In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:
```
data = pd.read_csv('SIMARGL2021/dataset-part1.csv')
```

In [5]:
```
# Load data
data.columns
```

Out[5]:
```
Index(['BIFLOW_DIRECTION', 'DIRECTION', 'DST_TO_SRC_SECOND_BYTES',
       'FIREWALL_EVENT', 'FIRST_SWITCHED', 'FLOW_ACTIVE_TIMEOUT',
       'FLOW_DURATION_MICROSECONDS', 'FLOW_DURATION_MILLISECONDS',
       'FLOW_END_MILLISECONDS', 'FLOW_END_SEC', 'FLOW_ID',
       'FLOW_INACTIVE_TIMEOUT', 'FLOW_START_MILLISECONDS', 'FLOW_START_SEC',
       'FRAME_LENGTH', 'IN_BYTES', 'IN_PKTS', 'IPV4_DST_ADDR', 'IPV4_SRC_ADDR',
       'L4_DST_PORT', 'L4_SRC_PORT', 'LAST_SWITCHED', 'MAX_IP_PKT_LEN',
       'MIN_IP_PKT_LEN', 'OOORDER_IN_PKTS', 'OOORDER_OUT_PKTS', 'OUT_BYTES',
       'OUT_PKTS', 'PROTOCOL', 'PROTOCOL_MAP', 'RETRANSMITTED_IN_BYTES',
       'RETRANSMITTED_IN_PKTS', 'RETRANSMITTED_OUT_BYTES',
       'RETRANSMITTED_OUT_PKTS', 'SRC_TO_DST_SECOND_BYTES', 'TCP_FLAGS',
       'TCP_WIN_MAX_IN', 'TCP_WIN_MAX_OUT', 'TCP_WIN_MIN_IN',
       'TCP_WIN_MIN_OUT', 'TCP_WIN_MSS_IN', 'TCP_WIN_MSS_OUT',
       'TCP_WIN_SCALE_IN', 'TCP_WIN_SCALE_OUT', 'SRC_TOS', 'DST_TOS',
       'L7_PROTO_NAME', 'SAMPLING_INTERVAL', 'TOTAL_FLOWS_EXP', 'LABEL'],
      dtype='object')
```

In [6]:
```
null_counts = data.isnull().sum()
# Print the number of null values
print(f"{null_counts.sum()} null entries have been found in the dataset\n")
# Drop null values
data.dropna(inplace=True)          # or df_data = df_data.dropna()

# Find and handle duplicates
duplicate_count = data.duplicated().sum()
# Print the number of duplicate entries
print(f"{duplicate_count} duplicate entries have been found in the dataset\n")
# Remove duplicates
data.drop_duplicates(inplace=True)  # or df_data = df_data.drop_duplicates()
# Display relative message
print(f"All duplicates have been removed\n")

# Reset the indexes
data.reset_index(drop=True, inplace=True)

# Inspect the dataset for categorical columns
print("Categorical columns:",data.select_dtypes(include=['object']).columns.tolist(),'\n')

# Print the first 5 lines
data.head()
```

```
0 null entries have been found in the dataset

0 duplicate entries have been found in the dataset

All duplicates have been removed

Categorical columns: ['DST_TO_SRC_SECOND_BYTES', 'IPV4_DST_ADDR', 'IPV4_SRC_ADDR', 'PROTOCOL_MAP', 'SRC_TO_DST_SECOND_BYTES', 'L7_PROTO_NAME', 'LABEL']
```

Out[6]:

	BIFLOW_DIRECTION	DIRECTION	DST_TO_SRC_SECOND_BYTES	FIREWALL_EVENT	FIRST_SWITCHED	FLOW_ACTIVE_TIMEOUT	FLOW_DURATION_MICROSECONDS	FLOW_DURATION_MILLISECONDS	FLOW_END_MILLIS
0	1	0	40	0	1616660040	120	339	0	16166
1	1	0	,	0	1616660040	120	0	0	16166
2	1	0	104	0	1616660040	120	44725	44	16166
3	1	0	,	0	1616660040	120	0	0	16166
4	1	0	40	0	1616660040	120	1114	1	16166

5 rows × 50 columns

In [7]:
```
drop_columns = [ # this list includes all spellings across CIC NIDS datasets
    'DST_TO_SRC_SECOND_BYTES', 'IPV4_DST_ADDR', 'IPV4_SRC_ADDR', 'PROTOCOL_MAP', 'SRC_TO_DST_SECOND_BYTES', 'L7_PROTO_NAME'
]
data.drop(columns=drop_columns, inplace=True, errors='ignore')
```

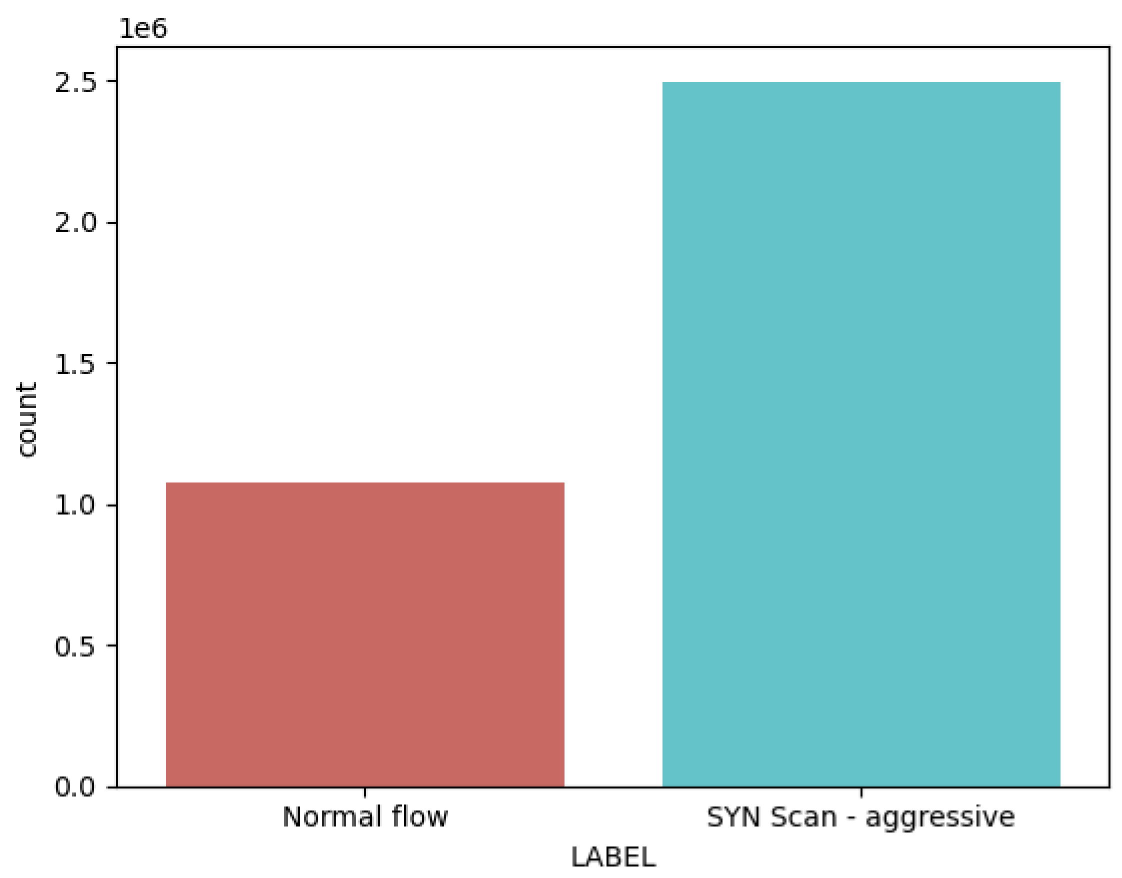
In [8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3570666 entries, 0 to 3570665
Data columns (total 44 columns):
 #   Column                     Dtype 
---  ------                     ----- 
 0   BIFLOW_DIRECTION           int64 
 1   DIRECTION                  int64 
 2   FIREWALL_EVENT             int64 
 3   FIRST_SWITCHED             int64 
 4   FLOW_ACTIVE_TIMEOUT        int64 
 5   FLOW_DURATION_MICROSECONDS  int64 
 6   FLOW_DURATION_MILLISECONDS  int64 
 7   FLOW_END_MILLISECONDS      int64 
 8   FLOW_END_SEC               int64 
 9   FLOW_ID                    int64 
 10  FLOW_INACTIVE_TIMEOUT      int64 
 11  FLOW_START_MILLISECONDS    int64 
 12  FLOW_START_SEC             int64 
 13  FRAME_LENGTH               int64 
 14  IN_BYTES                   int64 
 15  IN_PKTS                    int64 
 16  L4_DST_PORT                int64 
 17  L4_SRC_PORT                int64 
 18  LAST_SWITCHED              int64 
 19  MAX_IP_PKT_LEN             int64 
 20  MIN_IP_PKT_LEN             int64 
 21  OOORDER_IN_PKTS            int64 
 22  OOORDER_OUT_PKTS           int64 
 23  OUT_BYTES                  int64 
 24  OUT_PKTS                   int64 
 25  PROTOCOL                   int64 
 26  RETRANSMITTED_IN_BYTES     int64 
 27  RETRANSMITTED_IN_PKTS      int64 
 28  RETRANSMITTED_OUT_BYTES    int64 
 29  RETRANSMITTED_OUT_PKTS     int64 
 30  TCP_FLAGS                  int64 
 31  TCP_WIN_MAX_IN             int64 
 32  TCP_WIN_MAX_OUT            int64 
 33  TCP_WIN_MIN_IN             int64 
 34  TCP_WIN_MIN_OUT            int64 
 35  TCP_WIN_MSS_IN             int64 
 36  TCP_WIN_MSS_OUT            int64 
 37  TCP_WIN_SCALE_IN           int64 
 38  TCP_WIN_SCALE_OUT          int64 
 39  SRC_TOS                    int64 
 40  DST_TOS                    int64 
 41  SAMPLING_INTERVAL          int64 
 42  TOTAL_FLOWS_EXP            int64 
 43  LABEL                      object
dtypes: int64(43), object(1)
memory usage: 1.2+ GB
```

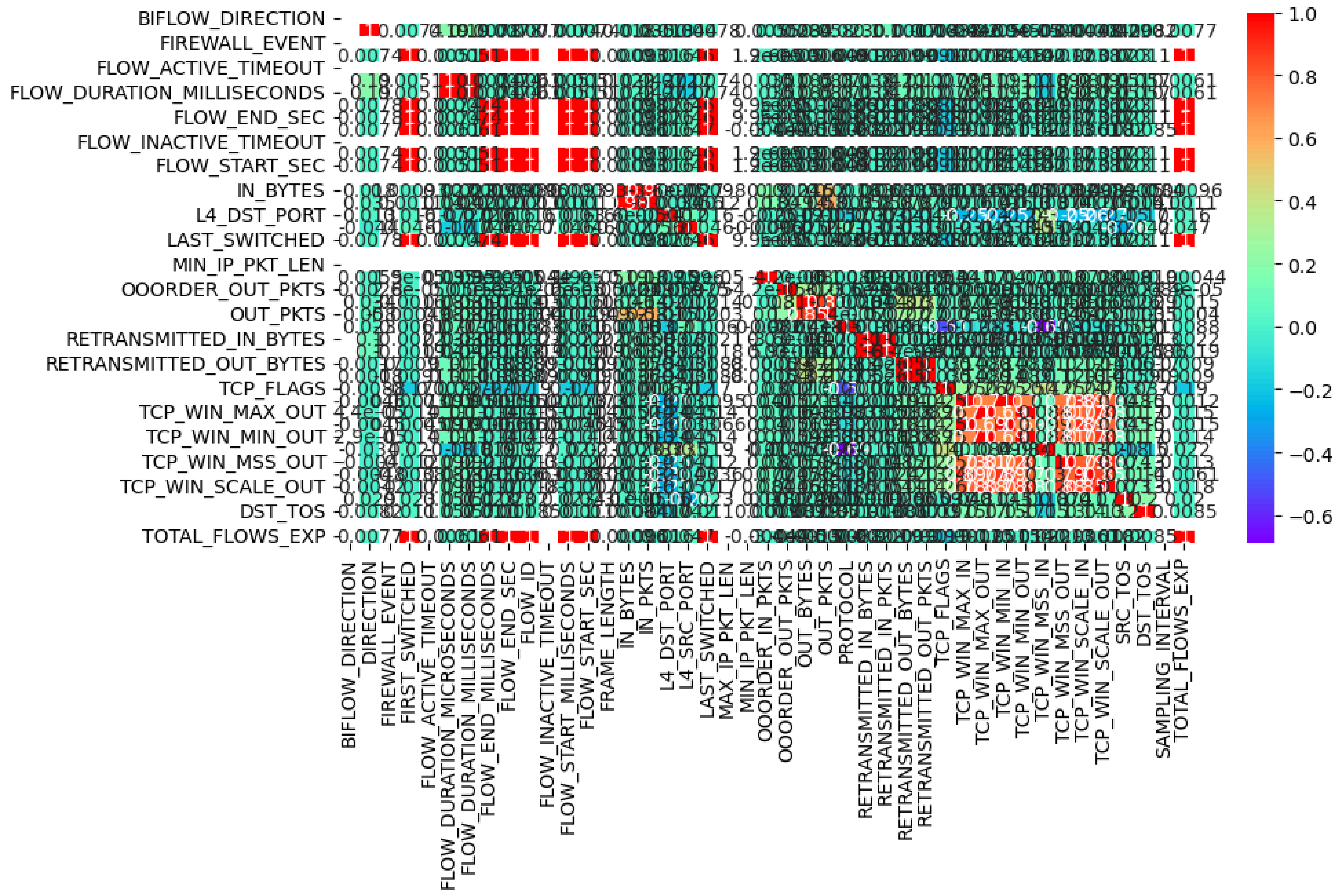
In [9]:
```
data['LABEL'].value_counts()
```

Out[9]:
```
SYN Scan - aggressive    2496814
Normal flow              1073852
Name: LABEL, dtype: int64
```

In [10]:
```
sns.countplot(x='LABEL',data=data, palette='hls')
plt.show()
#plt.savefig('count_plot') mal: the nodule malignancy, 0: benign, 1: malignant
```



In [11]:
```
plt.figure(figsize = (10,5))
sns.heatmap(data.corr(), annot = True, cmap="rainbow")
plt.show()
```



In [14]:
```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
data['LABEL']= label_encoder.fit_transform(data['LABEL'])

data['LABEL'].unique()
```

Out[14]:
```
array([1, 0])
```

In [15]:
```
X = data.drop(["LABEL"],axis =1)
y = data["LABEL"]
```

FS

In [16]:
```
from sklearn.feature_selection import SelectKBest, SelectPercentile, mutual_info_classif
```

In [17]:
```
selector = SelectPercentile(mutual_info_classif, percentile=25)
X_reduced = selector.fit_transform(X, y)
#X_reduced.shape
```

In [19]:
```
cols = selector.get_support(indices=True)
selected_columns = X.iloc[:,cols].columns.tolist()
selected_columns
```

Out[19]:
```
['FLOW_DURATION_MICROSECONDS',
 'FLOW_DURATION_MILLISECONDS',
 'IN_BYTES',
 'L4_DST_PORT',
 'L4_SRC_PORT',
 'OUT_BYTES',
 'PROTOCOL',
 'TCP_FLAGS',
 'TCP_WIN_MAX_IN',
 'TCP_WIN_MIN_IN',
 'TCP_WIN_MSS_IN']
```

In [20]:
```
len(selected_columns)
```

Out[20]:
```
11
```

In [21]:
```
df = data[['FLOW_DURATION_MICROSECONDS',
         'FLOW_DURATION_MILLISECONDS',
         'IN_BYTES',
         'L4_DST_PORT',
         'L4_SRC_PORT',
         'OUT_BYTES',
         'PROTOCOL',
         'TCP_FLAGS',
         'TCP_WIN_MAX_IN',
         'TCP_WIN_MIN_IN',
         'TCP_WIN_MSS_IN',
          'LABEL']]
```

In [22]:
```
df.to_csv('simarg_processed.csv')
```

In [3]:
```
df = pd.read_csv('simarg_processed.csv')
```

In [4]:
```
del df['Unnamed: 0']
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['FLOW_DURATION_MICROSECONDS', 'FLOW_DURATION_MILLISECONDS', 'IN_BYTES',
       'L4_DST_PORT', 'L4_SRC_PORT', 'OUT_BYTES', 'PROTOCOL', 'TCP_FLAGS',
       'TCP_WIN_MAX_IN', 'TCP_WIN_MIN_IN', 'TCP_WIN_MSS_IN', 'LABEL'],
      dtype='object')
```

In [6]:
```
X = df[['FLOW_DURATION_MICROSECONDS', 'FLOW_DURATION_MILLISECONDS', 'IN_BYTES',
       'L4_DST_PORT', 'L4_SRC_PORT', 'OUT_BYTES', 'PROTOCOL', 'TCP_FLAGS',
       'TCP_WIN_MAX_IN', 'TCP_WIN_MIN_IN', 'TCP_WIN_MSS_IN']]
y = df["LABEL"]
```

In [7]:
```
from sklearn.model_selection import train_test_split
```

In [8]:
```
from sklearn.metrics import accuracy_score # for calculating accuracy of model
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

In [9]:
```
ML_Model = []
accuracy = []
precision = []
recall = []

f1score = []


#function to call for storing the results
def storeResults(model, a,b,c,d):
    ML_Model.append(model)
    accuracy.append(round(a, 3))
    precision.append(round(b, 3))
    recall.append(round(c, 3))
    f1score.append(round(d, 3))
```

In [15]:
```
# importing lime
import lime
from lime import lime_tabular
import shap
```

DNN

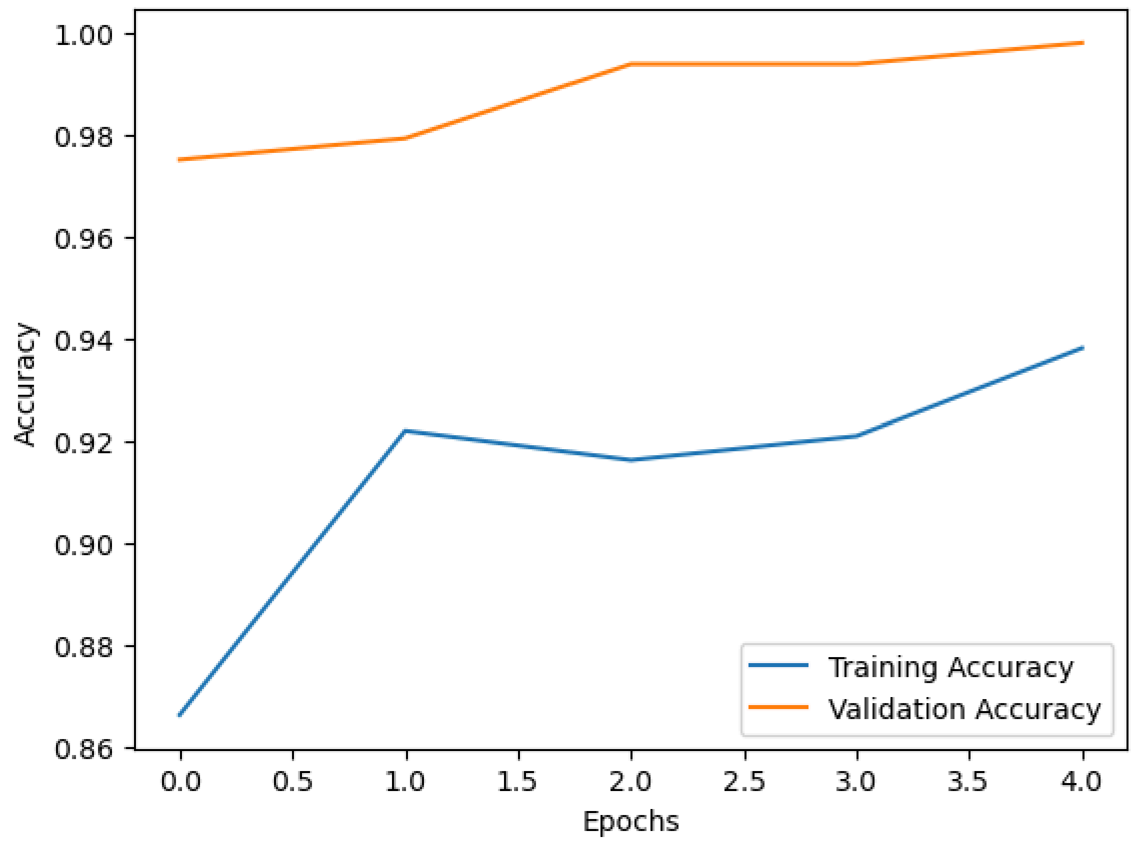
In [10]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

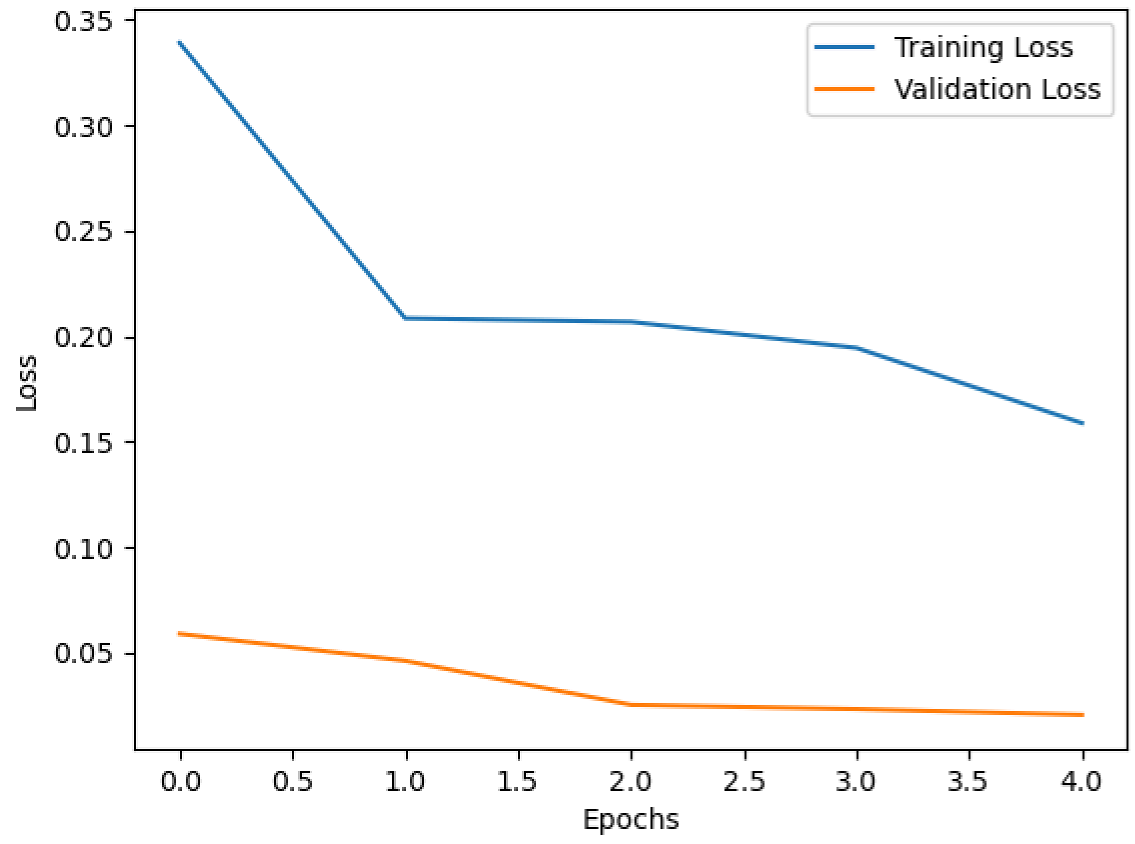
In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)
# Scale the features using StandardScaler
#scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)
```

In [13]:
```
# Build the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1024, activation='relu', input_dim=X_train.shape[1]),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=5, batch_size=4, validation_split=0.2)
```

```
Epoch 1/5
480/480 [==============================] - 5s 6ms/step - loss: 0.6289 - accuracy: 0.7318 - val_loss: 0.3588 - val_accuracy: 0.8583
Epoch 2/5
480/480 [==============================] - 2s 5ms/step - loss: 0.5105 - accuracy: 0.7729 - val_loss: 0.3445 - val_accuracy: 0.8792
Epoch 3/5
480/480 [==============================] - 2s 5ms/step - loss: 0.4602 - accuracy: 0.7995 - val_loss: 0.4112 - val_accuracy: 0.8104
Epoch 4/5
480/480 [==============================] - 2s 5ms/step - loss: 0.4237 - accuracy: 0.8245 - val_loss: 0.3230 - val_accuracy: 0.8813
Epoch 5/5
480/480 [==============================] - 2s 5ms/step - loss: 0.3814 - accuracy: 0.8448 - val_loss: 0.2907 - val_accuracy: 0.9688
```

In [13]:
```
import matplotlib.pyplot as plt

# Plotting akurasi
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting kerugian
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





In [14]:
```
# Evaluate the model on the testing set
predict_x=model.predict(X_test)
y_pred=np.argmax(predict_x,axis=1)

dl_acc = accuracy_score(y_pred, y_test)
dl_prec = precision_score(y_test,y_pred,average='weighted')
dl_rec = recall_score(y_test,y_pred,average='weighted')
dl_f1 = f1_score(y_test,y_pred,average='weighted')
```

```
50/50 [==============================] - 0s 2ms/step
```

In [15]:
```
storeResults('DNN',dl_acc,dl_prec,dl_rec,dl_f1)
```

In [16]:
```
lime_explainer = lime_tabular.LimeTabularExplainer(training_data=np.array(X_train), feature_names=X_train.columns,
            class_names=['0','1'], mode='classification')
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=model.predict, top_labels=6, num_features=19)
```

```
157/157 [==============================] - 0s 2ms/step
```

In [17]:
```
explanation.show_in_notebook()
```

ML

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 42)
```

AdaBoost

In [19]:
```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()

ada.fit(X_train, y_train)

y_pred    = ada.predict(X_test)

ada_acc = accuracy_score(y_pred, y_test)
ada_prec = precision_score(y_pred, y_test,average='weighted')
ada_rec = recall_score(y_pred, y_test,average='weighted')
ada_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [18]:
```
storeResults('AdaBoost',ada_acc,ada_prec,ada_rec,ada_f1)
```

In [20]:
```
lime_explainer = lime_tabular.LimeTabularExplainer(training_data=np.array(X_train), feature_names=X_train.columns,
            class_names=['0','1'], mode='classification')
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=ada.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

LightGBM

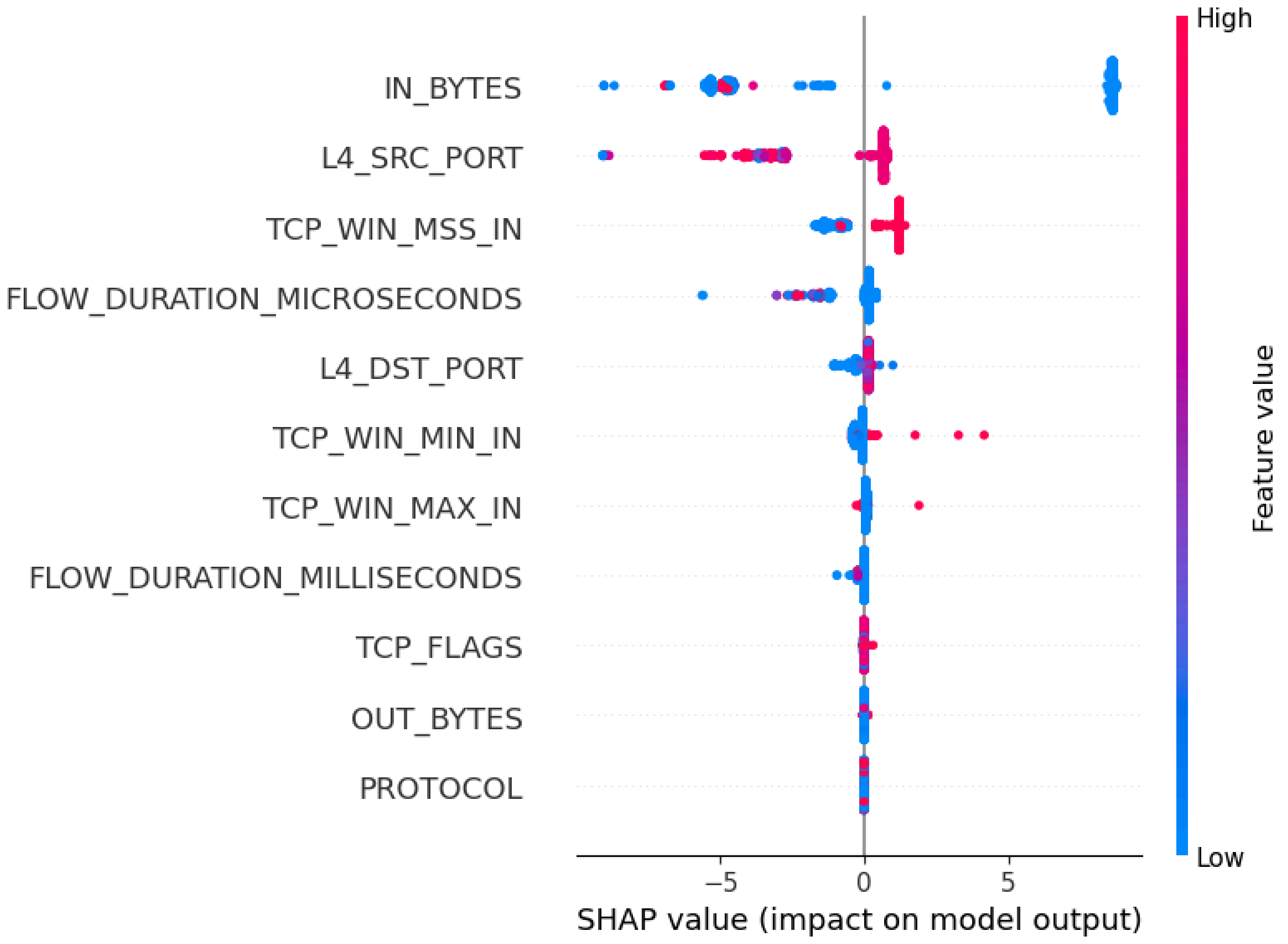
In [22]:
```
# build the lightgbm model
import lightgbm as lgb
clf = lgb.LGBMClassifier()
clf.fit(X_train, y_train)

y_pred    = clf.predict(X_test)

lgb_acc = accuracy_score(y_pred, y_test)
lgb_prec = precision_score(y_pred, y_test,average='weighted')
lgb_rec = recall_score(y_pred, y_test,average='weighted')
lgb_f1 = f1_score(y_pred, y_test,average='weighted')
```

```
[LightGBM] [Info] Number of positive: 1578, number of negative: 1622
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000625 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1698
[LightGBM] [Info] Number of data points in the train set: 3200, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.493125 -> initscore=-0.027502
[LightGBM] [Info] Start training from score -0.027502
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

In [21]:
```
storeResults('LightGBM',lgb_acc,lgb_prec,lgb_rec,lgb_f1)
```

In [23]:
```
lime_explainer = lime_tabular.LimeTabularExplainer(training_data=np.array(X_train), feature_names=X_train.columns,
            class_names=['0','1'], mode='classification')
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=clf.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

In [24]:
```
explainer = shap.Explainer(clf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



MLP

In [25]:
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(random_state=1, max_iter=300)

mlp.fit(X_train, y_train)

y_pred    = mlp.predict(X_test)

mlp_acc = accuracy_score(y_pred, y_test)
mlp_prec = precision_score(y_pred, y_test,average='weighted')
mlp_rec = recall_score(y_pred, y_test,average='weighted')
mlp_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [23]:
```
storeResults('MLP',mlp_acc,mlp_prec,mlp_rec,mlp_f1)
```

In [26]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=mlp.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

KNN

In [27]:
```
from sklearn.neighbors import KNeighborsClassifier
knn =  KNeighborsClassifier()
knn.fit(X_train, y_train)

y_pred    = knn.predict(X_test)

knn_acc = accuracy_score(y_pred, y_test)
knn_prec = precision_score(y_pred, y_test,average='weighted')
knn_rec = recall_score(y_pred, y_test,average='weighted')
knn_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [25]:
```
storeResults('KNN',knn_acc,knn_prec,knn_rec,knn_f1)
```

In [28]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=knn.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

Random Forest

In [29]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(criterion='entropy',max_features='log2',max_depth=20,n_estimators=600,min_samples_leaf=2)
rf.fit(X_train, y_train)

y_pred    = rf.predict(X_test)

rf_acc = accuracy_score(y_pred, y_test)
rf_prec = precision_score(y_pred, y_test,average='weighted')
rf_rec = recall_score(y_pred, y_test,average='weighted')
rf_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [27]:
```
storeResults('RandomForest',rf_acc,rf_prec,rf_rec,rf_f1)
```

In [30]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=rf.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

SVM

In [31]:
```
from sklearn import svm
svc = svm.SVC(decision_function_shape='ovo',probability=True)
svc.fit(X_train, y_train)

y_pred    = svc.predict(X_test)

svc_acc = accuracy_score(y_pred, y_test)
svc_prec = precision_score(y_pred, y_test,average='weighted')
svc_rec = recall_score(y_pred, y_test,average='weighted')
svc_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [29]:
```
storeResults('SVM',svc_acc,svc_prec,svc_rec,svc_f1)
```

In [32]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=svc.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

Extension

In [33]:
```
from sklearn.ensemble import VotingClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

brf = BaggingClassifier(RandomForestClassifier())

bdt = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), algorithm="SAMME", n_estimators=200
)

model = VotingClassifier(estimators= [('BoostDT', bdt),('BagRF', brf)], voting='soft')

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

ext_acc = accuracy_score(y_pred, y_test)
ext_prec = precision_score(y_pred, y_test,average='weighted')
ext_rec = recall_score(y_pred, y_test,average='weighted')
ext_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [31]:
```
storeResults('Extension',ext_acc,ext_prec,ext_rec,ext_f1)
```

In [34]:
```
explanation = lime_explainer.explain_instance(data_row=X_test.iloc[1], predict_fn=model.predict_proba, top_labels=6, num_features=19)
explanation.show_in_notebook()
```

Comparison

In [32]:
```
#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                       'Precision': precision,
                       'Recall'   : recall,
                       'F1_score' : f1score
                      })
```

In [33]:
```
result
```

Out[33]:

	ML Model	Accuracy	Precision	Recall	F1_score
0	DNN	0.487	0.237	0.487	0.319
1	AdaBoost	0.999	0.999	0.999	0.999
2	LightGBM	0.999	0.999	0.999	0.999
3	MLP	1.000	1.000	1.000	1.000
4	KNN	0.998	0.998	0.998	0.998
5	RandomForest	1.000	1.000	1.000	1.000
6	SVM	0.628	0.921	0.628	0.700
7	Extension	1.000	1.000	1.000	1.000

Modelling

In [34]:
```
import joblib
filename = 'models/model_simarg.sav'
joblib.dump(model, filename)
```

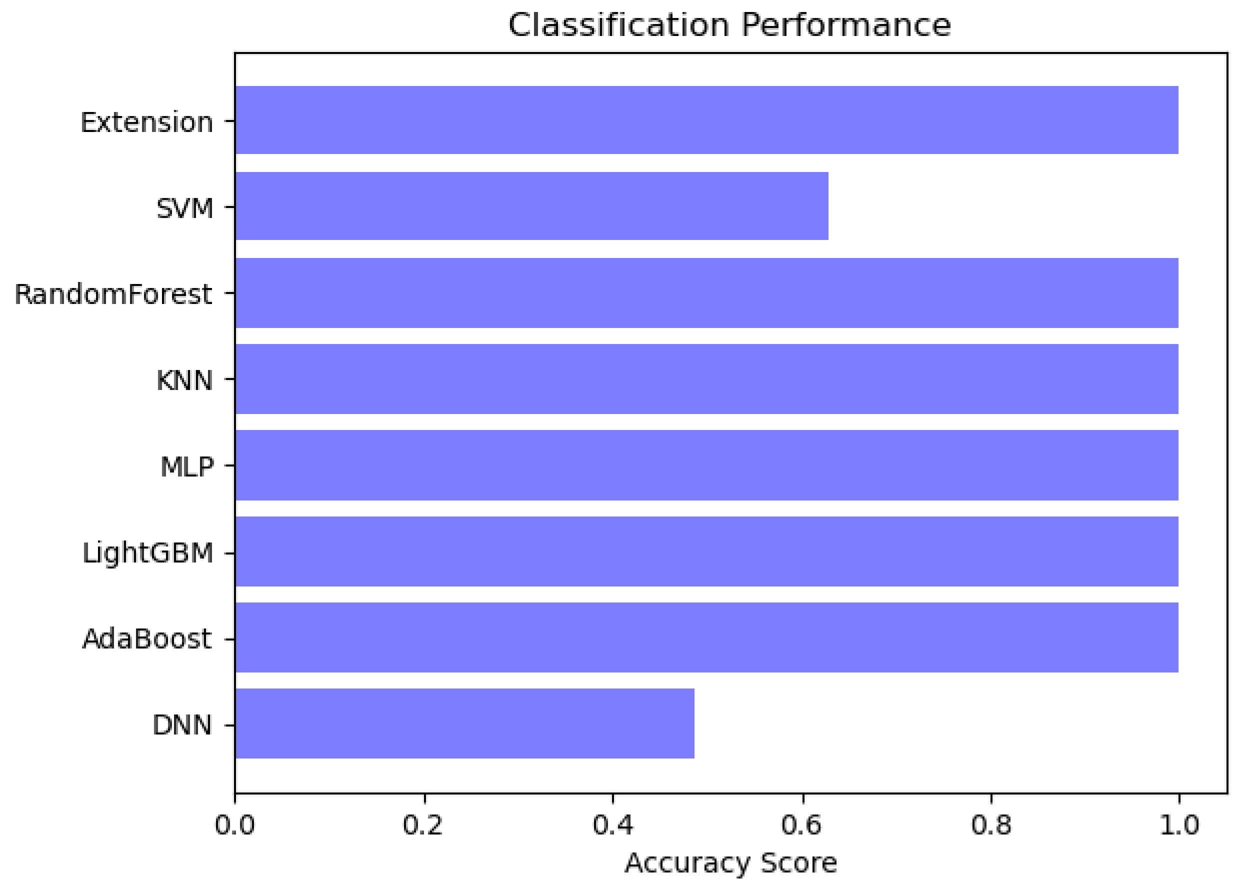
Out[34]: ['models/model_simarg.sav']

Graph

In [36]:
```
classifier = ML_Model
y_pos = np.arange(len(classifier))
```

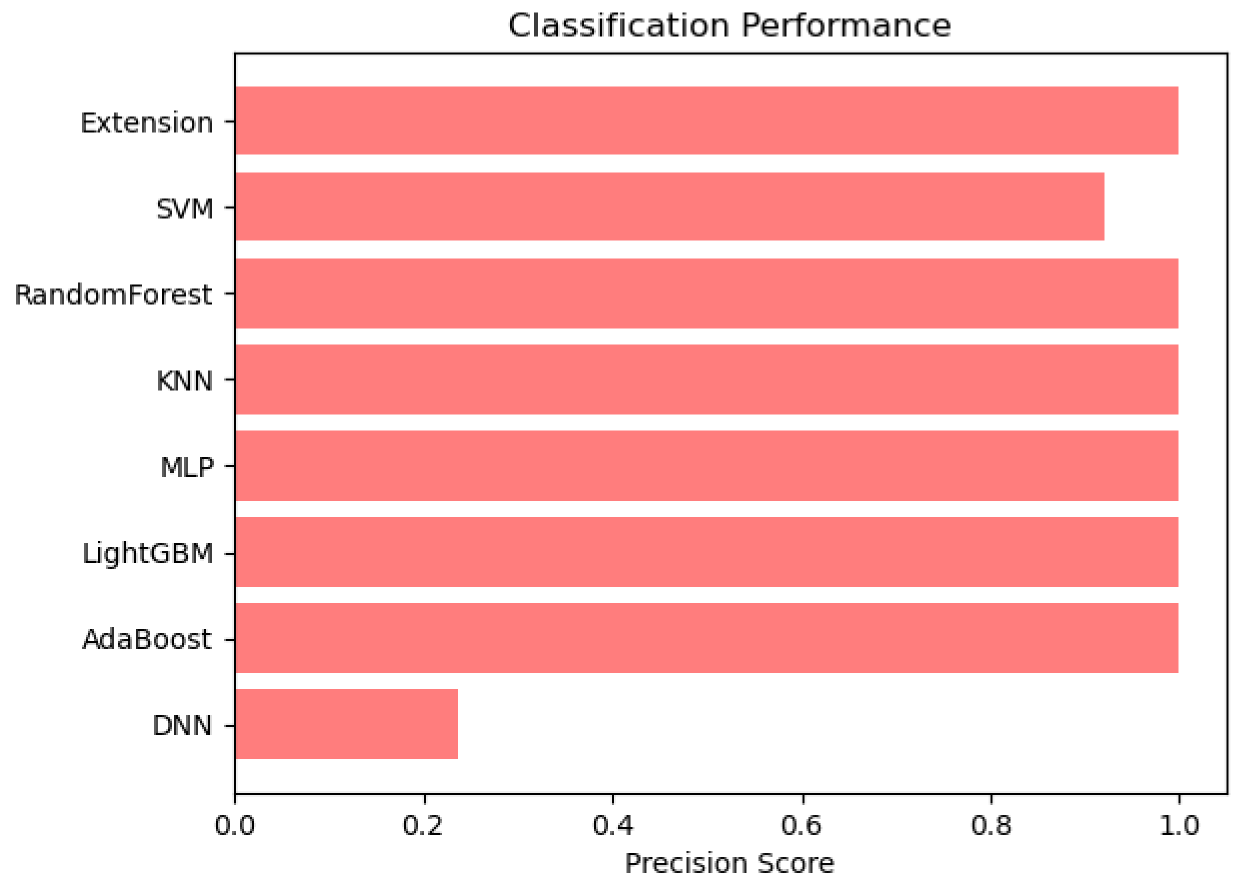
Accuracy

In [37]:
```
import matplotlib.pyplot as plt2
plt2.barh(y_pos, accuracy, align='center', alpha=0.5,color='blue')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Accuracy Score')
plt2.title('Classification Performance')
plt2.show()
```



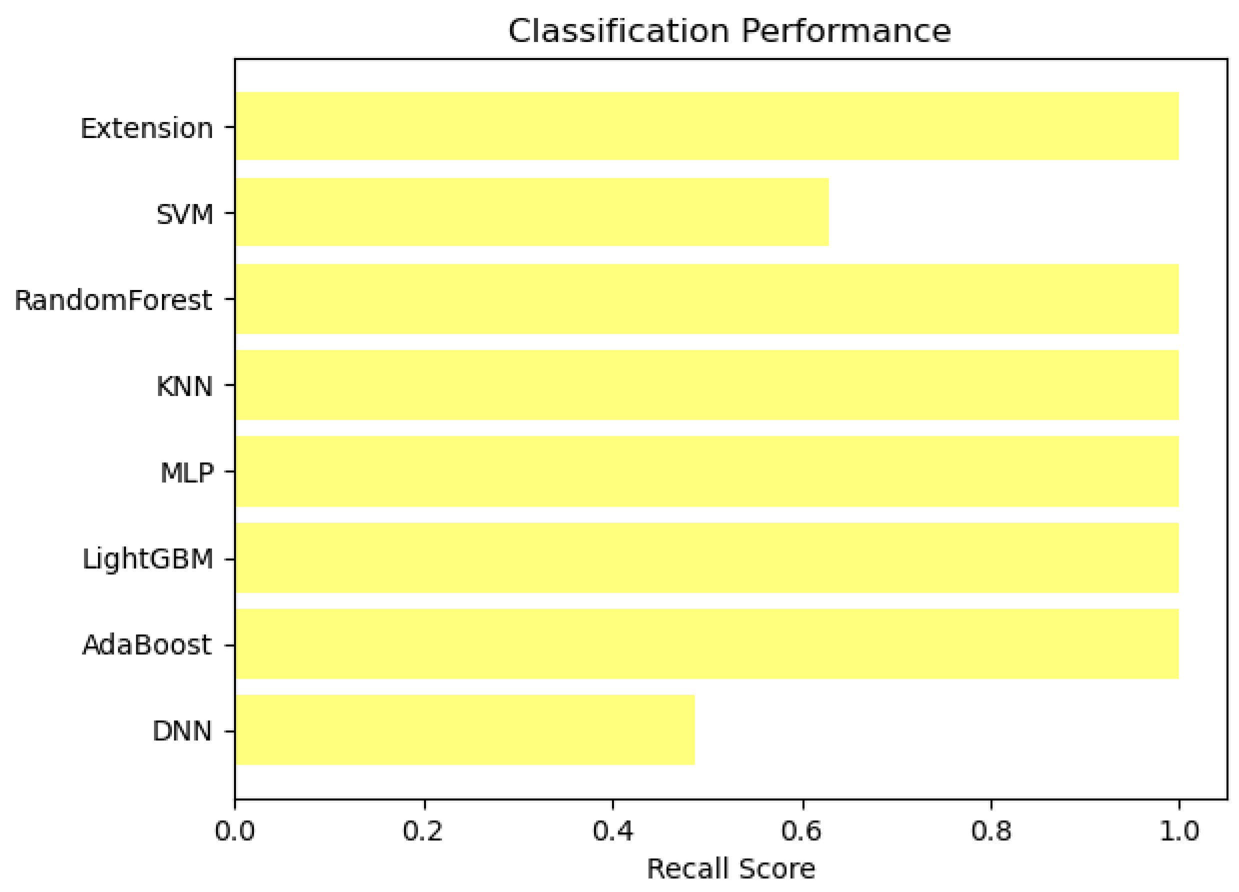
Precision

In [38]:
```
plt2.barh(y_pos, precision, align='center', alpha=0.5,color='red')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Precision Score')
plt2.title('Classification Performance')
plt2.show()
```



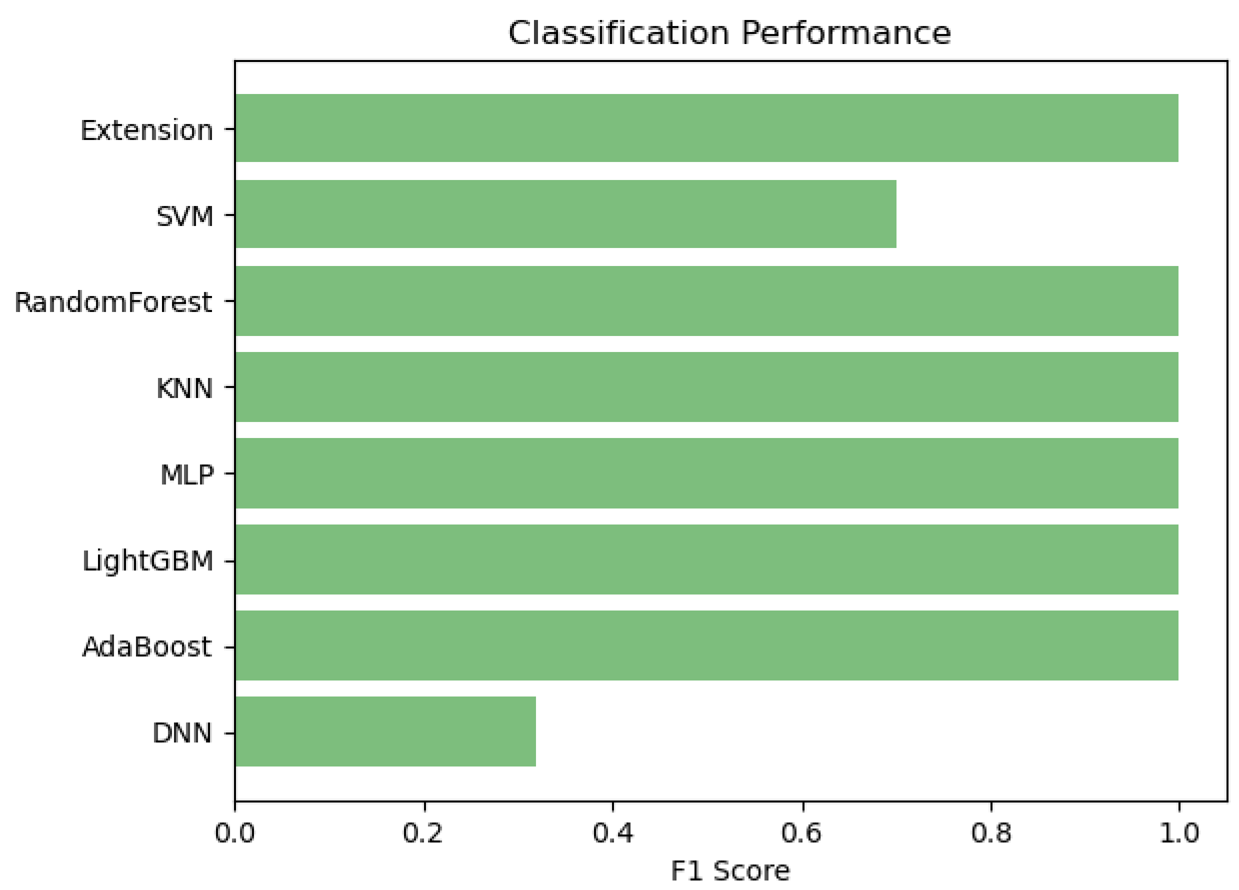
Recall

In [39]:
```
plt2.barh(y_pos, recall, align='center', alpha=0.5,color='yellow')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Recall Score')
plt2.title('Classification Performance')
plt2.show()
```



F1 Score

In [40]:
```
plt2.barh(y_pos, f1score, align='center', alpha=0.5,color='green')
plt2.yticks(y_pos, classifier)
plt2.xlabel('F1 Score')
plt2.title('Classification Performance')
plt2.show()
```



In [ ]: