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 [3]:
```
Columns = (['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgent','hot',
            'num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_root','num_file_creations',
            'num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','count','srv_count',
            'serror_rate','srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_srv_rate','srv_diff_host_rate',
            'dst_host_count','dst_host_srv_count','dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate',
            'dst_host_srv_diff_host_rate','dst_host_serror_rate','dst_host_srv_serror_rate','dst_host_rerror_rate',
            'dst_host_srv_rerror_rate','attack','level'])
```

In [4]:
```
data = pd.read_csv("nsl-kdd/KDDTrain+.txt" , sep = "," , encoding = 'utf-8')
```

In [5]:
```
# load data
data.columns = Columns
```

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: ['protocol_type', 'service', 'flag', 'attack']
```

Out[6]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serr
0	0	udp	other	SF	146	0	0	0	0	0	...	0.00	0.60	0.88	0.00	
1	0	tcp	private	S0	0	0	0	0	0	0	...	0.10	0.05	0.00	0.00	
2	0	tcp	http	SF	232	8153	0	0	0	0	...	1.00	0.00	0.03	0.04	
3	0	tcp	http	SF	199	420	0	0	0	0	...	1.00	0.00	0.00	0.00	
4	0	tcp	private	REJ	0	0	0	0	0	0	...	0.07	0.07	0.00	0.00	

5 rows × 43 columns

In [7]:
```
# number of attack labels
data['attack'].value_counts()
```

Out[7]:
```
normal             67342
neptune            41214
satan               3633
ipsweep             3599
portsweep           2931
smurf               2646
nmap                1493
back                 956
teardrop             892
warezclient          890
pod                  201
guess_passwd          53
buffer_overflow       30
warezmaster           20
land                  18
imap                  11
rootkit               10
loadmodule             9
ftp_write              8
multihop               7
phf                    4
perl                   3
spy                    2
Name: attack, dtype: int64
```

In [8]:
```
# changing attack labels to their respective attack class
def change_label(df):
  df.attack.replace(['back','land','neptune','pod','smurf','teardrop'],'Dos',inplace=True)
  df.attack.replace(['guess_passwd','imap','ftp_write','multihop','phf','spy','warezclient','warezmaster'],'R2L',inplace=True)
  df.attack.replace(['ipsweep','nmap','portsweep','satan'],'Probe',inplace=True)
  df.attack.replace(['buffer_overflow','loadmodule','perl','rootkit'],'U2R',inplace=True)
```

In [9]:
```
change_label(data)
```

In [10]:
```
# distribution of attack classes
data.attack.value_counts()
```

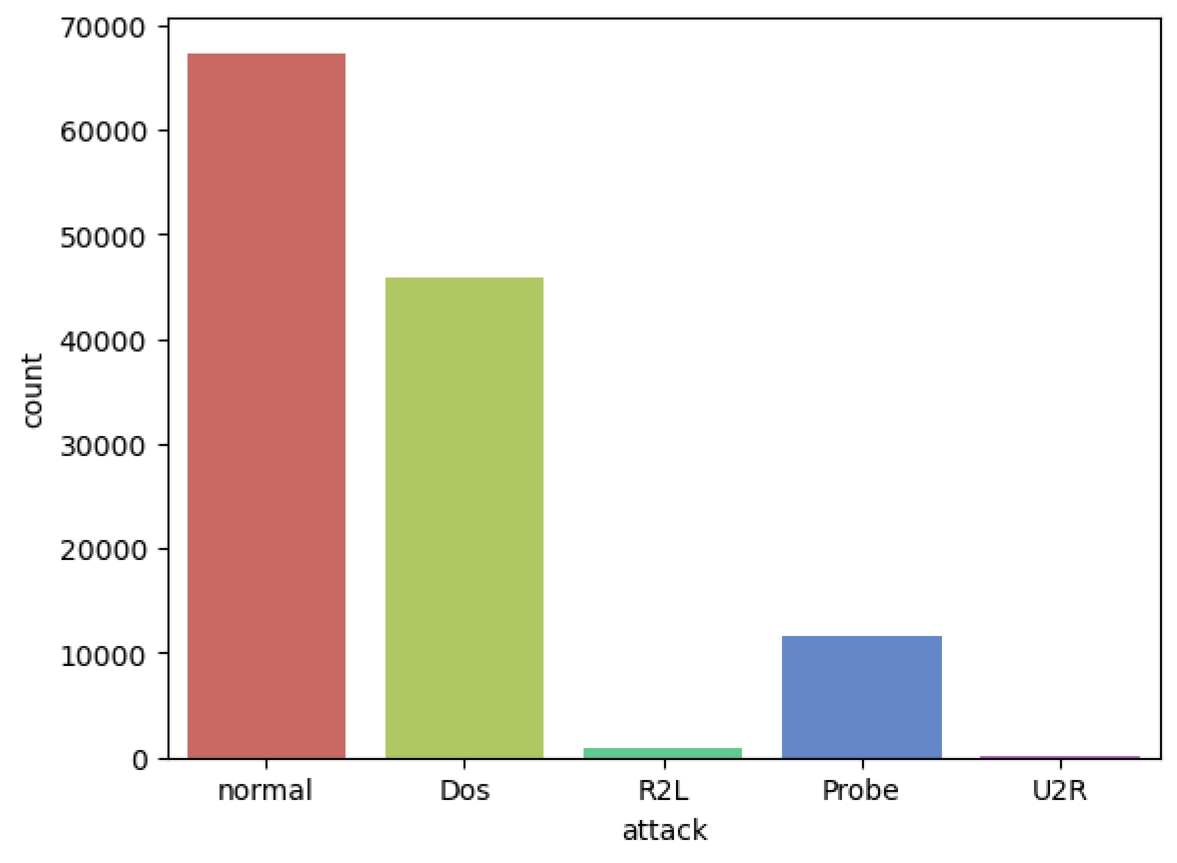
Out[10]:
```
normal    67342
Dos       45927
Probe     11656
R2L         995
U2R          52
Name: attack, dtype: int64
```

In [11]:
```
data.info()
```

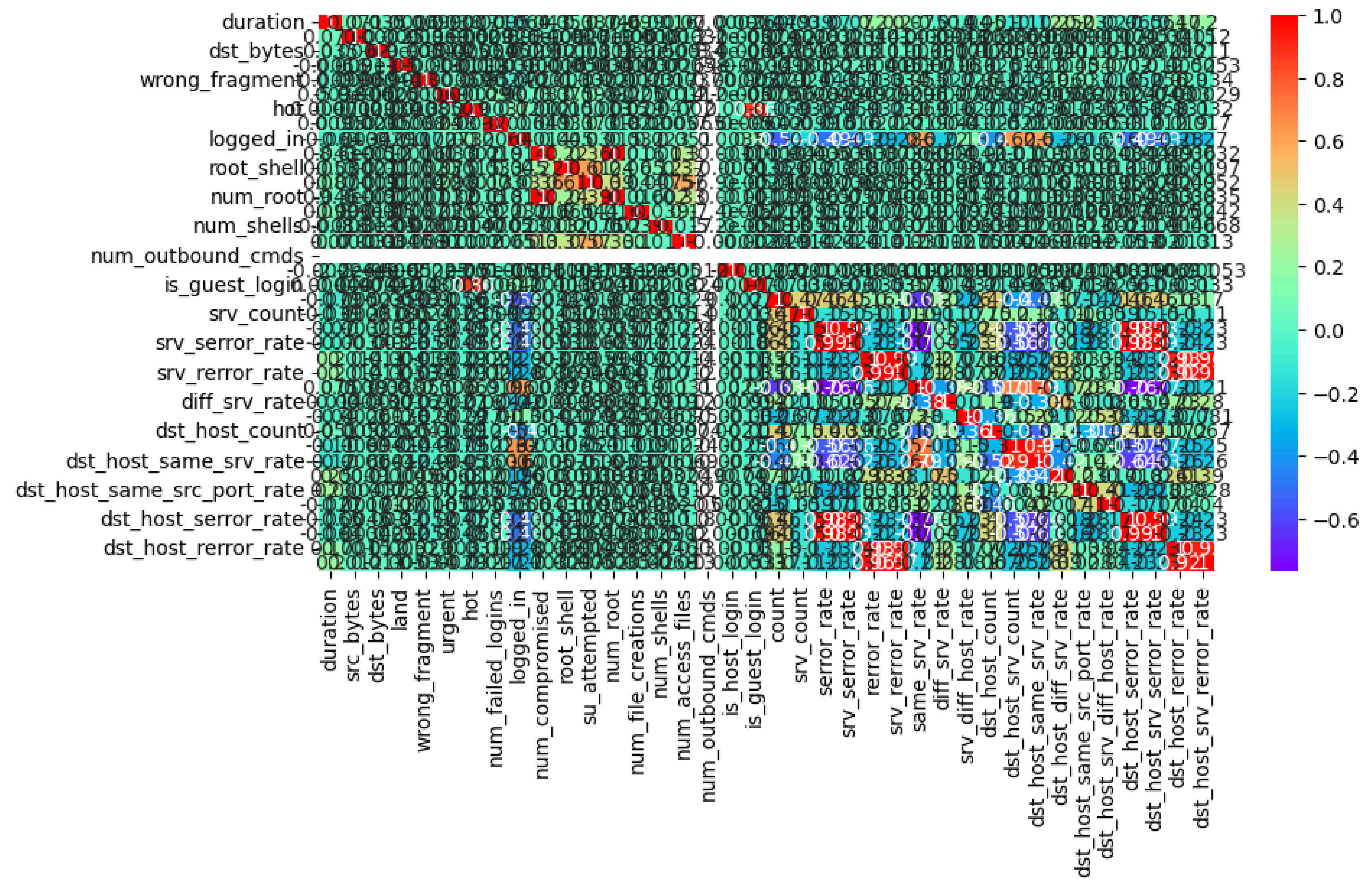
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125972 entries, 0 to 125971
Data columns (total 43 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   duration                     125972 non-null  int64  
 1   protocol_type                125972 non-null  object 
 2   service                      125972 non-null  object 
 3   flag                         125972 non-null  object 
 4   src_bytes                    125972 non-null  int64  
 5   dst_bytes                    125972 non-null  int64  
 6   land                         125972 non-null  int64  
 7   wrong_fragment               125972 non-null  int64  
 8   urgent                       125972 non-null  int64  
 9   hot                          125972 non-null  int64  
 10  num_failed_logins            125972 non-null  int64  
 11  logged_in                    125972 non-null  int64  
 12  num_compromised              125972 non-null  int64  
 13  root_shell                   125972 non-null  int64  
 14  su_attempted                 125972 non-null  int64  
 15  num_root                     125972 non-null  int64  
 16  num_file_creations           125972 non-null  int64  
 17  num_shells                   125972 non-null  int64  
 18  num_access_files             125972 non-null  int64  
 19  num_outbound_cmds            125972 non-null  int64  
 20  is_host_login                125972 non-null  int64  
 21  is_guest_login               125972 non-null  int64  
 22  count                        125972 non-null  int64  
 23  srv_count                    125972 non-null  int64  
 24  serror_rate                  125972 non-null  float64
 25  srv_serror_rate              125972 non-null  float64
 26  rerror_rate                  125972 non-null  float64
 27  srv_rerror_rate              125972 non-null  float64
 28  same_srv_rate                125972 non-null  float64
 29  diff_srv_rate                125972 non-null  float64
 30  srv_diff_host_rate           125972 non-null  float64
 31  dst_host_count               125972 non-null  int64  
 32  dst_host_srv_count           125972 non-null  int64  
 33  dst_host_same_srv_rate       125972 non-null  float64
 34  dst_host_diff_srv_rate       125972 non-null  float64
 35  dst_host_same_src_port_rate  125972 non-null  float64
 36  dst_host_srv_diff_host_rate  125972 non-null  float64
 37  dst_host_serror_rate         125972 non-null  float64
 38  dst_host_srv_serror_rate     125972 non-null  float64
 39  dst_host_rerror_rate         125972 non-null  float64
 40  dst_host_srv_rerror_rate     125972 non-null  float64
 41  attack                       125972 non-null  object 
 42  level                        125972 non-null  int64  
dtypes: float64(15), int64(24), object(4)
memory usage: 41.3+ MB
```

In [13]:
```
del data['level']
```

In [12]:
```
sns.countplot(x='attack',data=data, palette='hls')
plt.show()
#plt.savefig('count_plot') mal: the nodule malignancy, 0: benign, 1: malignant
```



In [14]:
```python
plt.figure(figsize = (10,5))
sns.heatmap(data.corr(), annot = True, cmap="rainbow")
plt.show()
```



In [16]:
```python
# 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['attack']= label_encoder.fit_transform(data['attack'])
data['protocol_type']= label_encoder.fit_transform(data['protocol_type'])
data['service']= label_encoder.fit_transform(data['service'])
data['flag']= label_encoder.fit_transform(data['flag'])
```

In [17]:
```python
X = data.drop(["attack"],axis =1)
y = data["attack"]
```

FS

In [18]:
```python
from sklearn.feature_selection import SelectKBest, SelectPercentile, mutual_info_classif
```

In [19]:
```python
selector = SelectPercentile(mutual_info_classif, percentile=25)
X_reduced = selector.fit_transform(X, y)
#X_reduced.shape
```

In [20]:
```python
cols = selector.get_support(indices=True)
selected_columns = X.iloc[:,cols].columns.tolist()
selected_columns
```

Out[20]:
```
['service',
 'flag',
 'src_bytes',
 'dst_bytes',
 'logged_in',
 'count',
 'same_srv_rate',
 'diff_srv_rate',
 'dst_host_srv_count',
 'dst_host_diff_srv_rate']
```

In [21]:
```python
len(selected_columns)
```

Out[21]:
```
10
```

In [22]:
```python
df = data[['service',
         'flag',
         'src_bytes',
         'dst_bytes',
         'logged_in',
         'count',
         'same_srv_rate',
         'diff_srv_rate',
         'dst_host_srv_count',
         'dst_host_diff_srv_rate','attack']]
```

In [23]:
```python
df.to_csv('nsl_processed.csv')
```

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

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

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

Out[5]:
```
Index(['service', 'flag', 'src_bytes', 'dst_bytes', 'logged_in', 'count',
       'same_srv_rate', 'diff_srv_rate', 'dst_host_srv_count',
       'dst_host_diff_srv_rate', 'attack'],
      dtype='object')
```

In [6]:
```python
X = df[['service', 'flag', 'src_bytes', 'dst_bytes', 'logged_in', 'count',
       'same_srv_rate', 'diff_srv_rate', 'dst_host_srv_count',
       'dst_host_diff_srv_rate',]]
y = df["attack"]
```

In [7]:
```python
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 [10]:
```
# importing lime
import lime
from lime import lime_tabular
import shap
```

DNN

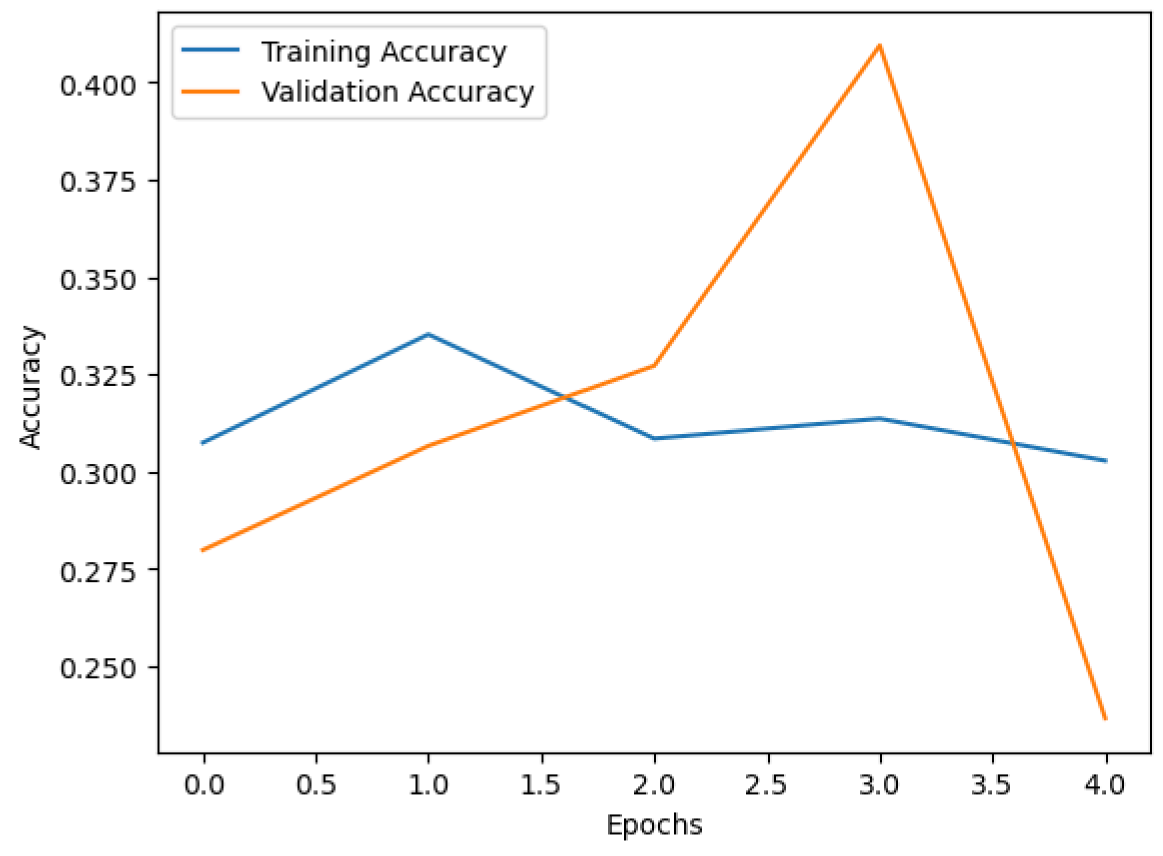
In [11]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

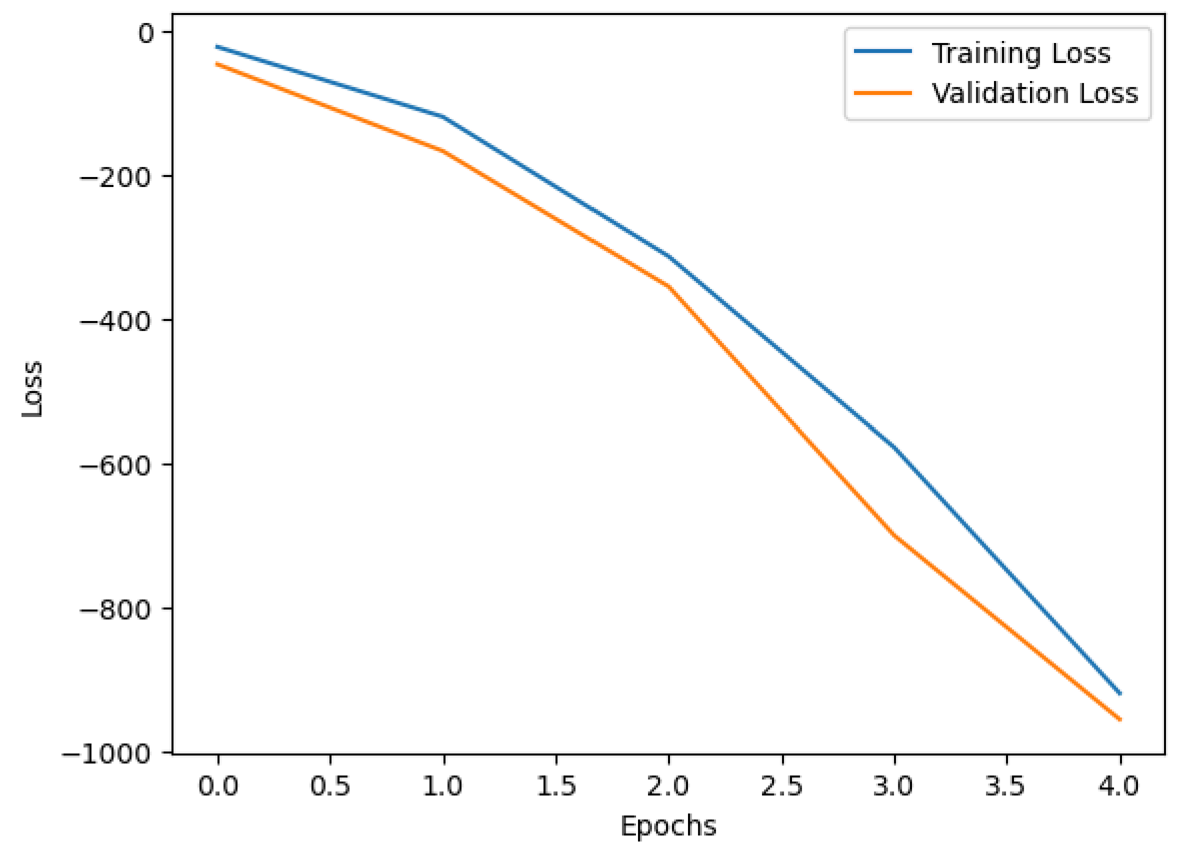
In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)
# Scale the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

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
486/486 [==============================] - 5s 7ms/step - loss: -21.1100 - accuracy: 0.2765 - val_loss: -70.3031 - val_accuracy: 0.3086
Epoch 2/5
486/486 [==============================] - 3s 7ms/step - loss: -115.8501 - accuracy: 0.2827 - val_loss: -226.5943 - val_accuracy: 0.2263
Epoch 3/5
486/486 [==============================] - 3s 7ms/step - loss: -307.2403 - accuracy: 0.2570 - val_loss: -487.6276 - val_accuracy: 0.2716
Epoch 4/5
486/486 [==============================] - 3s 6ms/step - loss: -613.3693 - accuracy: 0.2703 - val_loss: -875.3435 - val_accuracy: 0.2490
Epoch 5/5
486/486 [==============================] - 3s 7ms/step - loss: -966.9393 - accuracy: 0.2585 - val_loss: -1339.9371 - val_accuracy: 0.2366
```

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')
```

```
51/51 [==============================] - 0s 2ms/step
```

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

In [14]:
```
lime_explainer = lime_tabular.LimeTabularExplainer(training_data=np.array(X_train), feature_names=X_train.columns,
            class_names=['0','1', '2', '3','4'], 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 [15]:
```
explanation.show_in_notebook()
```

ML

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 42)
```

AdaBoost

In [17]:
```
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)
```

LIME

In [18]:
```
lime_explainer = lime_tabular.LimeTabularExplainer(training_data=np.array(X_train), feature_names=X_train.columns,
            class_names=['0','1', '2', '3','4'], 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 [20]:
```
# 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')
```

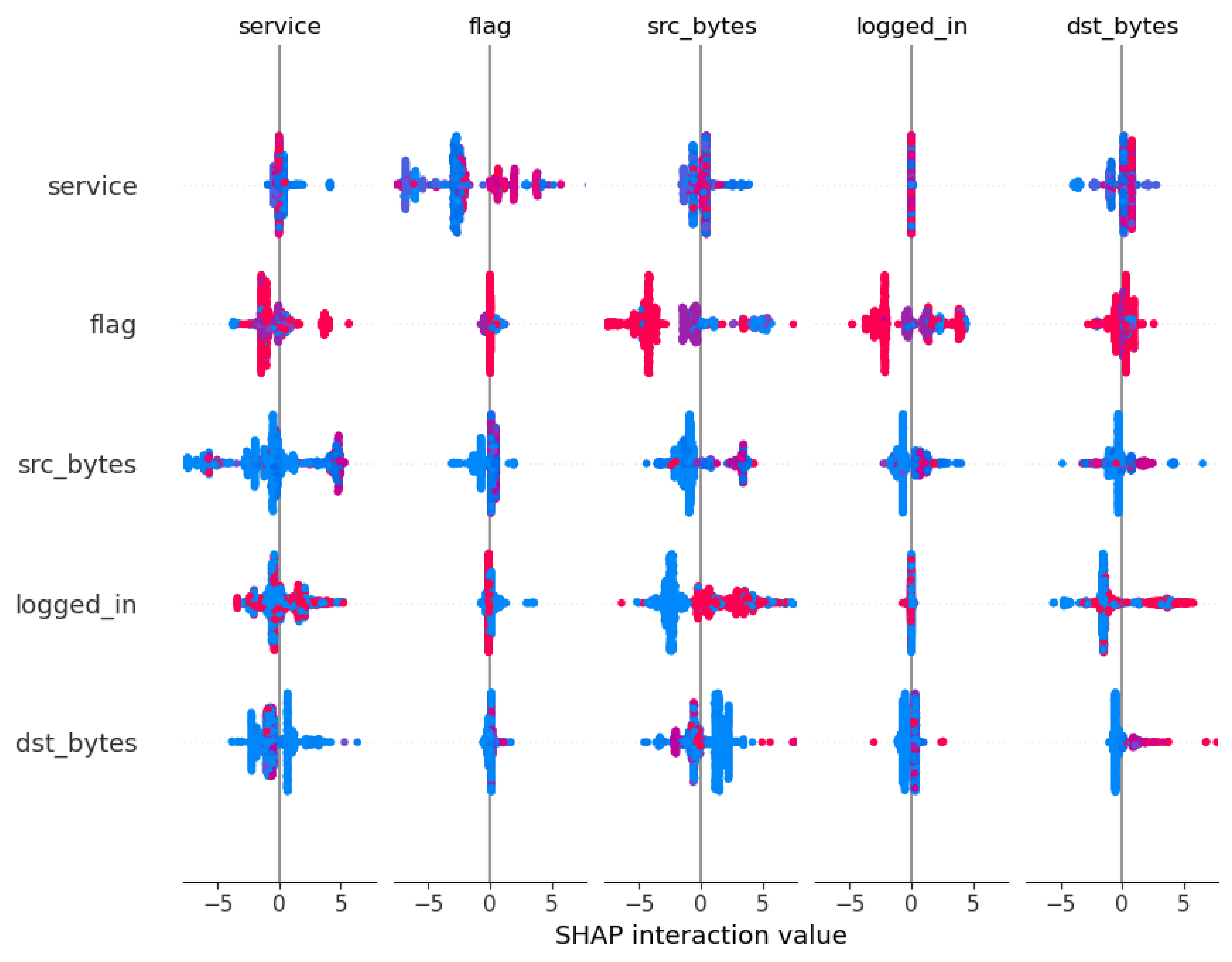
```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000469 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1190
[LightGBM] [Info] Number of data points in the train set: 3237, number of used features: 10
[LightGBM] [Info] Start training from score -1.412904
[LightGBM] [Info] Start training from score -1.390319
[LightGBM] [Info] Start training from score -1.379214
[LightGBM] [Info] Start training from score -4.418841
[LightGBM] [Info] Start training from score -1.411636
[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
[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
[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
[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
[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
[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
[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
[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
[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
[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
[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
[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
[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
[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
[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
[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
[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)
```

LIME

In [21]:
```
lime_explainer = lime_tabular.LimeTabularExplainer(training_data=np.array(X_train), feature_names=X_train.columns,
            class_names=['0','1', '2', '3','4'], 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 [22]:
```
explainer = shap.Explainer(clf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



MLP

In [23]:
```
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 [24]:
```
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 [25]:
```
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 [26]:
```
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 [27]:
```
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 [28]:
```
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 [29]:
```
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 [30]:
```
storeResults('SVM',svc_acc,svc_prec,svc_rec,svc_f1)
```

In [31]:
```
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 [32]:
```
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 [33]:
```
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	0.253	0.064	0.253	0.102
1	AdaBoost	0.699	0.830	0.699	0.733
2	LightGBM	0.984	0.985	0.984	0.984
3	MLP	0.847	0.837	0.847	0.840
4	KNN	0.935	0.938	0.935	0.936
5	RandomForest	0.977	0.978	0.977	0.977
6	SVM	0.257	0.984	0.257	0.387
7	Extension	0.994	0.994	0.994	0.994

Modelling

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

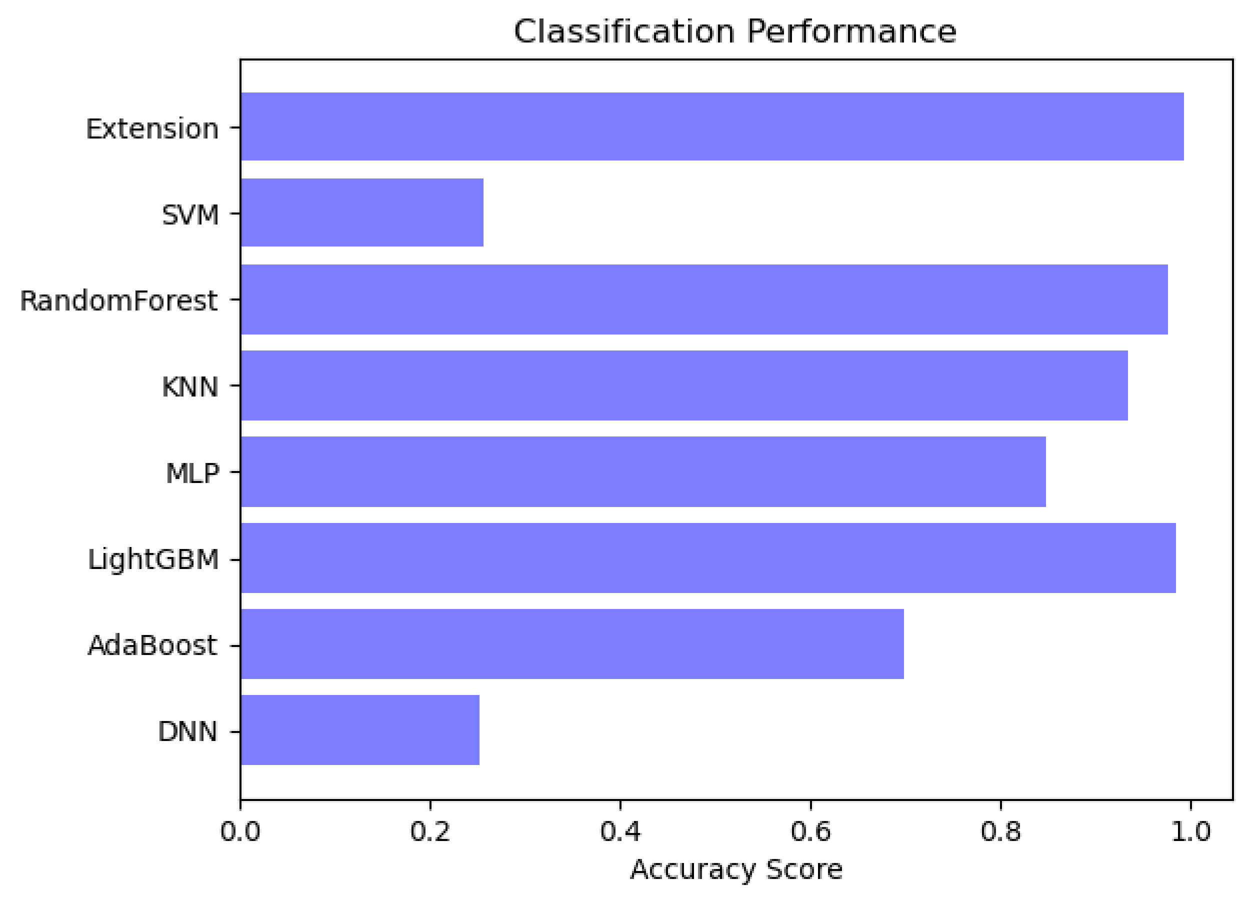
Out[34]:
```
['models/model_nsl.sav']
```

Graph

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

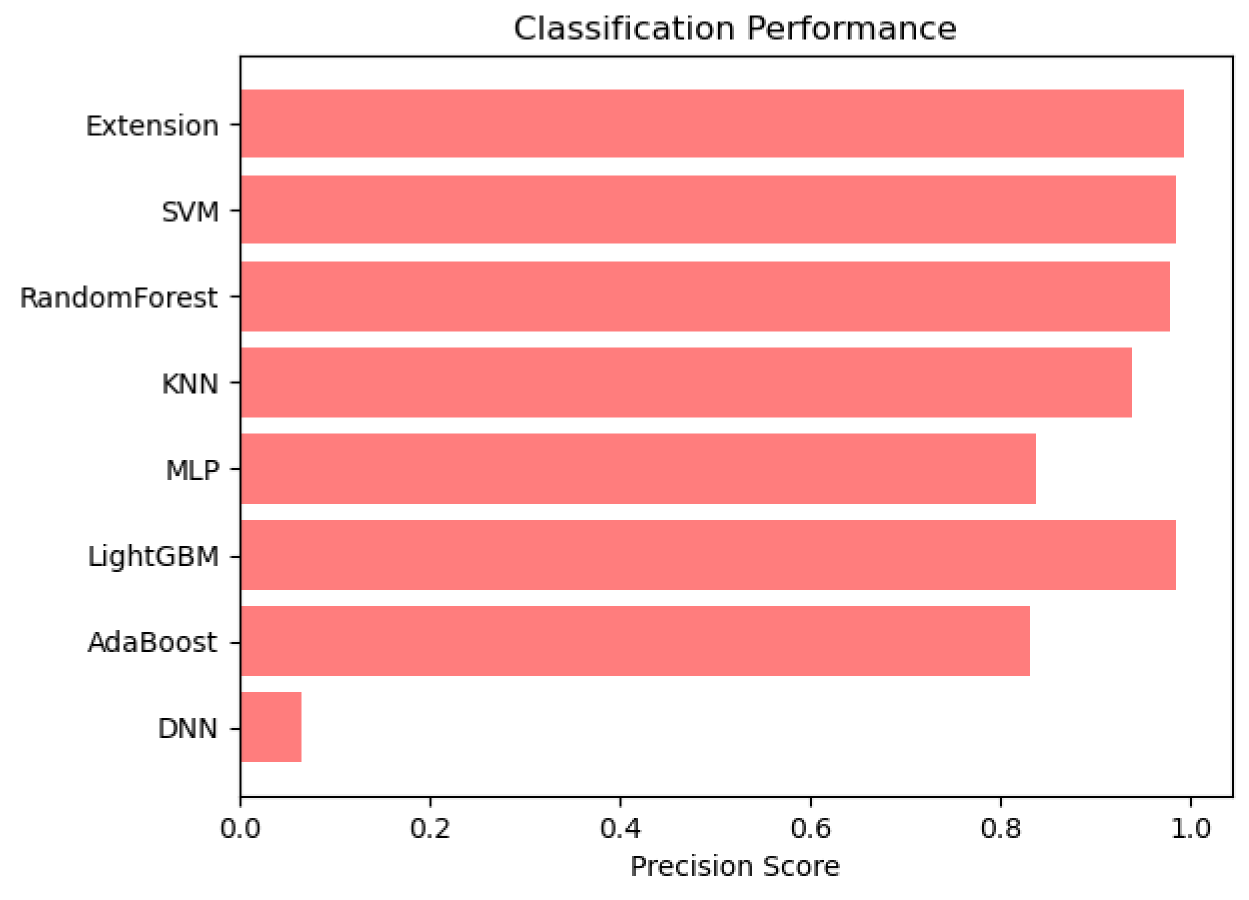
Accuracy

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()
```



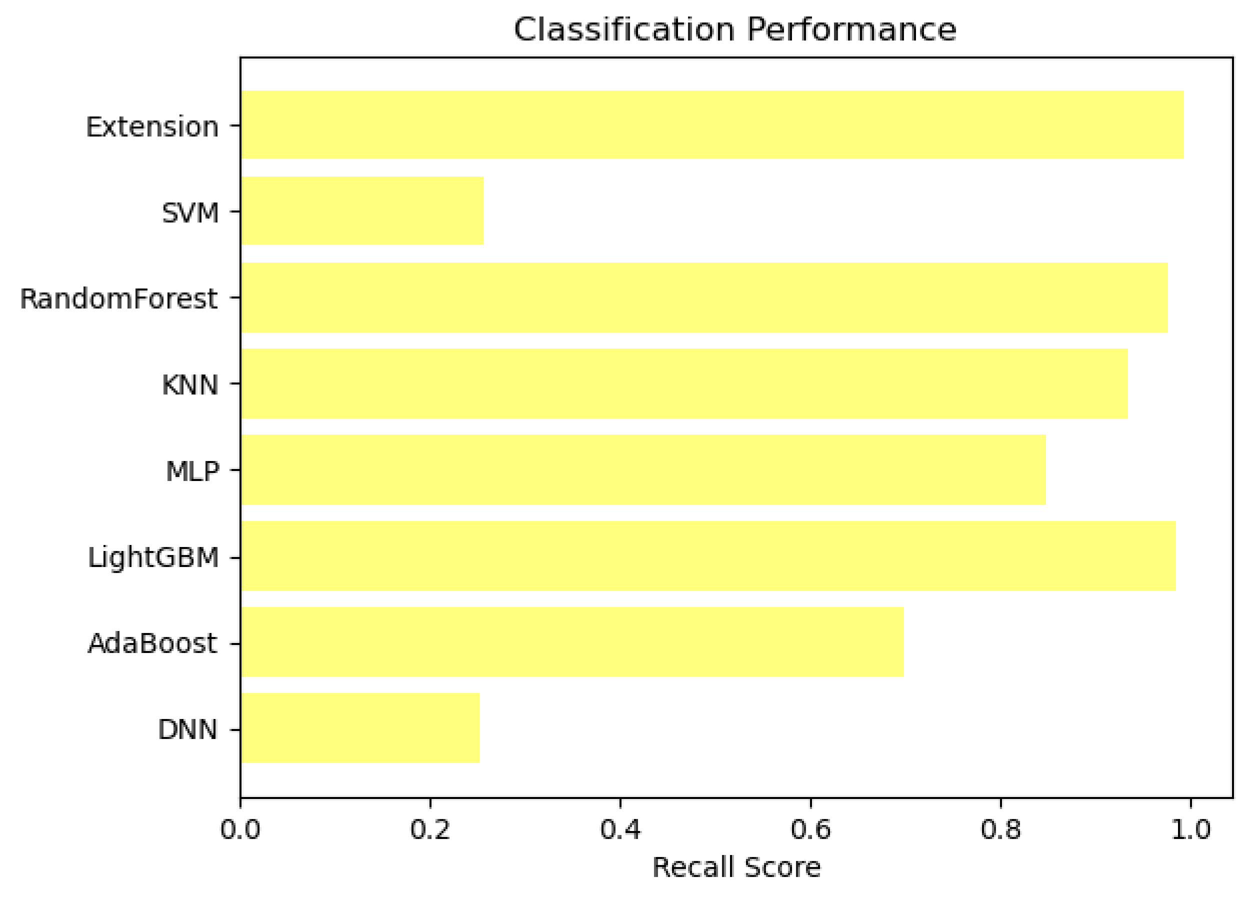
Precision

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()
```



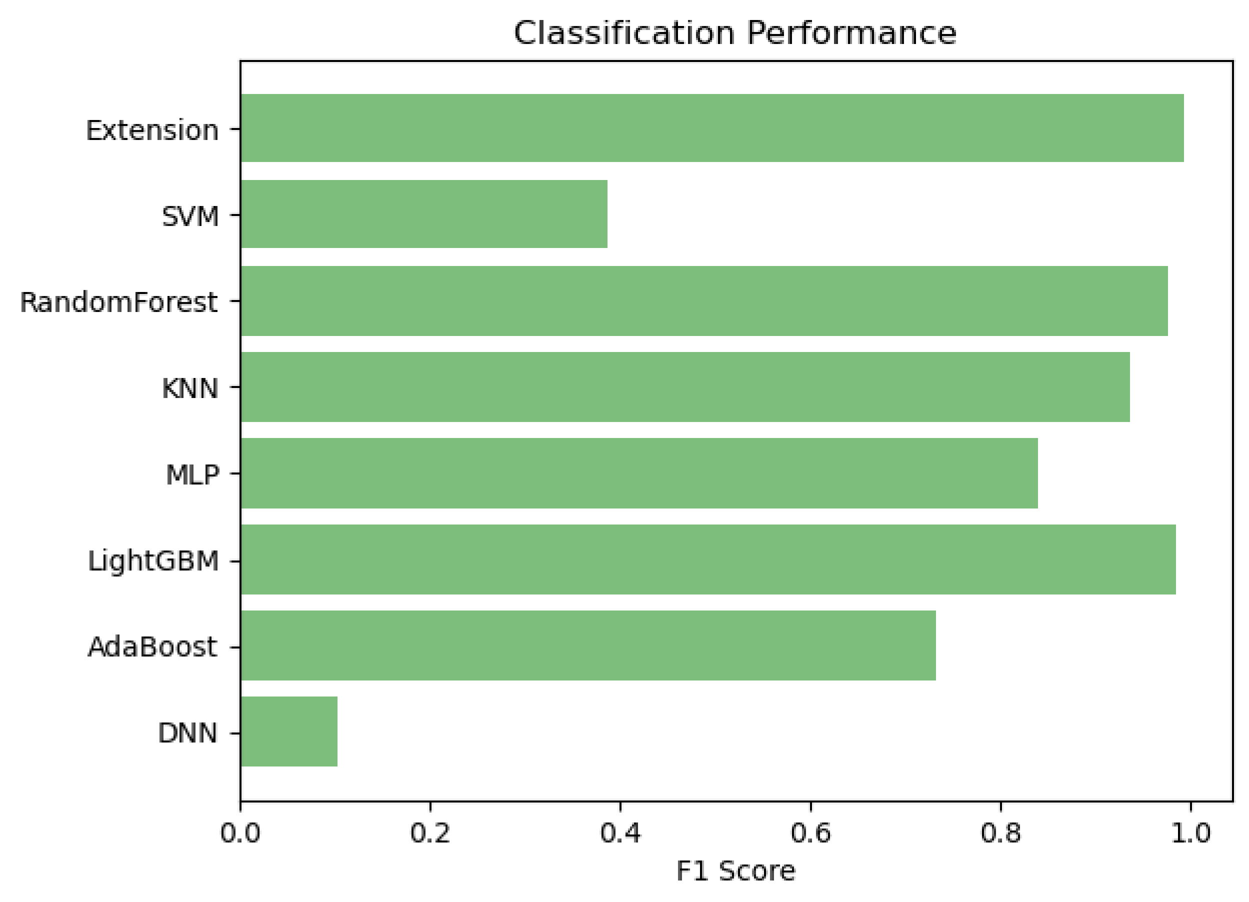
Recall

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 [ ]:
```
```