFrame shape after removing erroneous entries:", df_cleaned.
    shape)
139
140 ###
141 # Cell 9: Remove duplicates, handle infinities, and prepare for advanced
imputation
142 df_cleaned = df_cleaned.drop_duplicates()
143 df_cleaned.replace([np.inf, -np.inf], np.nan, inplace=True)
144
145 # Add indicators for missing values for columns that will be imputed
146 for col in df_cleaned.columns:
147     if df_cleaned[col].isnull().any():
148         df_cleaned[col + ' _missing'] = df_cleaned[col].isnull().astype(int)
149
150 print("Preparation complete. Ready for advanced imputation.")
151
152 ###
153 # Enable experimental features to use IterativeImputer
154 from sklearn.experimental import enable_iterative_imputer
155 from sklearn.impute import IterativeImputer
156 from sklearn.neighbors import KNeighborsRegressor
157
158 # Define the imputer
159 iterative_imputer = IterativeImputer(estimator=KNeighborsRegressor(
    n_neighbors=5), random_state=42, max_iter=10)
160
161 # Columns selected for imputation

```

```
162 numeric_cols = df_cleaned.select_dtypes(include=['float64', 'int64']).columns
163 df_cleaned[numeric_cols] = iterative_imputer.fit_transform(df_cleaned[
    numeric_cols])
164
165 # Adding indicators for missing values for columns that will be imputed
166 for col in numeric_cols:
167     if df_cleaned[col].isnull().any():
168         df_cleaned[col + '_missing'] = df_cleaned[col].isnull().astype(int)
169
170 print("Missing values imputed using advanced techniques. Missing
    indicators added.")
171
172 ###
173 # Cell 11: Data Preparation for Model Training
174
175 # Adjust column names to ensure consistency
176 df_cleaned.columns = df_cleaned.columns.str.strip()
177
178 # Scale the features
179 scaler = StandardScaler()
180 features = df_cleaned.drop('Label', axis=1)
181 labels = df_cleaned['Label']
182
183 features_scaled = scaler.fit_transform(features)
184
185 # Encode the labels
186 encoder = LabelEncoder()
187 labels_encoded = encoder.fit_transform(labels)
188
189 # Split the dataset into training and testing sets
190 X_train, X_test, y_train, y_test = train_test_split(features_scaled, labels_encoded
    , test_size=0.2, random_state=42)
191
192 # Convert training data to tensors
193 X_train_tensor = torch.tensor(X_train, dtype=torch.float).to(device)
194 y_train_tensor = torch.tensor(y_train, dtype=torch.long).to(device)
195
196 # Convert test data to tensors
197 X_test_tensor = torch.tensor(X_test, dtype=torch.float).to(device)
198 y_test_tensor = torch.tensor(y_test, dtype=torch.long).to(device)
199
200 print("Features scaled, labels encoded, and data split into training and test
    sets. Tensors are ready for model training and evaluation.")
201
202 # Verify Label Encoding
```

```

203 encoder = LabelEncoder()
204 labels_encoded = encoder.fit_transform(df_cleaned['Label'])
205 # Print the mapping of labels to integers
206 label_mapping = dict(zip(encoder.classes_, encoder.transform(encoder.classes_
)))
207 print("Label Encoding Mapping:", label_mapping)
208
209 ###
210 # Cell 12: Define the Neural Network Model for Binary Classification
211 class BasicNN(nn.Module):
212     def __init__(self, input_size, output_size):
213         super(BasicNN, self).__init__()
214         self.layer1 = nn.Linear(input_size, 64)
215         self.relu = nn.ReLU()
216         self.layer2 = nn.Linear(64, output_size)
217         self.sigmoid = nn.Sigmoid() # Only use if output_size == 1 for binary
classification
218         self.initialize_weights()
219
220     def forward(self, x):
221         x = self.relu(self.layer1(x))
222         x = self.layer2(x)
223         if self.layer2.out_features == 1: # Assuming binary classification
224             x = self.sigmoid(x)
225         return x
226
227     def initialize_weights(self):
228         for m in self.modules():
229             if isinstance(m, nn.Linear):
230                 nn.init.kaiming_normal_(m.weight, mode='fan_out')
231                 if m.bias is not None:
232                     nn.init.constant_(m.bias, 0)
233
234 ###
235 # Cell 13: Train the Neural Network
236 from torch.optim import Adam
237 from torch.utils.data import DataLoader, TensorDataset
238
239 # Define hyperparameters
240 learning_rate = 0.001
241 num_epochs = 50
242 batch_size = 64
243
244 # Prepare DataLoader for batch processing
245 train_data = TensorDataset(X_train_tensor, y_train_tensor)

```

```

246 train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
247
248 # Initialize the model
249 model = BasicNN(input_size=X_train_tensor.shape[1], output_size=1)
250 model.to(device)
251
252 # Loss and optimizer
253 criterion = nn.BCEWithLogitsLoss() # Suitable for binary classification with
logits
254 optimizer = Adam(model.parameters(), lr=learning_rate)
255
256 # Training loop
257 model.train()
258 for epoch in range(num_epochs):
259     for inputs, labels in train_loader:
260         inputs, labels = inputs.to(device), labels.to(device)
261
262         # Forward pass
263         outputs = model(inputs)
264         loss = criterion(outputs, labels.unsqueeze(1).float())
265
266         # Backward and optimize
267         optimizer.zero_grad()
268         loss.backward()
269         optimizer.step()
270
271     if (epoch+1) % 5 == 0:
272         print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
273
274 ###
275 # Cell 14: Evaluate the Neural Network Model with Correct Labels
276 from sklearn.metrics import classification_report, confusion_matrix
277 import numpy as np
278
279 # Set the model to evaluation mode
280 model.eval()
281
282 # Prepare the DataLoader for the test data
283 test_data = TensorDataset(X_test_tensor, y_test_tensor)
284 test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
285
286 # Initialize lists to store true labels and predictions
287 predictions = []
288 true_labels = []
289

```

```
290 # No need to track gradients for evaluation
291 with torch.no_grad():
292     for inputs, labels in test_loader:
293         inputs = inputs.to(device)
294         labels = labels.to(device)
295
296         # Forward pass to get outputs
297         outputs = model(inputs)
298
299         # Since outputs are logits, apply sigmoid to calculate probabilities
300         probs = torch.sigmoid(outputs)
301
302         # Convert probabilities to predicted classes
303         preds = (probs > 0.5).int()
304
305         # Store predictions and actual labels as numpy arrays
306         predictions.append(preds.cpu().numpy())
307         true_labels.append(labels.cpu().numpy())
308
309     # Concatenate all predictions and true labels from list of arrays
310     predictions = np.concatenate(predictions).flatten()
311     true_labels = np.concatenate(true_labels).flatten()
312
313     # Map numeric labels back to original labels using the encoder
314     predicted_labels = encoder.inverse_transform(predictions)
315     true_labels = encoder.inverse_transform(true_labels)
316
317     # Generate classification report and confusion matrix with actual label names
318     print("Classification Report:")
319     print(classification_report(true_labels, predicted_labels, target_names=encoder.
320                                classes_))
321
322     print("Confusion Matrix:")
323     cm = confusion_matrix(true_labels, predicted_labels)
324     print(cm)
325
326     # Optionally, display the confusion matrix using Matplotlib for better
327     visualization
328     plt.figure(figsize=(8, 6))
329     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=encoder.
330                classes_, yticklabels=encoder.classes_)
331     plt.xlabel('Predicted Labels')
332     plt.ylabel('True Labels')
333     plt.title('Confusion Matrix')
334     plt.show()
```



```
332
333 ###
334 # Cell 15: Make a Copy of the Cleaned Data for Future Use
335 df2 = df_cleaned.copy()
336 print("A copy of the cleaned data has been made and stored in df2.")
337
338 ###
339 #Cell 16: Prepare Data for Random Forest Model
340
341 from sklearn.model_selection import train_test_split
342 from sklearn.preprocessing import LabelEncoder
343
344 # Ensure all column names have no leading or trailing spaces
345 df2.columns = df2.columns.str.strip()
346
347 # Separating the features and the target variable
348 X = df2.drop('Label', axis=1)
349 y = df2['Label'].values
350
351 # Encoding the labels
352 encoder = LabelEncoder()
353 y_encoded = encoder.fit_transform(y)
354
355 # Splitting the dataset into training and testing sets
356 X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2,
357 random_state=42)
358
359 print("Data prepared for Random Forest model.")
360 ###
361 #Cell 17: Train Random Forest model
362
363 from sklearn.ensemble import RandomForestClassifier
364
365 # Create and train the Random Forest classifier
366 rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
367 rf_classifier.fit(X_train, y_train)
368
369 print("Random Forest model trained.")
370
371 ###
372 #Cell 18: Evaluate Random Forest Model
373 from sklearn.metrics import classification_report, confusion_matrix,
374 accuracy_score
```

```

375 # Making predictions on the test set
376 y_pred = rf_classifier.predict(X_test)
377
378 # Evaluating the model
379 print("Classification Report:")
380 print(classification_report(y_test, y_pred, target_names=encoder.classes_))
381
382 print("Confusion Matrix:")
383 print(confusion_matrix(y_test, y_pred))
384
385 print("Accuracy Score:")
386 print(accuracy_score(y_test, y_pred))
387
388 # Optionally, display the confusion matrix using Matplotlib for better
visualization
389 plt.figure(figsize=(8, 6))
390 sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='
Blues',
391             xticklabels=encoder.classes_, yticklabels=encoder.classes_)
392 plt.xlabel('Predicted Labels')
393 plt.ylabel('True Labels')
394 plt.title('Confusion Matrix')
395 plt.show()
396
397 ###
398 # Cell 19: Hypertunning RF
399 from sklearn.model_selection import GridSearchCV
400
401 # Setting up the parameter grid
402 param_grid = {
403     'n_estimators': [100, 200, 300], # Number of trees in the forest
404     'max_features': ['auto', 'sqrt', 'log2'], # Number of features to consider at
every split
405     'max_depth': [None, 10, 20, 30], # Maximum number of levels in tree
406     'min_samples_split': [2, 5, 10], # Minimum number of samples required to
split a node
407     'min_samples_leaf': [1, 2, 4] # Minimum number of samples required at each
leaf node
408 }
409
410 # Create the base model to tune
411 rf = RandomForestClassifier(random_state=42)
412
413 # Instantiate the grid search model
414 grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3,

```

```
414 n_jobs=-1, verbose=2, scoring='accuracy')
415
416 # Fit the grid search to the data
417 grid_search.fit(X_train, y_train)
418
419 # Best parameters and best score
420 print("Best parameters found: ", grid_search.best_params_)
421 print("Best accuracy achieved: ", grid_search.best_score_)
422
423 # Rebuild the model with the best parameters
424 best_rf = grid_search.best_estimator_
425
426 # Evaluate on the test set
427 y_pred_optimized = best_rf.predict(X_test)
428 y_pred_labels_optimized = encoder.inverse_transform(y_pred_optimized) #
Decode the predictions
429 y_test_labels = encoder.inverse_transform(y_test) # Decode y_test to use string
labels for evaluation
430
431 print("Optimized Classification Report:")
432 print(classification_report(y_test_labels, y_pred_labels_optimized, target_names
    =encoder.classes_))
433
434 print("Optimized Confusion Matrix:")
435 cm = confusion_matrix(y_test_labels, y_pred_labels_optimized)
436 print(cm)
437
438 # Display the confusion matrix using Matplotlib for better visualization
439 plt.figure(figsize=(8, 6))
440 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=encoder.
    classes_, yticklabels=encoder.classes_)
441 plt.xlabel('Predicted Labels')
442 plt.ylabel('True Labels')
443 plt.title('Optimized Random Forest Confusion Matrix')
444 plt.show()
445
446 ###
447 # Cell 20: Train K-Nearest Neighbors Model
448 from sklearn.neighbors import KNeighborsClassifier
449 from sklearn.metrics import classification_report, confusion_matrix
450
451 # Create and train the KNN classifier
452 knn = KNeighborsClassifier()
453 knn.fit(X_train, y_train)
454
```

```

455 # Making predictions on the test set
456 y_pred_knn = knn.predict(X_test)
457
458 # Decode the predictions and actual labels for reporting
459 y_pred_labels_knn = encoder.inverse_transform(y_pred_knn)
460 y_test_labels = encoder.inverse_transform(y_test)
461
462 print("KNN Classification Report:")
463 print(classification_report(y_test_labels, y_pred_labels_knn, target_names=
encoder.classes_))
464 print("KNN Confusion Matrix:")
465 cm_knn = confusion_matrix(y_test_labels, y_pred_labels_knn)
466 print(cm_knn)
467
468 # Optionally, display the confusion matrix using Matplotlib for better
visualization
469 plt.figure(figsize=(8, 6))
470 sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Blues', xticklabels=encoder.
classes_, yticklabels=encoder.classes_)
471 plt.xlabel('Predicted Labels')
472 plt.ylabel('True Labels')
473 plt.title('KNN Confusion Matrix')
474 plt.show()
475
476 ###
477 #Cell 21: Hyperparameter Tuning for K-Nearest Neighbors
478 from sklearn.model_selection import GridSearchCV
479
480 # Setting up the parameter grid
481 knn_param_grid = {
482     'n_neighbors': [3, 5, 7, 9], # Different values for the number of neighbors
483     'weights': ['uniform', 'distance'], # Weight function used in prediction
484     'metric': ['euclidean', 'manhattan', 'minkowski'] # Distance metric for tree
search
485 }
486
487 # Create the base model to tune
488 knn_base = KNeighborsClassifier()
489
490 # Instantiate the grid search model
491 knn_grid_search = GridSearchCV(estimator=knn_base, param_grid=
knn_param_grid, cv=3, n_jobs=-1, verbose=2, scoring='accuracy')
492
493 # Fit the grid search to the data
494 knn_grid_search.fit(X_train, y_train)

```

```

495
496 # Best parameters and best score
497 print("Best parameters found: ", knn_grid_search.best_params_)
498 print("Best accuracy achieved: ", knn_grid_search.best_score_)
499
500 ###
501 # Cell 22: Re-evaluate KNN with Optimized Parameters
502 from sklearn.metrics import classification_report, confusion_matrix
503
504 # Rebuild the model with the best parameters from hyperparameter tuning
505 best_knn = knn_grid_search.best_estimator_
506
507 # Evaluate on the test set
508 y_pred_optimized_knn = best_knn.predict(X_test)
509
510 # Decode the optimized predictions and actual labels for reporting
511 y_pred_labels_optimized_knn = encoder.inverse_transform(
    y_pred_optimized_knn)
512 y_test_labels = encoder.inverse_transform(y_test)
513
514 print("Optimized KNN Classification Report:")
515 print(classification_report(y_test_labels, y_pred_labels_optimized_knn,
    target_names=encoder.classes_))
516 print("Optimized KNN Confusion Matrix:")
517 cm_optimized_knn = confusion_matrix(y_test_labels,
    y_pred_labels_optimized_knn)
518 print(cm_optimized_knn)
519
520 # Optionally, display the confusion matrix using Matplotlib for better
visualization
521 plt.figure(figsize=(8, 6))
522 sns.heatmap(cm_optimized_knn, annot=True, fmt='d', cmap='Blues', xticklabels
    =encoder.classes_, yticklabels=encoder.classes_)
523 plt.xlabel('Predicted Labels')
524 plt.ylabel('True Labels')
525 plt.title('Confusion Matrix')
526 plt.show()
527
528 ###
529 # Cell 23: Train and Evaluate SVM Model with a reduced dataset
530 from sklearn.svm import SVC
531 from sklearn.metrics import classification_report, confusion_matrix
532
533 # Reduce the training dataset size
534 X_train_sub, _, y_train_sub, _ = train_test_split(

```

```

535 X_train, y_train, test_size=0.9, random_state=42) # Use only 10% of data for
    initial training
536
537 # Create and train the SVM classifier on a reduced dataset
538 svm_classifier = SVC(kernel='linear', random_state=42)
539 svm_classifier.fit(X_train_sub, y_train_sub)
540
541 # Making predictions on the full test set
542 y_pred_svm = svm_classifier.predict(X_test)
543
544 # Decode the predictions for reporting
545 y_pred_labels_svm = encoder.inverse_transform(y_pred_svm)
546
547 # Evaluating the model
548 print("SVM Classification Report:")
549 print(classification_report(y_test, y_pred_labels_svm, target_names=encoder.
    classes_))
550
551 print("SVM Confusion Matrix:")
552 cm_svm = confusion_matrix(y_test, y_pred_labels_svm)
553 print(cm_svm)
554
555 # Display the confusion matrix using Matplotlib for better visualization
556 plt.figure(figsize=(8, 6))
557 sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues', xticklabels=encoder.
    classes_, yticklabels=encoder.classes_)
558 plt.xlabel('Predicted Labels')
559 plt.ylabel('True Labels')
560 plt.title('SVM Confusion Matrix')
561 plt.show()
562 ###
563 # Cell 24: Hyperparameter Tuning for Support Vector Machine
564 from sklearn.model_selection import GridSearchCV
565
566 # Setting up the parameter grid
567 svm_param_grid = {
568     'C': [0.1, 1, 10, 100], # Regularization parameter
569     'gamma': ['scale', 'auto', 0.1, 1, 10, 100], # Kernel coefficient for 'rbf', 'poly'
    and 'sigmoid'
570     'kernel': ['rbf', 'poly', 'sigmoid'] # Specifies the kernel type to be used in the
    algorithm
571 }
572
573 # Create the base model to tune
574 svm_base = SVC(random_state=42)

```

```
575
576 # Instantiate the grid search model
577 svm_grid_search = GridSearchCV(estimator=svm_base, param_grid=
    svm_param_grid, cv=3, n_jobs=-1, verbose=2,
578                                 scoring='accuracy')
579
580 # Fit the grid search to the data
581 svm_grid_search.fit(X_train, y_train)
582
583 # Best parameters and best score
584 print("Best parameters found: ", svm_grid_search.best_params_)
585 print("Best accuracy achieved: ", svm_grid_search.best_score_)
586
587 # Cell 25: Re-evaluate SVM with Optimized Parameters
588 best_svm = svm_grid_search.best_estimator_
589
590 # Evaluate on the test set
591 y_pred_optimized_svm = best_svm.predict(X_test)
592 y_pred_labels_optimized_svm = encoder.inverse_transform(
    y_pred_optimized_svm)
593
594 print("Optimized SVM Classification Report:")
595 print(classification_report(y_test, y_pred_labels_optimized_svm, target_names=
    encoder.classes_))
596 print("Optimized SVM Confusion Matrix:")
597 cm_optimized_svm = confusion_matrix(y_test, y_pred_labels_optimized_svm)
598 print(cm_optimized_svm)
599
600 # Display the confusion matrix using Matplotlib for better visualization
601 plt.figure(figsize=(8, 6))
602 sns.heatmap(cm_optimized_svm, annot=True, fmt='d', cmap='Blues',
    xticklabels=encoder.classes_,
603             yticklabels=encoder.classes_)
604 plt.xlabel('Predicted Labels')
605 plt.ylabel('True Labels')
606 plt.title('Optimized SVM Confusion Matrix')
607 plt.show()
608
609 %%
610 # Cell 25: Re-evaluate SVM with Optimized Parameters
611
612 # Rebuild the model with the best parameters
613 best_svm = svm_grid_search.best_estimator_
614
615 # Evaluate on the test set
```

```
616 y_pred_optimized_svm = best_svm.predict(X_test)
617 y_pred_labels_optimized_svm = encoder.inverse_transform(
    y_pred_optimized_svm) # Decode labels
618
619 # Evaluating the model
620 print("Optimized SVM Classification Report:")
621 print(classification_report(y_test, y_pred_labels_optimized_svm, target_names=
    encoder.classes_))
622
623 print("Optimized SVM Confusion Matrix:")
624 cm_optimized_svm = confusion_matrix(y_test, y_pred_labels_optimized_svm)
625 print(cm_optimized_svm)
626
627 # Display the confusion matrix using Matplotlib for better visualization
628 plt.figure(figsize=(8, 6))
629 sns.heatmap(cm_optimized_svm, annot=True, fmt='d', cmap='Blues',
    xticklabels=encoder.classes_, yticklabels=encoder.classes_)
630 plt.xlabel('Predicted Labels')
631 plt.ylabel('True Labels')
632 plt.title('Optimized SVM Confusion Matrix')
633 plt.show()
634
```