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

In [2]:
```python
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```python
data = pd.read_csv('SDN Dataset/dataset_sdn.csv')
```

In [4]:
```python
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()
```

```
1012 null entries have been found in the dataset

5091 duplicate entries have been found in the dataset

All duplicates have been removed

Categorical columns: ['src', 'dst', 'Protocol']
```

Out[4]:

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_bytes	tx_kbps	rx_kbps	tot_kbps	label
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	3	143928631	3917	0	0.0	0.0	0
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451	0	UDP	4	3842	3520	0	0.0	0.0	0
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	3795	1242	0	0.0	0.0	0
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	2	3688	1492	0	0.0	0.0	0
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	3	3413	3665	0	0.0	0.0	0

5 rows × 23 columns

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

Out[5]:
```
Index(['dt', 'switch', 'src', 'dst', 'pktcount', 'bytecount', 'dur',
       'dur_nsec', 'tot_dur', 'flows', 'packetins', 'pktperflow',
       'byteperflow', 'pktrate', 'Pairflow', 'Protocol', 'port_no', 'tx_bytes',
       'rx_bytes', 'tx_kbps', 'rx_kbps', 'tot_kbps', 'label'],
      dtype='object')
```

In [7]:
```python
data['label'].value_counts()
```

Out[7]:
```
0    61022
1    37726
Name: label, dtype: int64
```

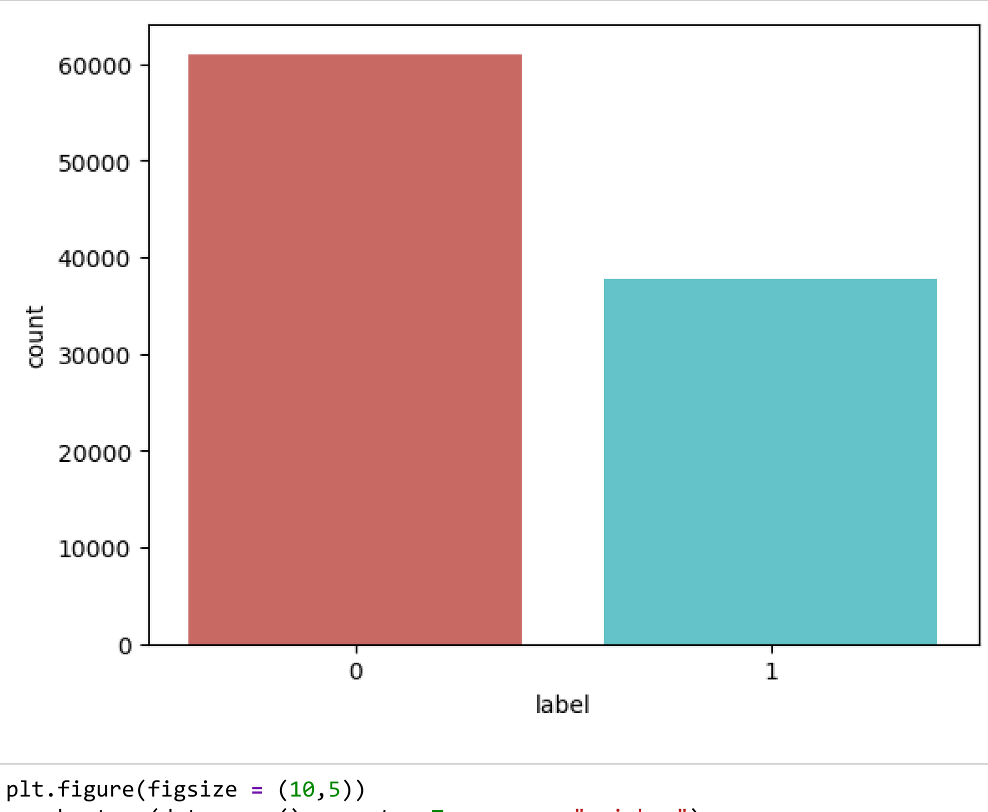
In [9]:
```python
del data['src']
del data['dst']
del data['Protocol']
```

In [10]:
```python
#change_label(data)
```

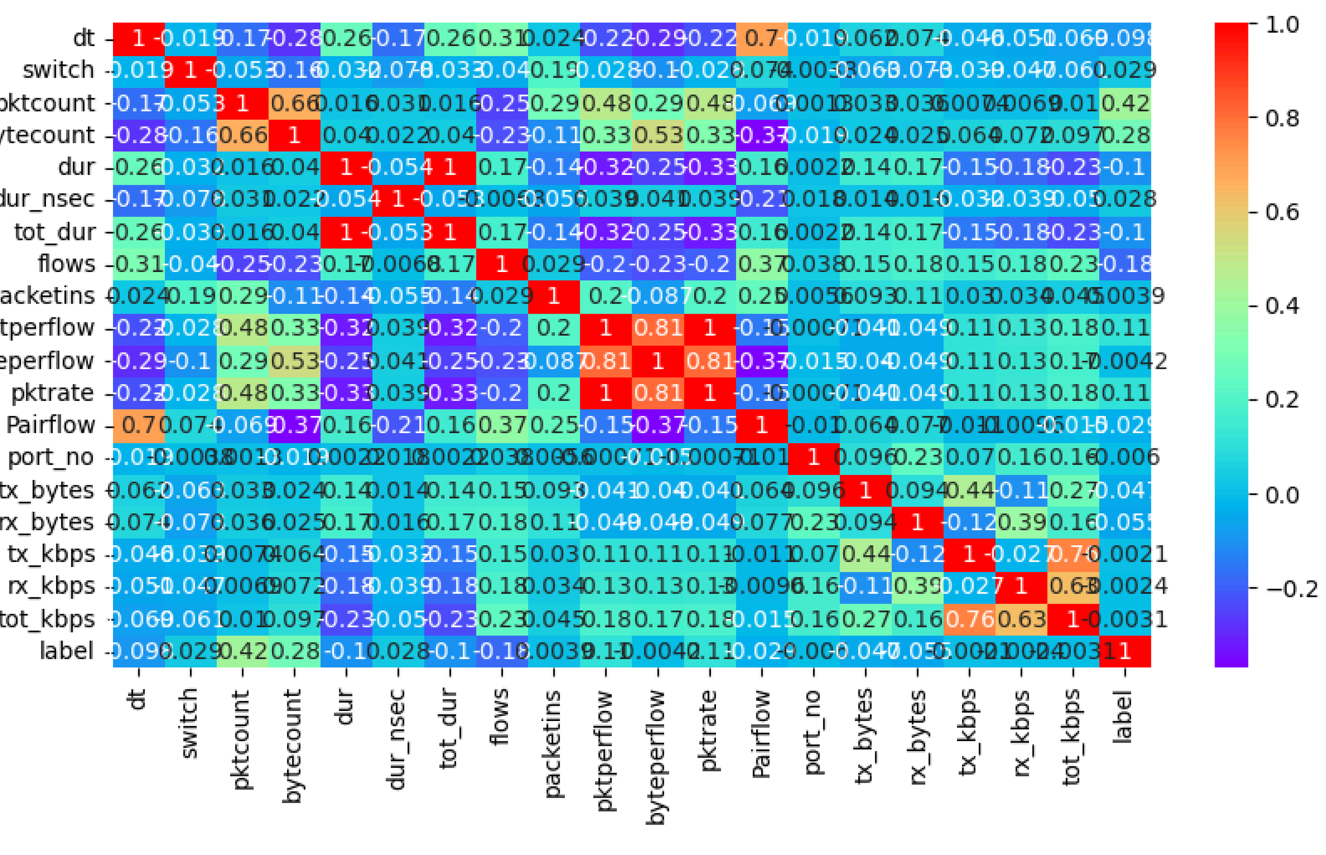
In [11]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98748 entries, 0 to 98747
Data columns (total 20 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   dt           98748 non-null  int64  
 1   switch       98748 non-null  int64  
 2   pktcount     98748 non-null  int64  
 3   bytecount    98748 non-null  int64  
 4   dur          98748 non-null  int64  
 5   dur_nsec     98748 non-null  int64  
 6   tot_dur      98748 non-null  float64
 7   flows        98748 non-null  int64  
 8   packetins    98748 non-null  int64  
 9   pktperflow   98748 non-null  int64  
 10  byteperflow  98748 non-null  int64  
 11  pktrate      98748 non-null  int64  
 12  Pairflow     98748 non-null  int64  
 13  port_no      98748 non-null  int64  
 14  tx_bytes     98748 non-null  int64  
 15  rx_bytes     98748 non-null  int64  
 16  tx_kbps      98748 non-null  int64  
 17  rx_kbps      98748 non-null  float64
 18  tot_kbps     98748 non-null  float64
 19  label        98748 non-null  int64  
dtypes: float64(3), int64(17)
memory usage: 15.1 MB
```

In [12]:
```python
sns.countplot(x='label',data=data, palette='hls')
plt.show()
#plt.savefig('count_plot') Labeling traffic as normal (0) or malicious (1).
```



In [13]:
```python
plt.figure(figsize = (10,5))
sns.heatmap(data.corr(), annot = True, cmap="rainbow")
plt.show()
```



In [15]:
```
#print(data.info())
#data = pd.concat([

    #data[data.Label == 'Attack'].sample(n=50_00),
    #data[data.Label == 'Normal'].sample(n=50_00),
#])
```

In [16]:
```
# 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'])
```

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

FS

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

In [24]:
```
selector = SelectPercentile(mutual_info_classif, percentile=30)
X_reduced = selector.fit_transform(X, y)
#X_reduced.shape
```

In [25]:
```
cols = selector.get_support(indices=True)
selected_columns = X.iloc[:,cols].columns.tolist()
selected_columns
```

Out[25]:
```
['dt', 'pktcount', 'bytecount', 'pktperflow', 'byteperflow', 'pktrate']
```

In [26]:
```
len(selected_columns)
```

Out[26]:
```
6
```

In [27]:
```
df = data[['dt', 'pktcount', 'bytecount', 'pktperflow', 'byteperflow', 'pktrate','label']]
```

In [28]:
```
df.columns
```

Out[28]:
```
Index(['dt', 'pktcount', 'bytecount', 'pktperflow', 'byteperflow', 'pktrate',
       'label'],
      dtype='object')
```

In [30]:
```
X = df.drop(["label"],axis =1)
y = df["label"]
```

In [31]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 42)
#X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

In [32]:
```
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 [33]:
```
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))
```

BernoulliNB

In [34]:
```
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB(alpha=1.0, binarize=0.0, fit_prior=True, class_prior=None)

bnb.fit(X_train, y_train)

y_pred = bnb.predict(X_test)

bnb_acc = accuracy_score(y_pred, y_test)
bnb_prec = precision_score(y_pred, y_test,average='weighted')
bnb_rec = recall_score(y_pred, y_test,average='weighted')
bnb_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [35]:
```
storeResults('BernoulliNB',bnb_acc,bnb_prec,bnb_rec,bnb_f1)
```

Passive Aggressive

In [36]:
```
from sklearn.linear_model import PassiveAggressiveClassifier

pa = PassiveAggressiveClassifier(C=1.0, fit_intercept=True, max_iter=1000, tol=0.001, early_stopping=False,
                                 validation_fraction=0.1, n_iter_no_change=5, shuffle=True, verbose=0,
                                 loss='hinge', n_jobs=None, random_state=None, warm_start=False,
                                 class_weight=None, average=False)

pa.fit(X_train, y_train)

y_pred = pa.predict(X_test)

pa_acc = accuracy_score(y_pred, y_test)
pa_prec = precision_score(y_pred, y_test,average='weighted')
pa_rec = recall_score(y_pred, y_test,average='weighted')
pa_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [37]:
```
storeResults('PassiveAggressive',pa_acc,pa_prec,pa_rec,pa_f1)
```

SGDClassifier

In [38]:
```
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
                   max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None,
                   random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False,
                   validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False)

sgd.fit(X_train, y_train)

y_pred = sgd.predict(X_test)

sgd_acc = accuracy_score(y_pred, y_test)
sgd_prec = precision_score(y_pred, y_test,average='weighted')
sgd_rec = recall_score(y_pred, y_test,average='weighted')
sgd_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [39]:
```
storeResults('SGDClassifier',sgd_acc,sgd_prec,sgd_rec,sgd_f1)
```

MLP Classifier

In [40]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto',
                    learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True,
                    random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True,
                    early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
                    n_iter_no_change=10, max_fun=15000)

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 [41]:
```
storeResults('MLPClassifier',mlp_acc,mlp_prec,mlp_rec,mlp_f1)
```

Ensemble

In [42]:
```
from sklearn.ensemble import VotingClassifier

eclf1 = VotingClassifier(estimators=[('BNB', bnb),('PA', pa),('SGD', sgd),('MLP', mlp)], voting='hard')


eclf1.fit(X_train, y_train)

y_pred = eclf1.predict(X_test)

stac_acc = accuracy_score(y_pred, y_test)
stac_prec = precision_score(y_pred, y_test,average='weighted')
stac_rec = recall_score(y_pred, y_test,average='weighted')
stac_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [43]:
```
storeResults('Ensemble',stac_acc,stac_prec,stac_rec,stac_f1)
```

Extension

In [44]:
```
from sklearn.ensemble import VotingClassifier, AdaBoostClassifier, RandomForestClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

brf = BaggingClassifier(RandomForestClassifier(),n_estimators=10, random_state=0,max_samples=1.0,max_features=1.0)

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

ext = VotingClassifier(estimators=[('BoostDT', bdt),('BagRF', brf)], voting='soft')
ext.fit(X_train, y_train)

y_pred    = ext.predict(X_test)

ml_acc = accuracy_score(y_pred, y_test)
ml_prec = precision_score(y_pred, y_test,average='weighted')
ml_rec = recall_score(y_pred, y_test,average='weighted')
ml_f1 = f1_score(y_pred, y_test,average='weighted')
```

In [45]:
```
storeResults('Extension',ml_acc,ml_prec,ml_rec,ml_f1)
```

Comparison

In [46]:
```
#creating dataframe
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                       'Precision': precision,
                       'Recall'   : recall,
                       'F1_score' : f1score
                      })
```

In [47]:
```
result
```

Out[47]:

	ML Model	Accuracy	Precision	Recall	F1_score
0	BernoulliNB	0.633	0.920	0.633	0.732
1	PassiveAggressive	0.587	0.654	0.587	0.611
2	SGDClassifier	0.688	0.706	0.688	0.684
3	MLPClassifier	0.697	0.845	0.697	0.704
4	Ensemble	0.737	0.735	0.737	0.734
5	Extension	1.000	1.000	1.000	1.000

Modelling

In [48]:
```
import joblib
filename = 'models/model_sdn.sav'
joblib.dump(ext, filename)
```

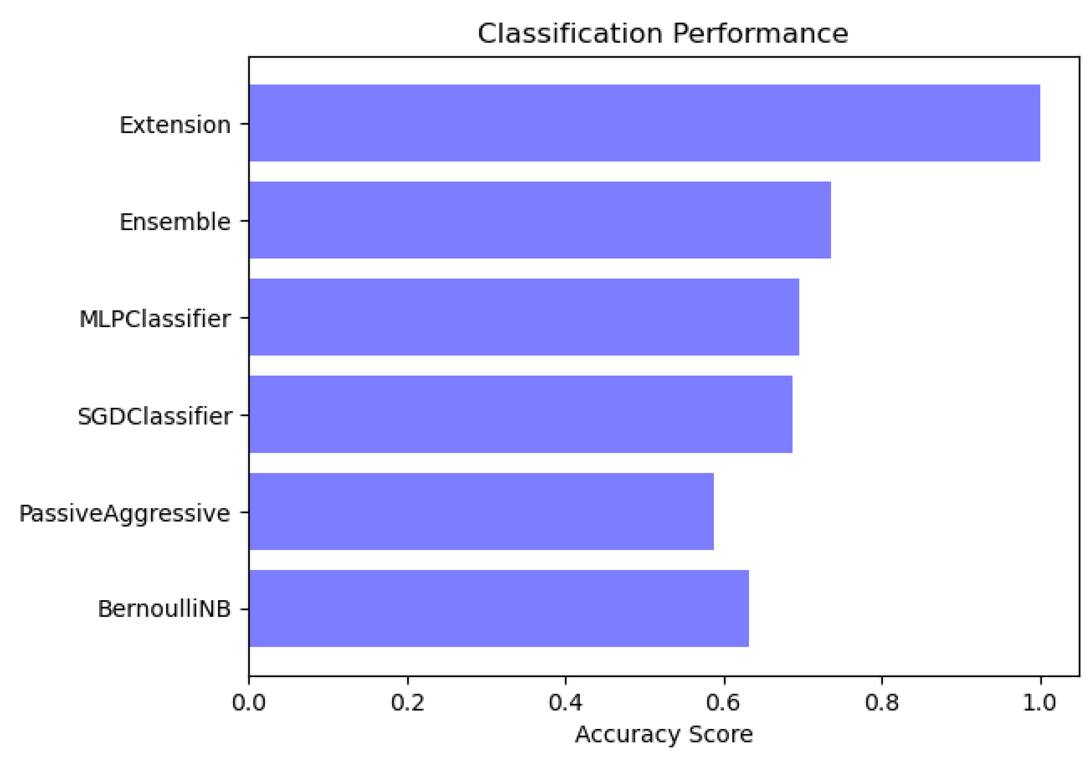
Out[48]: ['models/model_sdn.sav']

Graph

In [49]:
```
classifier = ML_Model
y_pos = np.arange(len(classifier))
```

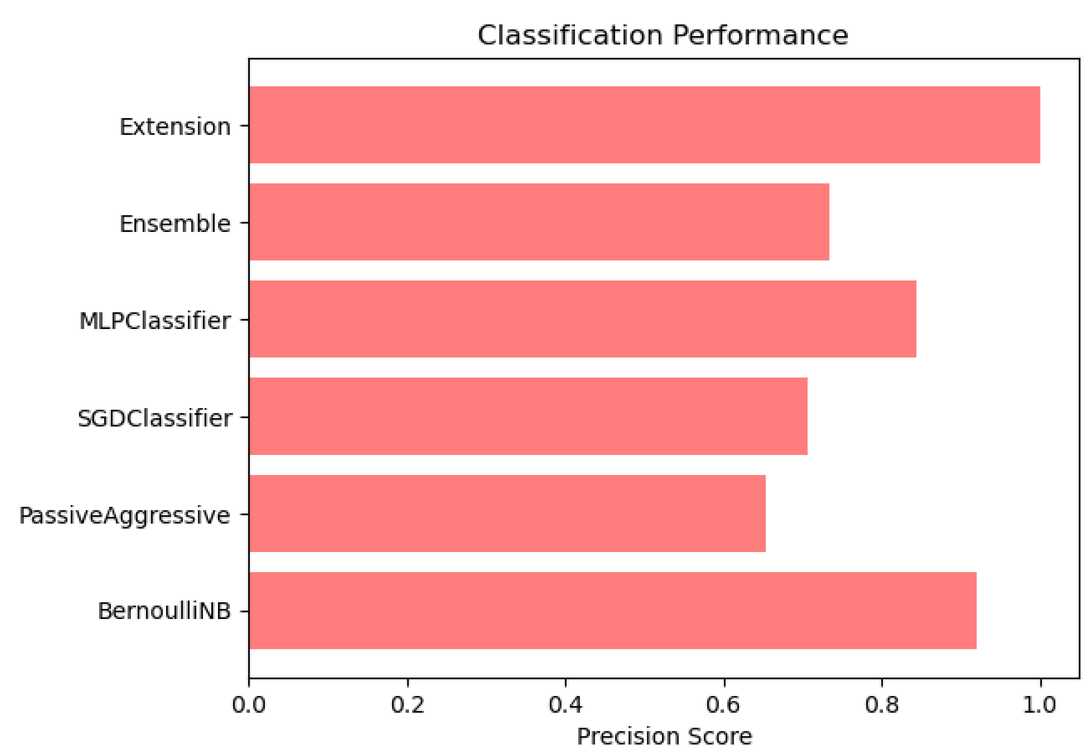
Accuracy

In [50]:
```
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()
```



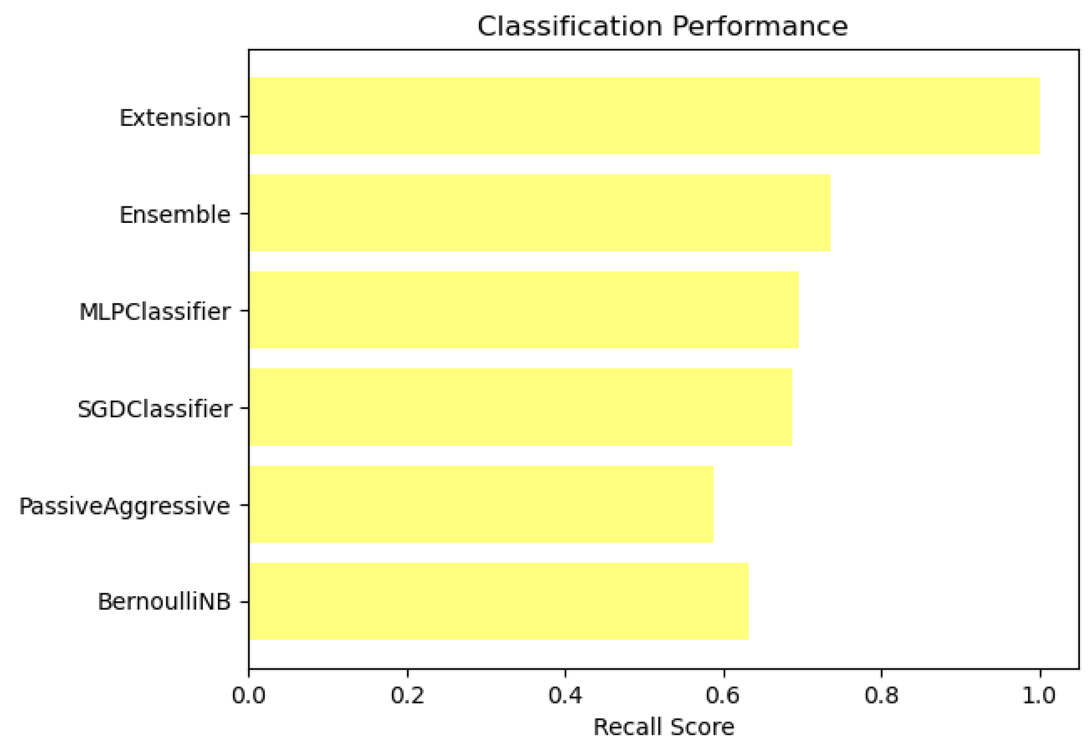
Precision

In [51]:
```
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()
```



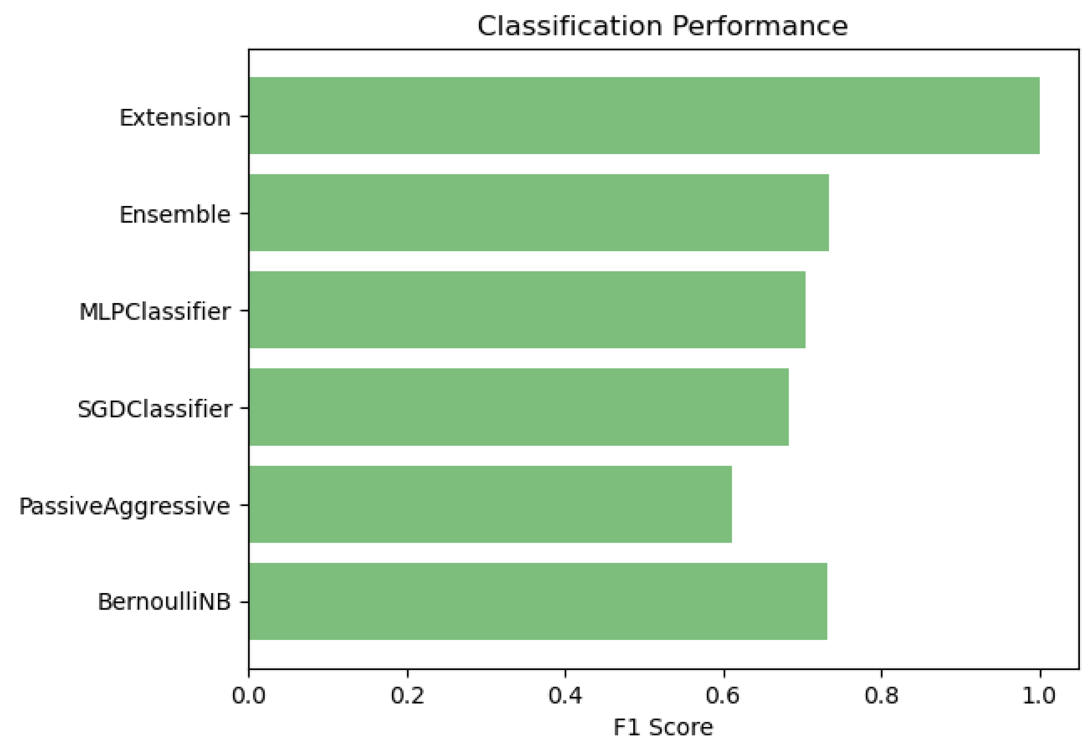
Recall

In [52]:
```
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 [53]:
```
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 [ ]: