Ishan Prabhune

A20538828

In [1]:
```
import numpy as npy
import scipy as spy
import pandas as pds
from IPython.display import display, HTML
import warnings

#Ignore warnings
warnings.filterwarnings('ignore')

# Loading the dataset
DataFrame=pds.read_csv('C:/Users/ishan/Downloads/malware_MultiClass.csv')

# Printing and Display the Dataframe in HTML
display(HTML(DataFrame.head(15).to_html()))

# Displaying the shape of the DataFrame
print("Shape of the DataFrame:", DataFrame.shape)
```

	hash	millisecond	class
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	0	
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	1	
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	2	
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	3	
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	4	
5	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	5	
6	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	6	
7	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	7	
8	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	8	
9	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	9	
10	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	10	
11	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	11	
12	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	12	
13	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	13	
14	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	14	

```
Shape of the DataFrame: (100000, 36)
```

In [2]:
```
# Selecting columns which are relevant for further analysis
SelectedColumns = ['classification', 'os', 'usage_counter', 'prio', 'static_prio

# Then we will use the SelectedColumns in the DataFrame
DataFrame = DataFrame[SelectedColumns]

# Striping the column names
DataFrame = DataFrame.rename(columns=lambda x: x.strip())

# Displaying rows of the DataFrame
display(HTML(DataFrame.head(10).to_html()))

# Checking for missing values in DataFrame
print('\nColumnName, DataType, MissingValues')
for col in DataFrame.columns:
    print(col, ',', DataFrame[col].dtype, ',', DataFrame[col].isnull().any())
```

	classification	os	usage_counter	prio	static_prio	normal_prio	vm_pgoff
0	malware	CentOS	0	3069378560	14274	0	0
1	malware	Windows	0	3069378560	14274	0	0
2	malware	Mac	0	3069378560	14274	0	0
3	malware	Ubuntu	0	3069378560	14274	0	0
4	malware	Mac	0	3069378560	14274	0	0
5	malware	Windows	0	3069378560	14274	0	0
6	malware	Ubuntu	0	3069378560	14274	0	0
7	malware	Mac	0	3069378560	14274	0	0
8	malware	CentOS	0	3069378560	14274	0	0
9	malware	Mac	0	3069378560	14274	0	0

```
ColumnName, DataType, MissingValues
classification , object , False
os , object , False
usage_counter , int64 , False
prio , int64 , False
static_prio , int64 , False
normal_prio , int64 , False
vm_pgoff , int64 , False
vm_truncate_count , int64 , False
task_size , int64 , False
map_count , int64 , False
hiwater_rss , int64 , False
total_vm , int64 , False
shared_vm , int64 , False
exec_vm , int64 , False
reserved_vm , int64 , False
nr_ptes , int64 , False
nvcsw , int64 , False
nivcsw , int64 , False
signal_nvcsw , int64 , False
```

In [3]:
```
# Encoding the labels
from sklearn import preprocessing

# Defining the label as nominal values
y = DataFrame['classification']
le = preprocessing.LabelEncoder()
le.fit(y)

# Encoding the nominal labels to integers
y_encoded = le.transform(y)
DataFrame['classification'] = y_encoded

# Printing and displaying dataframe as tables in HTML
display(HTML(DataFrame.head(10).to_html()))
```

	classification	os	usage_counter	prio	static_prio	normal_prio	vm_pgoff
0	1	CentOS	0	3069378560	14274	0	0
1	1	Windows	0	3069378560	14274	0	0
2	1	Mac	0	3069378560	14274	0	0
3	1	Ubuntu	0	3069378560	14274	0	0
4	1	Mac	0	3069378560	14274	0	0
5	1	Windows	0	3069378560	14274	0	0
6	1	Ubuntu	0	3069378560	14274	0	0
7	1	Mac	0	3069378560	14274	0	0
8	1	CentOS	0	3069378560	14274	0	0
9	1	Mac	0	3069378560	14274	0	0

In [4]:
```
# Data preprocessing
print('Column Datatypes:\n',DataFrame.dtypes)

# Converting all the nominal variables to binary variables
DataFrame_num=DataFrame.copy(deep=True)

# Creating the new binary columns
DataFrame_dummies=pds.get_dummies(DataFrame_num[['os']])

# Adding them to the dataframe
DataFrame_num=DataFrame_num.join(DataFrame_dummies)

# Droping the original columns
DataFrame_num=DataFrame_num.drop('os',axis=1)

# Dropping the extra binary columns, since we will only need N-1 binary columns
DataFrame_num=DataFrame_num.drop('os_Debian', axis=1)

display('DataFrame_num:',HTML(DataFrame_num.head(10).to_html()))
```

```
Column Datatypes:
 classification        int32
os                    object
usage_counter          int64
prio                   int64
static_prio            int64
normal_prio            int64
vm_pgoff               int64
vm_truncate_count      int64
task_size              int64
map_count              int64
hiwater_rss            int64
total_vm               int64
shared_vm              int64
exec_vm                int64
reserved_vm            int64
nr_ptes                int64
nvcsw                  int64
nivcsw                 int64
signal_nvcsw           int64
dtype: object
'DataFrame_num:'
```

	classification	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_trun
0	1	0	3069378560	14274	0	0	
1	1	0	3069378560	14274	0	0	
2	1	0	3069378560	14274	0	0	
3	1	0	3069378560	14274	0	0	
4	1	0	3069378560	14274	0	0	
5	1	0	3069378560	14274	0	0	
6	1	0	3069378560	14274	0	0	
7	1	0	3069378560	14274	0	0	
8	1	0	3069378560	14274	0	0	
9	1	0	3069378560	14274	0	0	

In [5]:
```
# Standarizing the data
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
DataFrame_num_std = DataFrame_num.copy(deep=True)
x_features = DataFrame_num_std.loc[:, DataFrame_num_std.columns != 'classificati
cols = x_features.columns
DataFrame_num_std = pds.DataFrame(scaler.fit_transform(x_features), columns = cc

DataFrame_num_std['classification'] = y_encoded
display('DataFrame_num_std:',HTML(DataFrame_num_std.head(10).to_html()))
```

```
'DataFrame_num_std:'
```

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task
0	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
1	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
2	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
3	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
4	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
5	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
6	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
7	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
8	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
9	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	

In [6]:
```
# Binary features

DataFrame_binary = DataFrame_num.copy(deep=True)
numCols = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
DataFrame_numerical = DataFrame_binary.iloc[:,numCols]
DataFrame_dummy = DataFrame_binary.drop(DataFrame_binary.columns[numCols], axis=
display('DataFrame_numerical:',HTML(DataFrame_numerical.head(10).to_html()))
display('DataFrame_dummy:',HTML(DataFrame_dummy.head(10).to_html()))
```

'DataFrame_numerical:'

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	ta
0	0	3069378560	14274	0	0	13173	
1	0	3069378560	14274	0	0	13173	
2	0	3069378560	14274	0	0	13173	
3	0	3069378560	14274	0	0	13173	
4	0	3069378560	14274	0	0	13173	
5	0	3069378560	14274	0	0	13173	
6	0	3069378560	14274	0	0	13173	
7	0	3069378560	14274	0	0	13173	
8	0	3069378560	14274	0	0	13173	
9	0	3069378560	14274	0	0	13173	

'DataFrame_dummy:'

	classification	os_CentOS	os_Mac	os_Ubuntu	os_Windows
0	1	True	False	False	False
1	1	False	False	False	True
2	1	False	True	False	False
3	1	False	False	True	False
4	1	False	True	False	False
5	1	False	False	False	True
6	1	False	False	True	False
7	1	False	True	False	False
8	1	True	False	False	False
9	1	False	True	False	False

In [7]:
```
# Grouping the Data into High, Medium and Low groups

GroupNames = ['L','M','H']
for col in DataFrame_numerical.columns:
    DataFrame_numerical[col] = pds.cut(DataFrame_numerical[col], 3, labels=Group
display('DataFrame_numerical:',HTML(DataFrame_numerical.head(10).to_html()))
```

'DataFrame_numerical:'

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_size
0	M	L	L	M	M	L	M
1	M	L	L	M	M	L	M
2	M	L	L	M	M	L	M
3	M	L	L	M	M	L	M
4	M	L	L	M	M	L	M
5	M	L	L	M	M	L	M
6	M	L	L	M	M	L	M
7	M	L	L	M	M	L	M
8	M	L	L	M	M	L	M
9	M	L	L	M	M	L	M

In [8]:
```
# DataFrame for the Dummies

DataFrame_dummies=pds.get_dummies(DataFrame_numerical)
display('DataFrame_dummies:',HTML(DataFrame_dummies.head(10).to_html()))

cols_removed = ['usage_counter_L','normal_prio_L','task_size_L','total_vm_L','re

DataFrame_dummies = DataFrame_dummies.drop(cols_removed, axis=1)
DataFrame_dummies = DataFrame_dummies.astype(int)
```

```
display('DataFrame_dummies without L:',HTML(DataFrame_numerical.head(10).to_html

DataFrame_binary = DataFrame_binary.astype(int)
display('DataFrame_binary:',HTML(DataFrame_binary.head(10).to_html()))
```

'DataFrame_dummies:'

	usage_counter_L	usage_counter_M	usage_counter_H	prio_L	prio_M	prio_H	static_pri
0	False	True	False	True	False	False	
1	False	True	False	True	False	False	
2	False	True	False	True	False	False	
3	False	True	False	True	False	False	
4	False	True	False	True	False	False	
5	False	True	False	True	False	False	
6	False	True	False	True	False	False	
7	False	True	False	True	False	False	
8	False	True	False	True	False	False	
9	False	True	False	True	False	False	

'DataFrame_dummies without L:'

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_size
0	M	L	L	M	M	L	M
1	M	L	L	M	M	L	M
2	M	L	L	M	M	L	M
3	M	L	L	M	M	L	M
4	M	L	L	M	M	L	M
5	M	L	L	M	M	L	M
6	M	L	L	M	M	L	M
7	M	L	L	M	M	L	M
8	M	L	L	M	M	L	M
9	M	L	L	M	M	L	M

'DataFrame_binary:'

	classification	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_tru
0	1	0	-1225588736	14274	0	0	
1	1	0	-1225588736	14274	0	0	
2	1	0	-1225588736	14274	0	0	
3	1	0	-1225588736	14274	0	0	
4	1	0	-1225588736	14274	0	0	
5	1	0	-1225588736	14274	0	0	
6	1	0	-1225588736	14274	0	0	
7	1	0	-1225588736	14274	0	0	
8	1	0	-1225588736	14274	0	0	
9	1	0	-1225588736	14274	0	0	

In [9]:
```
# Merging the Binary and Dummies dataframes

DataFrame_binary = pds.concat([DataFrame_dummies, DataFrame_dummy], axis=1)
display('DataFrame_binary:',HTML(DataFrame_binary.head(10).to_html()))

# Defining the x and y values

y = DataFrame_binary['classification']
x = DataFrame_binary.drop('classification', axis=1)
```

'DataFrame_binary:'

	usage_counter_M	usage_counter_H	prio_M	prio_H	static_prio_M	static_prio_H	norm
0	1	0	0	0	0	0	
1	1	0	0	0	0	0	
2	1	0	0	0	0	0	
3	1	0	0	0	0	0	
4	1	0	0	0	0	0	
5	1	0	0	0	0	0	
6	1	0	0	0	0	0	
7	1	0	0	0	0	0	
8	1	0	0	0	0	0	
9	1	0	0	0	0	0	

In [10]:
```
# CategoricalNB

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import CategoricalNB
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sc
```

```
# Splitting the Dataset into training and testing sets taking 80% for training a
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

# Initialising the Classifier
clfn = CategoricalNB(alpha = 1)

# Training the classifier
clfn.fit(x_train, y_train)

# Making predictions on the testing set
y_prediction = clfn.predict(x_test)

# Calculating the evaluation metrics
accu = accuracy_score(y_test, y_prediction)
mic_prec = precision_score(y_test, y_prediction, average='micro')
mic_rec = recall_score(y_test, y_prediction, average='micro')
mic_f1 = f1_score(y_test, y_prediction, average='micro')

# Positive class probability estimates
y_scores = clfn.predict_proba(x_test)
mic_auc_cat = roc_auc_score(y_test, clfn.predict_proba(x_test), average='micro',

# Printing out all the metrics
print("Hold-out Validation for CategoricalNB with 80% as training and 20% as tes
print(f"Accuracy = {accu}")
print(f"Micro-precision = {mic_prec}")
print(f"Micro-recall = {mic_rec}")
print(f"Micro-F1 = {mic_f1}")
print(f"Micro-AUC = {mic_auc_cat}")
```

```
Hold-out Validation for CategoricalNB with 80% as training and 20% as testing dat
a:
Accuracy = 0.7522
Micro-precision = 0.7522
Micro-recall = 0.7522
Micro-F1 = 0.7522
Micro-AUC = 0.91044017875
```

In [11]:

```
# CategoricalNB (with Hyperparameters added)

from sklearn.naive_bayes import ComplementNB, MultinomialNB, GaussianNB, Categor
from sklearn.metrics import make_scorer, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split

# Splitting the Dataset into training and testing sets taking 80% for training a
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

# Initialising the Classifier and adding the hyper parameters
clfn = CategoricalNB(alpha=0.5, force_alpha=True)

# Training the classifier
clfn.fit(x_train, y_train)

# Making predictions on the testing set
y_prediction = clfn.predict(x_test)

# Calculating the evaluation metrics
accu = accuracy_score(y_test, y_prediction)
mic_prec = precision_score(y_test, y_prediction, average='micro')
```

```
mic_rec = recall_score(y_test, y_prediction, average='micro')
mic_f1 = f1_score(y_test, y_prediction, average='micro')

# Positive class probability estimates
y_scores = clfn.predict_proba(x_test)
mic_auc_cat = roc_auc_score(y_test, clfn.predict_proba(x_test), average='micro',

# Printing out all the metrics
print("Hold-out Validation for CategoricalNB (with Hyperparameters added) with 8
print(f"Accuracy = {accu}")
print(f"Micro-precision = {mic_prec}")
print(f"Micro-recall = {mic_rec}")
print(f"Micro-F1 = {mic_f1}")
print(f"Micro-AUC = {mic_auc_cat}")
```

```
Hold-out Validation for CategoricalNB (with Hyperparameters added) with 80% as tr
aining and 20% as testing data:
Accuracy = 0.7522
Micro-precision = 0.7522
Micro-recall = 0.7522
Micro-F1 = 0.7522
Micro-AUC = 0.9103671275
```

In [12]:

```
# GaussianNB

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sc

# Splitting the Dataset into training and testing sets taking 80% for training a
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

# Initialising the Classifier
clfn = GaussianNB()

# Training the classifier
clfn.fit(x_train, y_train)

# Making predictions on the testing set
y_prediction = clfn.predict(x_test)

# Calculating the evaluation metrics
accu = accuracy_score(y_test, y_prediction)
mic_prec_gauss = precision_score(y_test, y_prediction, average='micro')
mic_rec_gauss = recall_score(y_test, y_prediction, average='micro')
mic_f1_gauss = f1_score(y_test, y_prediction, average='micro')

# Positive class probability estimates
y_scores = clfn.predict_proba(x_test)
mic_auc_gauss = roc_auc_score(y_test, clfn.predict_proba(x_test), average='micro

# Printing out all the metrics
print("Hold-out Validation for GaussianNB with 80% as training and 20% as testin
print(f"Accuracy = {accu}")
print(f"Micro-precision = {mic_prec_gauss}")
print(f"Micro-recall = {mic_rec_gauss}")
print(f"Micro-F1 = {mic_f1_gauss}")
print(f"Micro-AUC = {mic_auc_gauss}")
```

```
Hold-out Validation for GaussianNB with 80% as training and 20% as testing data:
Accuracy = 0.16755
Micro-precision = 0.16755
Micro-recall = 0.16755
Micro-F1 = 0.16755
Micro-AUC = 0.443720005
```

In [13]:
```python
# GaussianNB (with HyperParameters)

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sc

# Splitting the Dataset into training and testing sets taking 80% for training a
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

# Initialising the Classifier and adding the hyper parameters
clfn = GaussianNB(priors=None, var_smoothing=0.5)

# Training the classifier
clfn.fit(x_train, y_train)

# Making predictions on the testing set
y_prediction = clfn.predict(x_test)

# Calculating the evaluation metrics
accu = accuracy_score(y_test, y_prediction)
mic_prec_gauss = precision_score(y_test, y_prediction, average='micro')
mic_rec_gauss = recall_score(y_test, y_prediction, average='micro')
mic_f1_gauss = f1_score(y_test, y_prediction, average='micro')

# Positive class probability estimates
y_scores = clfn.predict_proba(x_test)
mic_auc_gauss = roc_auc_score(y_test, clfn.predict_proba(x_test), average='micro

# Printing out all the metrics
print("Hold-out Validation for GaussianNB (with Hyperparameters added) with 80%
print(f"Accuracy = {accu}")
print(f"Micro-precision = {mic_prec_gauss}")
print(f"Micro-recall = {mic_rec_gauss}")
print(f"Micro-F1 = {mic_f1_gauss}")
print(f"Micro-AUC = {mic_auc_gauss}")
```

```
Hold-out Validation for GaussianNB (with Hyperparameters added) with 80% as train
ing and 20% as testing data:
Accuracy = 0.6268
Micro-precision = 0.6268
Micro-recall = 0.6268
Micro-F1 = 0.6268
Micro-AUC = 0.8687441925
```

In [14]:
```python
# BernoulliNB

from sklearn.naive_bayes import BernoulliNB

# Splitting the Dataset into training and testing sets taking 80% for training a
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

# Initialising the Classifier
clfn = BernoulliNB(alpha = 1)
```

```
# Training the classifier
clfn.fit(x_train, y_train)

y_prediction = clfn.predict(x_test)

# Calculating the evaluation metrics
accu = accuracy_score(y_test, y_prediction)
mic_prec_bern = precision_score(y_test, y_prediction, average='micro')
mic_rec_bern = recall_score(y_test, y_prediction, average='micro')
mic_f1_bern = f1_score(y_test, y_prediction, average='micro')

# Positive class probability estimates
y_scores = clfn.predict_proba(x_test)
mic_auc_bern = roc_auc_score(y_test, clfn.predict_proba(x_test), average='micro'

# Printing out all the metrics
print("Hold-out Validation for BernoulliNB with 80% as training and 20% as testi
print(f"Accuracy = {accu}")
print(f"Micro-precision = {mic_prec_bern}")
print(f"Micro-recall = {mic_rec_bern}")
print(f"Micro-F1 = {mic_f1_bern}")
print(f"Micro-AUC = {mic_auc_bern}")
```

```
Hold-out Validation for BernoulliNB with 80% as training and 20% as testing data:
Accuracy = 0.75215
Micro-precision = 0.75215
Micro-recall = 0.75215
Micro-F1 = 0.7521500000000001
Micro-AUC = 0.9115201424999999
```

In [15]:

```
# BernoulliNB (with HyperParameters)

from sklearn.naive_bayes import BernoulliNB

# Splitting the Dataset into training and testing sets taking 80% for training a
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

# Initialising the Classifier and adding the hyper parameters
clfn = CategoricalNB(alpha=0.5, force_alpha=True)

# Training the classifier
clfn.fit(x_train, y_train)

y_prediction = clfn.predict(x_test)

# Calculating the evaluation metrics
accu = accuracy_score(y_test, y_prediction)
mic_prec_bern = precision_score(y_test, y_prediction, average='micro')
mic_rec_bern = recall_score(y_test, y_prediction, average='micro')
mic_f1_bern = f1_score(y_test, y_prediction, average='micro')

# Positive class probability estimates
y_scores = clfn.predict_proba(x_test)
mic_auc_bern = roc_auc_score(y_test, clfn.predict_proba(x_test), average='micro'

# Printing out all the metrics
print("Hold-out Validation for BernoulliNB (with Hyperparameters added) with 80%
print(f"Accuracy = {accu}")
print(f"Micro-precision = {mic_prec_bern}")
```

```
print(f"Micro-recall = {mic_rec_bern}")
print(f"Micro-F1 = {mic_f1_bern}")
print(f"Micro-AUC = {mic_auc_bern}")
```

```
Hold-out Validation for BernoulliNB (with Hyperparameters added) with 80% as trai
ning and 20% as testing data:
Accuracy = 0.75215
Micro-precision = 0.75215
Micro-recall = 0.75215
Micro-F1 = 0.7521500000000001
Micro-AUC = 0.9115231725000001
```

In [16]:
```
# MultinomialNB

from sklearn.naive_bayes import MultinomialNB

# Splitting the Dataset into training and testing sets taking 80% for training a
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

# Initialising the Classifier
clfn = MultinomialNB(alpha = 1)

# Training the classifier
clfn.fit(x_train, y_train)

y_prediction = clfn.predict(x_test)

# Calculating the evaluation metrics
accu = accuracy_score(y_test, y_prediction)
mic_prec_multi = precision_score(y_test, y_prediction, average='micro')
mic_rec_multi = recall_score(y_test, y_prediction, average='micro')
mic_f1_multi = f1_score(y_test, y_prediction, average='micro')

# Positive class probability estimates
y_scores = clfn.predict_proba(x_test)
mic_auc_multi = roc_auc_score(y_test, clfn.predict_proba(x_test), average='micro

# Printing out all the metrics
print("Hold-out Validation for MultinomialNB with 80% as training and 20% as tes
print(f"Accuracy = {accu}")
print(f"Micro-precision = {mic_prec_multi}")
print(f"Micro-recall = {mic_rec_multi}")
print(f"Micro-F1 = {mic_f1_multi}")
print(f"Micro-AUC = {mic_auc_multi}")
```

```
Hold-out Validation for MultinomialNB with 80% as training and 20% as testing dat
a:
Accuracy = 0.70335
Micro-precision = 0.70335
Micro-recall = 0.70335
Micro-F1 = 0.70335
Micro-AUC = 0.8808239625000001
```

In [17]:
```
# MultinomialNB (with HyperParameters)

from sklearn.naive_bayes import MultinomialNB

# Splitting the Dataset into training and testing sets taking 80% for training a
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

# Initialising the Classifier and adding the hyper parameters
```

```
clfn = MultinomialNB(alpha=0.5, force_alpha=True)

# Training the classifier
clfn.fit(x_train, y_train)

y_prediction = clfn.predict(x_test)

# Calculating the evaluation metrics
accu = accuracy_score(y_test, y_prediction)
mic_prec_multi = precision_score(y_test, y_prediction, average='micro')
mic_rec_multi = recall_score(y_test, y_prediction, average='micro')
mic_f1_multi = f1_score(y_test, y_prediction, average='micro')

# Positive class probability estimates
y_scores = clfn.predict_proba(x_test)
mic_auc_multi = roc_auc_score(y_test, clfn.predict_proba(x_test), average='micro

# Printing out all the metrics
print("Hold-out Validation for MultinomialNB (with Hyperparameters added) with 8
print(f"Accuracy = {accu}")
print(f"Micro-precision = {mic_prec_multi}")
print(f"Micro-recall = {mic_rec_multi}")
print(f"Micro-F1 = {mic_f1_multi}")
print(f"Micro-AUC = {mic_auc_multi}")
```

```
Hold-out Validation for MultinomialNB (with Hyperparameters added) with 80% as tr
aining and 20% as testing data:
Accuracy = 0.70335
Micro-precision = 0.70335
Micro-recall = 0.70335
Micro-F1 = 0.70335
Micro-AUC = 0.8808478612499999
```

In [18]:

```
# ComplementNB

from sklearn.naive_bayes import ComplementNB

# Splitting the Dataset into training and testing sets taking 80% for training a
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

# Initialising the Classifier
clfn = ComplementNB(alpha = 1)

# Training the classifier
clfn.fit(x_train, y_train)

y_prediction = clfn.predict(x_test)

# Calculating the evaluation metrics
accu = accuracy_score(y_test, y_prediction)
mic_prec_comp = precision_score(y_test, y_prediction, average='micro')
mic_rec_comp = recall_score(y_test, y_prediction, average='micro')
mic_f1_comp = f1_score(y_test, y_prediction, average='micro')

# Positive class probability estimates
y_scores = clfn.predict_proba(x_test)
mic_auc_comp = roc_auc_score(y_test, clfn.predict_proba(x_test), average='micro'

# Printing out all the metrics
print("Hold-out Validation for ComplementNB with 80% as training and 20% as test
```

```
print(f"accuracy = {accu}")
print(f"Micro-precision = {mic_prec_comp}")
print(f"Micro-recall = {mic_rec_comp}")
print(f"Micro-F1 = {mic_f1_comp}")
print(f"Micro-AUC = {mic_auc_comp}")
```

```
Hold-out Validation for ComplementNB with 80% as training and 20% as testing dat
a:
accuracy = 0.7023
Micro-precision = 0.7023
Micro-recall = 0.7023
Micro-F1 = 0.7023
Micro-AUC = 0.77812922
```

In [19]:
```
# ComplementNB (with HyperParameters)

from sklearn.naive_bayes import ComplementNB

# Splitting the Dataset into training and testing sets taking 80% for training a
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

# Initialising the Classifier and adding the hyper parameters
clfn = ComplementNB(alpha=0.5, force_alpha=True)

# Training the classifier
clfn.fit(x_train, y_train)

y_prediction = clfn.predict(x_test)

# Calculating the evaluation metrics
accu = accuracy_score(y_test, y_prediction)
mic_prec_comp = precision_score(y_test, y_prediction, average='micro')
mic_rec_comp = recall_score(y_test, y_prediction, average='micro')
mic_f1_comp = f1_score(y_test, y_prediction, average='micro')

# Positive class probability estimates
y_scores = clfn.predict_proba(x_test)
mic_auc_comp = roc_auc_score(y_test, clfn.predict_proba(x_test), average='micro'

# Printing out all the metrics
print("Hold-out Validation for ComplementNB (with Hyperparameters added) with 80
print(f"accuracy = {accu}")
print(f"Micro-precision = {mic_prec_comp}")
print(f"Micro-recall = {mic_rec_comp}")
print(f"Micro-F1 = {mic_f1_comp}")
print(f"Micro-AUC = {mic_auc_comp}")
```

```
Hold-out Validation for ComplementNB (with Hyperparameters added) with 80% as tra
ining and 20% as testing data:
accuracy = 0.7023
Micro-precision = 0.7023
Micro-recall = 0.7023
Micro-F1 = 0.7023
Micro-AUC = 0.77811888125
```

In [20]:
```
print ('Conclusion')

print ('From the models given above, the CategoricalNB and BernoulliNB have cons

print ('CategoricalNB:')
print ('Accuracy: 0.7522 , Micro-precision: 0.7522, Micro-recall: 0.7522, Micro-
```

```
print ('CategoricalNB (with Hyperparameters added :')
print ('Accuracy: 0.7522, Micro-precision: 0.7522, Micro-recall: 0.7522, Micro-F

print ('BernoulliNB:')
print ('Accuracy: 0.7521, Micro-precision: 0.7522, Micro-recall: 0.7522, Micro-F

print ('BernoulliNB (with Hyperparameters added):')
print ('Accuracy: 0.7521, Micro-precision: 0.7522, Micro-recall: 0.7522, Micro-F

print ('The GaussianNB has shown the lowest performance accross all the metrics,

print ('The MultinomialNB and ComplementNB have showed moderate performance.')

print ('Overall, CategoricalNB and BernoulliNB are the preferred choices for thi
```

```
Conclusion
From the models given above, the CategoricalNB and BernoulliNB have consistently
achieved the highest accuracy, micro-precision, micro-recall, micro-f1score and m
icro-AUC values.
CategoricalNB:
Accuracy: 0.7522 , Micro-precision: 0.7522, Micro-recall: 0.7522, Micro-F1: 0.752
2, Micro-AUC: 0.9104
CategoricalNB (with Hyperparameters added :
Accuracy: 0.7522, Micro-precision: 0.7522, Micro-recall: 0.7522, Micro-F1: 0.752
2, Micro-AUC: 0.9103
BernoulliNB:
Accuracy: 0.7521, Micro-precision: 0.7522, Micro-recall: 0.7522, Micro-F1: 0.752
2, Micro-AUC: 0.9115
BernoulliNB (with Hyperparameters added):
Accuracy: 0.7521, Micro-precision: 0.7522, Micro-recall: 0.7522, Micro-F1: 0.752
2, Micro-AUC: 0.9115
The GaussianNB has shown the lowest performance accross all the metrics, which te
lls us it is not suitable to use in this dataset.
The MultinomialNB and ComplementNB have showed moderate performance.
Overall, CategoricalNB and BernoulliNB are the preferred choices for this task, d
ue to their high performance metrics.
```

In [ ]: