ML LAB ASSIGNMENT 8

NAIVE BAYES ALGORITHM IMPLEMENTATION

Name: Madhumithaa RP | Reg No: 20BCE1648

Implementing the necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv(r"C:\Users\abhia\Desktop\VIT SEM6\ML\Lab\Lab3\
final.csv")
data.head(10)
```

```
   Destination Port  Flow Duration  Total Fwd Packets  Total Backward
Packets  \
0                80       99077735                  7
5
1              1010             42                  1
1
2             15003             43                  1
1
3              8080          82322                 26
34
4                80       84813161                  6
6
5              9090             71                  1
1
6                80        3000585                  3
0
7              8080          60469                  4
3
8                21        9182361                  9
15
9                22            106                  1
1

   Total Length of Fwd Packets  Total Length of Bwd Packets  \
0                          373                        11595
1                            0                            6
2                            2                            6
3                        70312                          320
4                          355                        11595
5                            2                            6
6                            0                            0
```

```
7                          207                          134
8                          105                          188
9                            0                            0

   Fwd Packet Length Max  Fwd Packet Length Min  Fwd Packet Length
Mean  \
0                    367                      0
53.285714
1                      0                      0
0.000000
2                      2                      2
2.000000
3                   6060                      0
2704.307692
4                    355                      0
59.166667
5                      2                      2
2.000000
6                      0                      0
0.000000
7                    195                      0
51.750000
8                     23                      0
11.666667
9                      0                      0
0.000000

   Fwd Packet Length Std  ...  min_seg_size_forward  Active Mean
Active Std  \
0              138.353068  ...                    20       1036.0
0.0
1                0.000000  ...                    40          0.0
0.0
2                0.000000  ...                    24          0.0
0.0
3             1652.450175  ...                    20          0.0
0.0
4              144.928143  ...                    32          4.0
0.0
5                0.000000  ...                    24          0.0
0.0
6                0.000000  ...                    40          0.0
0.0
7               95.541876  ...                    20          0.0
0.0
8                9.300538  ...                    32          0.0
0.0
9                0.000000  ...                    32          0.0
0.0
```

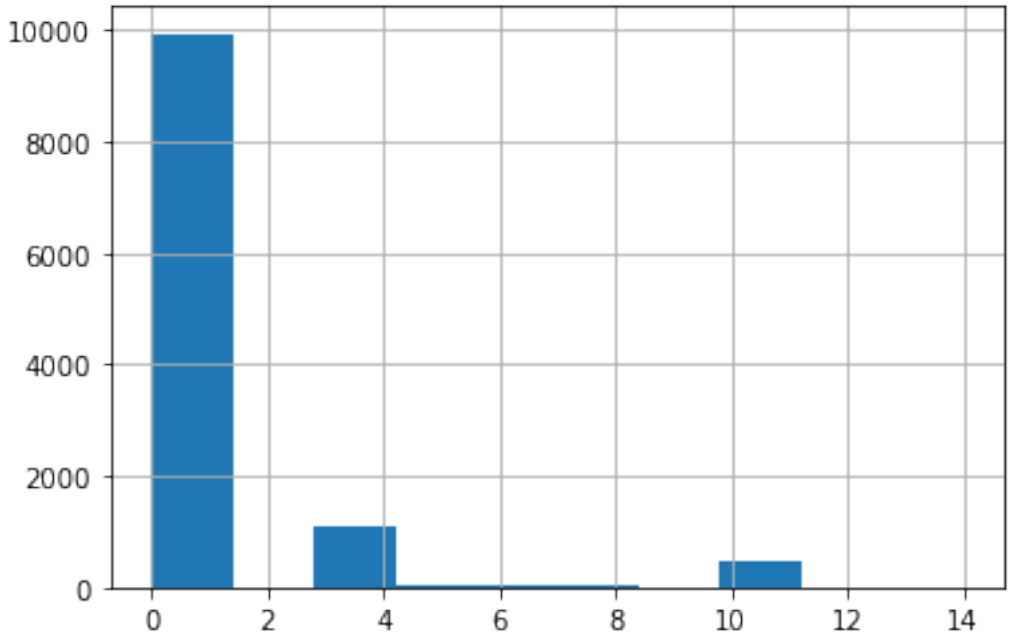
Label	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min
0	1036	1036	98900000.0	0.0	98900000	98900000
1	0	0	0.0	0.0	0	0
2	0	0	0.0	0.0	0	0
3	0	0	0.0	0.0	0	0
4	4	4	84700000.0	0.0	84700000	84700000
5	0	0	0.0	0.0	0	0
6	0	0	0.0	0.0	0	0
7	0	0	0.0	0.0	0	0
8	0	0	0.0	0.0	0	0
9	0	0	0.0	0.0	0	0

[10 rows x 79 columns]

Mapping the attributes and their correlation

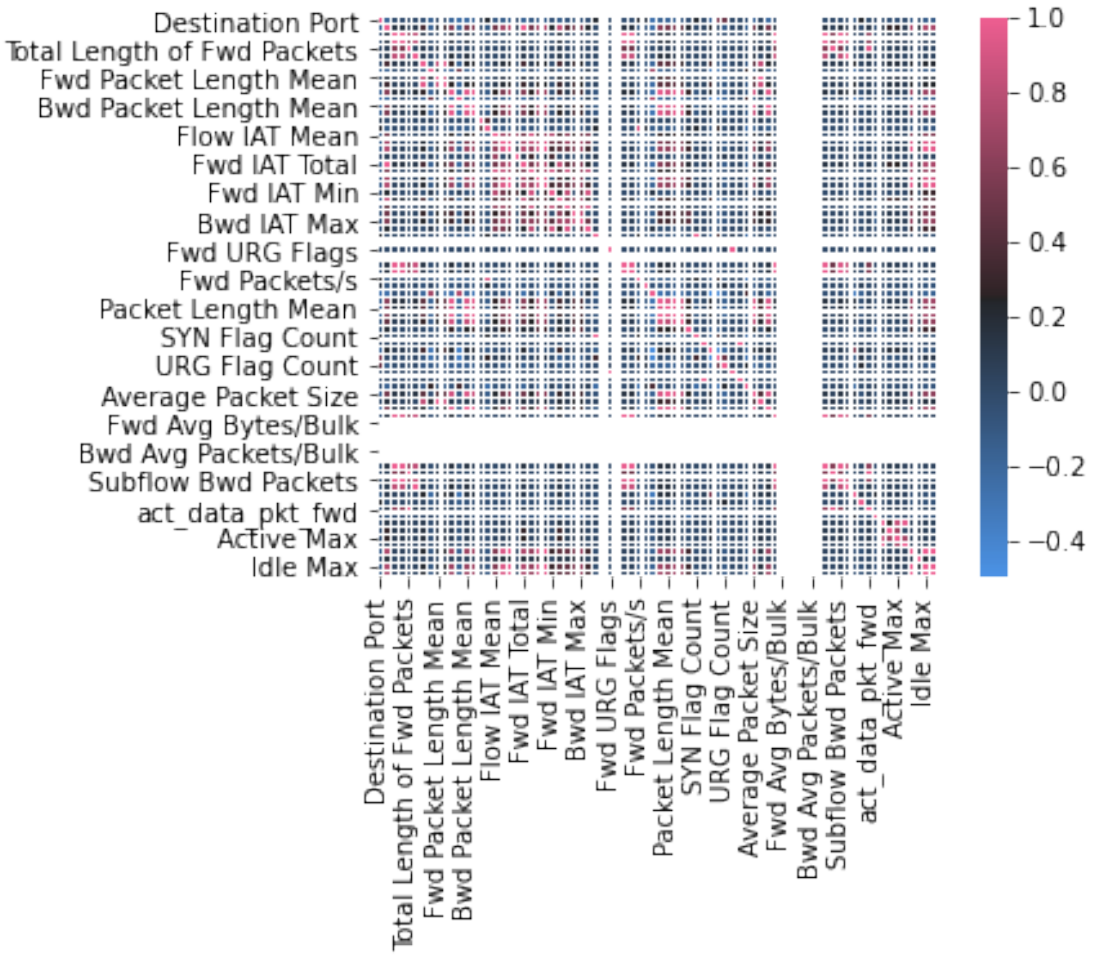
```
data["Label"].hist()
```

```
<AxesSubplot:>
```



```
corr = data.iloc[:,:-1].corr(method="pearson")
cmap = sns.diverging_palette(250,354,80,60,center='dark',as_cmap=True)
sns.heatmap(corr, vmax=1, vmin=-.5, cmap=cmap, square=True,
linewidths=.2)
```

```
<AxesSubplot:>
```




```
data = data[["Destination Port", "SYN Flag Count", "Fwd Avg
Bytes/Bulk", "Bwd Avg Packets/Bulk", "Label"]]
data.head(10)
```

```
   Destination Port  SYN Flag Count  Fwd Avg Bytes/Bulk  Bwd Avg
Packets/Bulk  \
0                80               0                   0
0
1              1010               0                   0
0
2             15003               0                   0
0
3              8080               0                   0
0
```

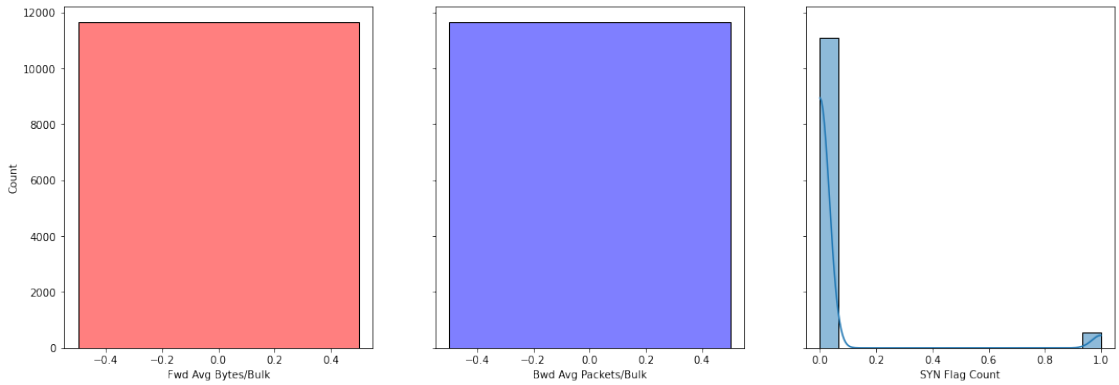
```
4                80              0                   0
0
5              9090              0                   0
0
6                80              0                   0
0
7              8080              0                   0
0
8                21              0                   0
0
9                22              0                   0
0
```

```
   Label
0      4
1     10
2     10
3      1
4      4
5     10
6      6
7      1
8      7
9     11
```

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
sns.histplot(data, ax=axes[0], x="Fwd Avg Bytes/Bulk", kde=True,
color='r')
sns.histplot(data, ax=axes[1], x="Bwd Avg Packets/Bulk", kde=True,
color='b')
sns.histplot(data, ax=axes[2], x="SYN Flag Count", kde=True)
```

```
C:\Users\abhia\anaconda3\lib\site-packages\seaborn\
distributions.py:306: UserWarning: Dataset has 0 variance; skipping
density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\abhia\anaconda3\lib\site-packages\seaborn\
distributions.py:306: UserWarning: Dataset has 0 variance; skipping
density estimate.
  warnings.warn(msg, UserWarning)
```

```
<AxesSubplot:xlabel='SYN Flag Count', ylabel='Count'>
```



Calculate P(Y=y) for all possible y

```
def calculate_prior(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior
```

Approach 1: Calculate P(X=x|Y=y) using Gaussian distribution formula

```
def calculate_likelihood_gaussian(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    mean, std = df[feat_name].mean(), df[feat_name].std()
    p_x_given_y = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-
((feat_val-mean)**2 / (2 * std**2 )))
    return p_x_given_y
```

Calculate P(X=x1|Y=y)P(X=x2|Y=y)...P(X=xn|Y=y) * P(Y=y) for all y and find the maximum

```
def naive_bayes_gaussian(df, X, Y):
    # get feature names
    features = list(df.columns)[:-1]

    # calculate prior
    prior = calculate_prior(df, Y)

    Y_pred = []
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood_gaussian(df,
features[i], x[i], Y, labels[j])
```

```
        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

Testing this Gaussian model

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size=.2, random_state=41)

X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values
Y_pred = naive_bayes_gaussian(train, X=X_test, Y="Label")

from sklearn.metrics import confusion_matrix, f1_score
print(confusion_matrix(Y_test, Y_pred))
print(f1_score(Y_test, Y_pred))
```

Convert continuous features to Categorical features

```
data["Con_Fwd Avg Bytes/Bulk"] = pd.cut(data["Fwd Avg
Bytes/Bulk"].values, bins = 3, labels = [0,1,2])
data["Con_Bwd Avg Packets/Bulk"] = pd.cut(data["Bwd Avg
Packets/Bulk"].values, bins = 3, labels = [0,1,2])

data = data[["Con_Fwd Avg Bytes/Bulk","Con_Bwd Avg Packets/Bulk",
"Label"]]
data.head(10)
```

	Con_Fwd Avg Bytes/Bulk	Con_Bwd Avg Packets/Bulk	Label
0	1	1	4
1	1	1	10
2	1	1	10
3	1	1	1
4	1	1	4
5	1	1	10
6	1	1	6
7	1	1	1
8	1	1	7
9	1	1	11

Approach 2: Calculate P(X=x|Y=y) categorically

```
def calculate_likelihood_categorical(df, feat_name, feat_val, Y,
label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    p_x_given_y = len(df[df[feat_name]==feat_val]) / len(df)
    return p_x_given_y
```

Calculate P(X=x1|Y=y)P(X=x2|Y=y)...P(X=xn|Y=y) * P(Y=y) for all y and find the maximum

```
def naive_bayes_categorical(df, X, Y):
    features = list(df.columns)[:-1]

    prior = calculate_prior(df, Y)

    Y_pred = []
    for x in X:
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood_categorical(df,
features[i], x[i], Y, labels[j])

        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

Test Categorical model

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size=.2, random_state=41)

X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values
Y_pred = naive_bayes_categorical(train, X=X_test, Y="Label")

from sklearn.metrics import confusion_matrix, f1_score
print(confusion_matrix(Y_test, Y_pred))
print(f1_score(Y_test, Y_pred))
```

```
[[1987    0    0    0    0    0    0    0    0    0    0    0]
 [   1    0    0    0    0    0    0    0    0    0    0    0]
 [   1    0    0    0    0    0    0    0    0    0    0    0]
 [  12    0    0    0    0    0    0    0    0    0    0    0]
 [ 205    0    0    0    0    0    0    0    0    0    0    0]
 [   5    0    0    0    0    0    0    0    0    0    0    0]
 [   4    0    0    0    0    0    0    0    0    0    0    0]
 [   4    0    0    0    0    0    0    0    0    0    0    0]
 [ 103    0    0    0    0    0    0    0    0    0    0    0]
 [   3    0    0    0    0    0    0    0    0    0    0    0]
 [   2    0    0    0    0    0    0    0    0    0    0    0]
 [   2    0    0    0    0    0    0    0    0    0    0    0]
 [   1    0    0    0    0    0    0    0    0    0    0    0]]
```

```
---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-22-b6e4cc749a14> in <module>
      8 from sklearn.metrics import confusion_matrix, f1_score
      9 print(confusion_matrix(Y_test, Y_pred))
---> 10 print(f1_score(Y_test, Y_pred))

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in
inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in
f1_score(y_true, y_pred, labels, pos_label, average, sample_weight,
zero_division)
   1066     modified with ``zero_division``.
   1067     """
-> 1068     return fbeta_score(y_true, y_pred, beta=1, labels=labels,
   1069                        pos_label=pos_label, average=average,
   1070                        sample_weight=sample_weight,

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in
inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in
fbeta_score(y_true, y_pred, beta, labels, pos_label, average,
sample_weight, zero_division)
   1190     """
   1191
-> 1192     _, _, f, _ = precision_recall_fscore_support(y_true,
y_pred,
   1193                                                  beta=beta,
   1194
labels=labels,

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in
inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
```

```
     64
     65             # extra_args > 0

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in
precision_recall_fscore_support(y_true, y_pred, beta, labels,
pos_label, average, warn_for, sample_weight, zero_division)
   1459     if beta < 0:
   1460         raise ValueError("beta should be >=0 in the F-beta
score")
-> 1461     labels = _check_set_wise_labels(y_true, y_pred, average,
labels,
   1462                                     pos_label)
   1463

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in
_check_set_wise_labels(y_true, y_pred, average, labels, pos_label)
   1289             if y_type == 'multiclass':
   1290                 average_options.remove('samples')
-> 1291             raise ValueError("Target is %s but
average='binary'. Please "
   1292                              "choose another average setting,
one of %r."
   1293                              % (y_type, average_options))

ValueError: Target is multiclass but average='binary'. Please choose
another average setting, one of [None, 'micro', 'macro', 'weighted'].
```