

(Group_20) Assignment_2

Prepared for Prof: Dr. Murat Simsek

Faculty of Engineering, University of u Ottawa

ELG5255[EG] Applied Machine Learning

Prepared by

Abdelsalam, Mohanad

Attia, Nada

El-Sabow, Shady

Part1:

(a) Using a Naive Bayes Classifier to classify a new instance which follows a condition
New Instance = (Blue, SUV, Domestic) into (Yes or No).

Solve

The dataset as:

Example No.	Color	Type	Origin	Stolen
1	Red	Sports	Domestic	Yes
2	Red	Sports	Domestic	No
3	Red	Sports	Domestic	Yes
4	Yellow	Sports	Domestic	No
5	Yellow	Sports	Imported	Yes
6	Blue	SUV	Imported	No
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	No
10	Blue	Sports	Imported	Yes
11	Red	SUV	Domestic	No
12	Red	SUV	Domestic	No
13	Blue	Sports	Imported	No
14	Red	SUV	Imported	Yes

So, we get a calculation table from it as following, it indicate the number of features in each class:

	Blue	SUV	Domestic	Total
Stolen=yes	1	2	2	6
Stolen=no	2	5	5	8
Total	3	7	7	14

We need to predict if the car will steal or not at given features, so we prepare some calculation before as:

$$\begin{aligned}
 P(\text{Blue, SUV, Domestic}) &= P(\text{Blue, SUV, Domestic} \mid \text{Stolen} = \text{Yes}) + P(\text{Blue, SUV, Domestic} \mid \text{Stolen} = \text{No}) \\
 &= 6/14 * [(1/6) * (2/6) * (2/6)] + 8/14 * [(2/8) * (5/8) * (5/8)] \\
 &= 1/126 + 25/448 = 257 / 4032 = 0.06374
 \end{aligned}$$

$$\begin{aligned}
 P(\text{Blue, SUV, Domestic} \mid \text{Stolen} = \text{Yes}) &= P(\text{Blue} \mid \text{Stolen} = \text{Yes}) * P(\text{SUV} \mid \text{Stolen} = \text{Yes}) * P(\text{Domestic} \mid \text{Stolen} = \text{Yes}) \\
 &= (1/6) * (2/6) * (2/6) \\
 &= 1/54
 \end{aligned}$$

$$\begin{aligned}
 P(\text{Blue, SUV, Domestic} \mid \text{Stolen} = \text{No}) &= P(\text{Blue} \mid \text{Stolen} = \text{No}) * P(\text{SUV} \mid \text{Stolen} = \text{No}) * P(\text{Domestic} \mid \text{Stolen} = \text{No}) \\
 &= (2/8) * (5/8) * (5/8) \\
 &= 25 / 256
 \end{aligned}$$

In this section, we will provide the calculation for the prediction

1. The probability of stole the car at given Blue, SUV, and Domestic features:

$$\begin{aligned}
 P(\text{Stolen} = \text{Yes} \mid \text{Blue, SUV, Domestic}) &= [(P(\text{Stolen} = \text{Yes}) * P(\text{Blue, SUV, Domestic} \mid \text{Stolen} = \text{Yes})) / \\
 &\quad P(\text{Blue, SUV, Domestic})] \\
 &= ((6/14) * (1/54)) / (257 / 4032) \\
 &= 32 / 257 = 0.1245
 \end{aligned}$$

2. The probability of not stole the car at given Blue, SUV, and Domestic features:

$$\begin{aligned}
 P(\text{Stolen} = \text{No} \mid \text{Blue, SUV, Domestic}) &= \\
 & (P(\text{Stolen} = \text{No}) * P(\text{Blue, SUV, Domestic} \mid \text{Stolen} = \text{No})) / P(\text{Blue, SUV, Domestic}) \\
 &= ((8/14) * (25 / 256)) / (257 / 4032) \\
 &= 225 / 257 = 0.8755
 \end{aligned}$$

The calculation provides that with these features the class will be Not stolen.

(b) Calculate the expected risk of three actions, and determine the rejection area of $P(\text{Class1} | x)$.

Solve

Target	Class 1	Class 2
a1 (Choose Class1)	0	6
a2 (Choose Class 2)	3	0
a3 (Rejection)	2	2

From the table below:

Calculate the risk for the action as:

$$\begin{aligned} R(a1 | x) &= \lambda_{11} p(\text{class 1} | x) + \lambda_{12} p(\text{class 2} | x) \\ &= 0 * p(\text{class 1} | x) + 6 * (1 - p(\text{class 1} | x)) \\ &= 6 - 6 p(\text{class 1} | x) \end{aligned}$$

$$\begin{aligned} R(a2 | x) &= \lambda_{21} p(\text{class 1} | x) + \lambda_{22} p(\text{class 2} | x) \\ &= 3 p(\text{class 1} | x) + 0 \\ &= 3 p(\text{class 1} | x) \end{aligned}$$

$$\begin{aligned} R(a3 | x) &= \lambda_{31} p(\text{class 1} | x) + \lambda_{32} p(\text{class 2} | x) \\ &= 2 * [p(\text{class 1} | x) + p(\text{class 2} | x)] \\ &= 2 \end{aligned}$$

If we selected a1 so:

$$R(a_1 | x) < R(a_2 | x)$$

$$6 - 6 p(a_1 | x) < 3 p(a_1 | x)$$

$$-9 p(a_1 | x) < -6$$

$$p(a_1 | x) > 2/3$$

and

$$R(a_1 | x) < R(a_3 | x)$$

$$6 - 6 p(a_1 | x) < 2$$

$$-6 p(a_1 | x) < -4$$

$$p(a_1 | x) > 2/3$$

If we selected a2 so:

$$R(a_2 | x) < R(a_1 | x)$$

$$3 p(a_1 | x) < 6 - 6 p(a_1 | x)$$

$$9 p(a_1 | x) < 6$$

$$p(a_1 | x) < 2/3$$

and

$$R(a_2 | x) < R(a_3 | x)$$

$$3 p(a_2 | x) < 2$$

$$p(a_2 | x) > 2/3$$

There is no rejection Area

part2:

(a)

We Split the dataset into two parts as training data and test data manually by this code:

without function used (a)

```
[ ] # Calculate the number of training samples
num_training_samples = int(0.8 * len(data))

# get training and testing data
training_data=data[:num_training_samples]      # 80% training data with shape (3680, 58)
testing_data=data[num_training_samples:]        # 20% testing data with shape (921, 58)

# separate target from the data
x_train = training_data[:, :-1] # training data without target , shape (3680, 57)
y_train = training_data[:, -1] # target of training data      , shape (3680,)

x_test = testing_data[:, :-1] # testing data without target   , shape (921, 57)
y_test = testing_data[:, -1] # target of testing data        , shape (921,)
```

Then, we used two classifiers, we make them as a function there are the code :

This function takes the splitted data to return the gaussian model:

```
[ ] # Create Gaussian Naive Bayes classifiers function
def gaussine(X_train,X_test,y_train,y_test):
    gnb = GaussianNB()
    # Train the classifier on the training data
    gnb.fit(X_train, y_train)
    # Predict the labels for the test data
    y_pred_gnb = gnb.predict(X_test)

    # Compute the accuracy of the Gaussian Naive Bayes classifier
    acc_gnb = accuracy_score(y_test, y_pred_gnb)
    print("Gaussian Naive Bayes accuracy:", acc_gnb)

    # Compute and print the confusion matrix for the Gaussian Naive Bayes classifier
    print("Gaussian Naive Bayes confusion matrix:")
    conf_and_plot(y_test, y_pred_gnb)

    return(acc_gnb)
```

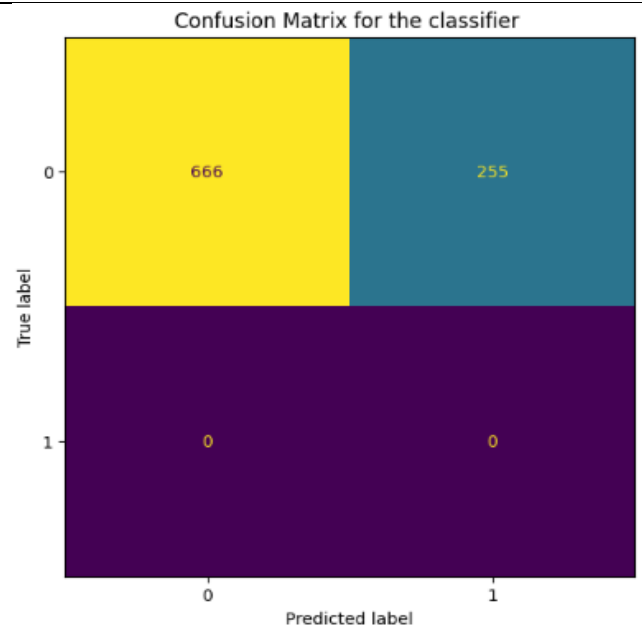
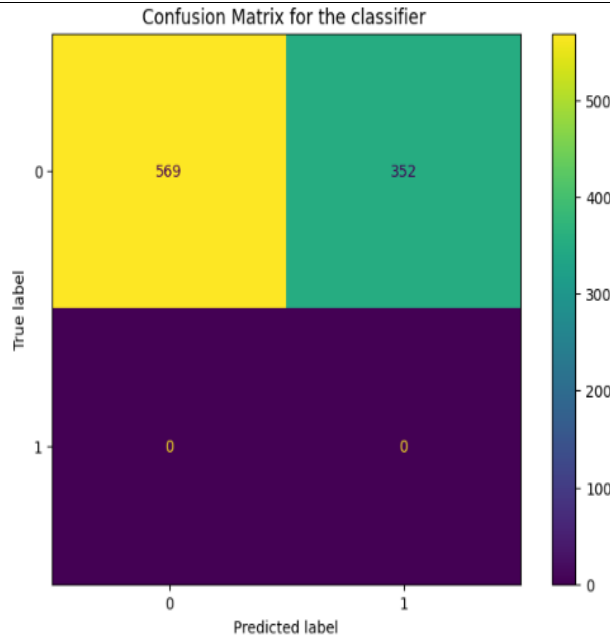
This function builds the multinomial naïve bayes model:

```
def multinomial(X_train,X_test,y_train,y_test):  
    # Create Multinomial Naive Bayes classifier  
    mnb = MultinomialNB()  
    # Train the classifier on the training data  
    mnb.fit(X_train, y_train)  
    # Predict the labels for the test data  
    y_pred_mnb = mnb.predict(X_test)  
  
    # Compute the accuracy Multinomial Naive Bayes classifier  
    acc_mnb = accuracy_score(y_test, y_pred_mnb)  
    print("Multinomial Naive Bayes accuracy:", acc_mnb)  
  
    # Compute the confusion matric for Multinomial Naive Bayes classifier  
    print("Multinomial Naive Bayes classifier confusion matrix:")  
    conf_and_plot(y_test, y_pred_mnb)  
  
    return(acc_mnb)
```

After we pass the data into this function, we get this result:

	Gaussian Naive Bayes Classifiers	Multinomial Naive Bayes Classifiers
Accuracy	0.6178067318132465	0.7231270358306189

Confusion matrix



Comments :

Due to manual splitting without any shuffle , the model train on data contain same target value so ,The zero's found. In addition to this Two classifier dosen't work will with this data so we get this lower accuracy .we will mention the difference in section (C)

b.

it's a same as (a) but with splitting function and a shuffle data as the code illusterated :

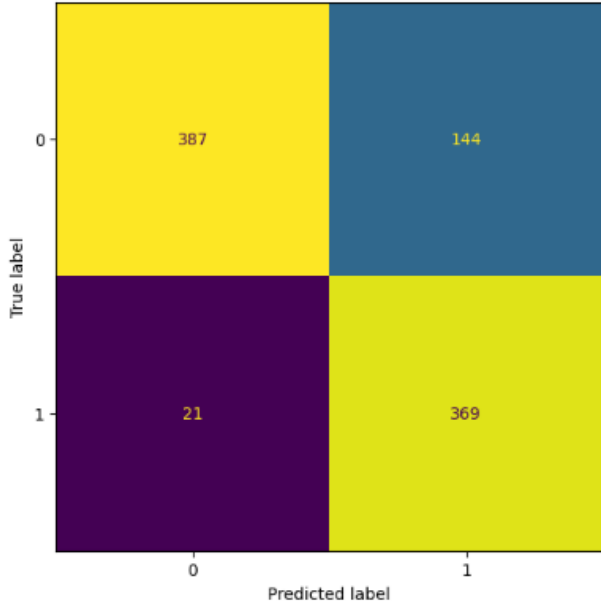
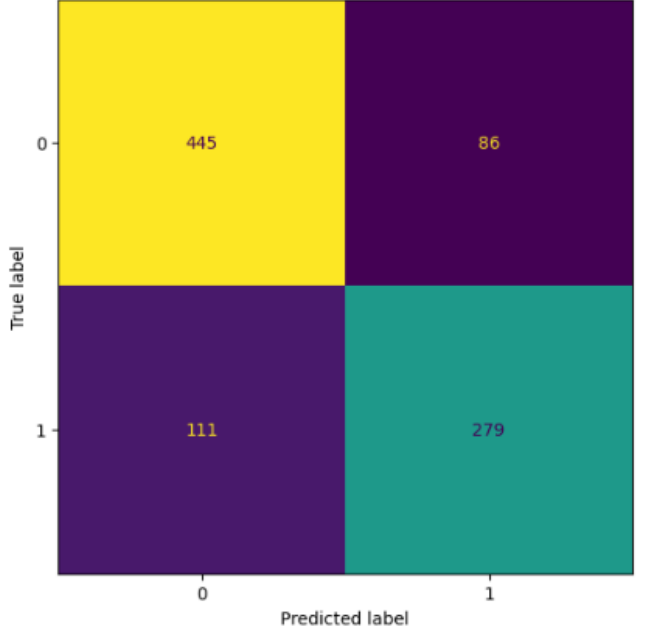
```
[ ] # splitting with function
# Split the data into the data and target variables
x = data[:, :-1]
y = data[:, -1]

# Split the dataset into training and test data
x_trainf, x_testf, y_trainf, y_testf = train_test_split(x, y, test_size=0.2, random_state=42)

# ouput{x_trainf, x_testf, y_trainf, y_testf}
```

The shape o
The shape o

so we get this result after we pass this data into the same two classifiers :

	Gaussian Naive Bayes Classifiers	Multinomial Naive Bayes Classifiers																		
Accuracy	0.8208469055374593	0.7861020629750272																		
Confusion matrix	<p>Confusion Matrix for the classifier</p>  <p>A 2x2 confusion matrix for the Gaussian Naive Bayes classifier. The y-axis is labeled 'True label' with values 0 and 1. The x-axis is labeled 'Predicted label' with values 0 and 1. The matrix cells contain the following counts: (0,0) is 387 (yellow), (0,1) is 144 (blue), (1,0) is 21 (purple), and (1,1) is 369 (yellow).</p> <table border="1"> <thead> <tr> <th>True \ Predicted</th> <th>0</th> <th>1</th> </tr> </thead> <tbody> <tr> <th>0</th> <td>387</td> <td>144</td> </tr> <tr> <th>1</th> <td>21</td> <td>369</td> </tr> </tbody> </table>	True \ Predicted	0	1	0	387	144	1	21	369	<p>Confusion Matrix for the classifier</p>  <p>A 2x2 confusion matrix for the Multinomial Naive Bayes classifier. The y-axis is labeled 'True label' with values 0 and 1. The x-axis is labeled 'Predicted label' with values 0 and 1. The matrix cells contain the following counts: (0,0) is 445 (yellow), (0,1) is 86 (purple), (1,0) is 111 (purple), and (1,1) is 279 (teal).</p> <table border="1"> <thead> <tr> <th>True \ Predicted</th> <th>0</th> <th>1</th> </tr> </thead> <tbody> <tr> <th>0</th> <td>445</td> <td>86</td> </tr> <tr> <th>1</th> <td>111</td> <td>279</td> </tr> </tbody> </table>	True \ Predicted	0	1	0	445	86	1	111	279
True \ Predicted	0	1																		
0	387	144																		
1	21	369																		
True \ Predicted	0	1																		
0	445	86																		
1	111	279																		

Comments:

The accuracy results are better than the previous one due to shuffle in the data and contain data a lot of different variables and the two target.

(c) Use another Naive Bayes classifier to check for the improvement in terms of accuracy score:

When we add a classifier as Bernoulli Naive Bayes classifier, we make it as a function this code explain what we meant:

```

[3] def brenoulli(X_train,X_test,y_train,y_test):
    # Create a BernoulliNB classifier
    ber = BernoulliNB()

    # Train the classifier on the training data
    ber.fit(X_train, y_train)

    # Predict the labels for the test data
    y_pred_ber = ber.predict(X_test)

    # Calculate the accuracy of the classifier
    acc_ber = accuracy_score(y_test, y_pred_ber)
    print("Brenoulli Naive Bayes classifier accuracy:", acc_ber)

    # Compute the confusion matrix for Brenoulli Naive Bayes classifier
    print("Brenoulli Naive Bayes classifier confusion matrix:")
    conf_and_plot(y_test, y_pred_ber)

    return(y_test, y_pred_ber, acc_ber)

```

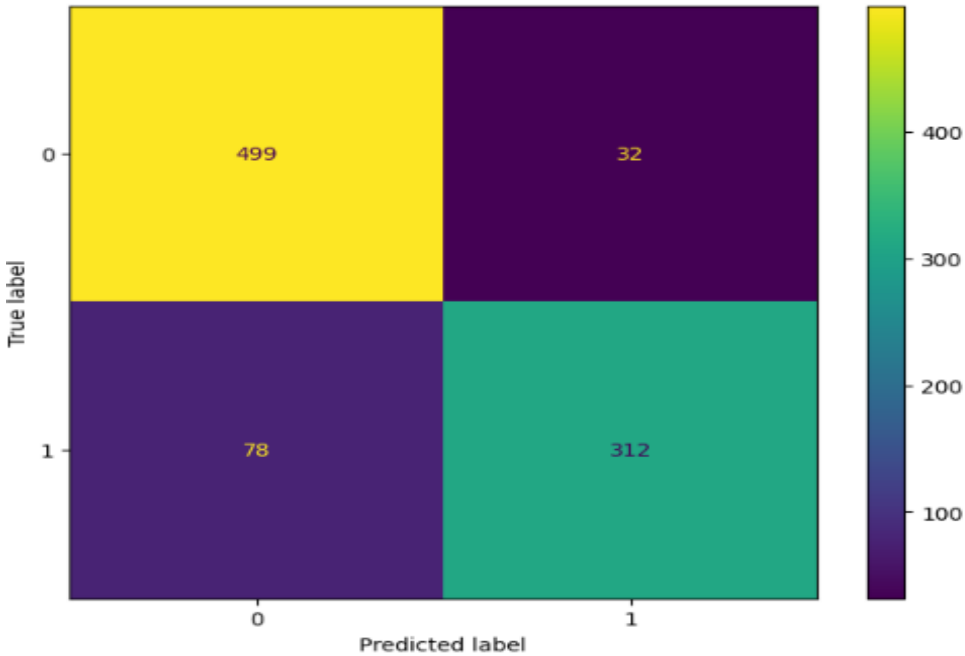
For show the classification report:

```

def classification_report_bernoulli(X_trainf,X_testf,y_trainf,y_testf):
    y_test_,y_pred_ber_, acc =brenoulli(X_trainf,X_testf,y_trainf,y_testf)
    # Print the classification report
    print(classification_report(y_test_, y_pred_ber_))

```

We get these results:

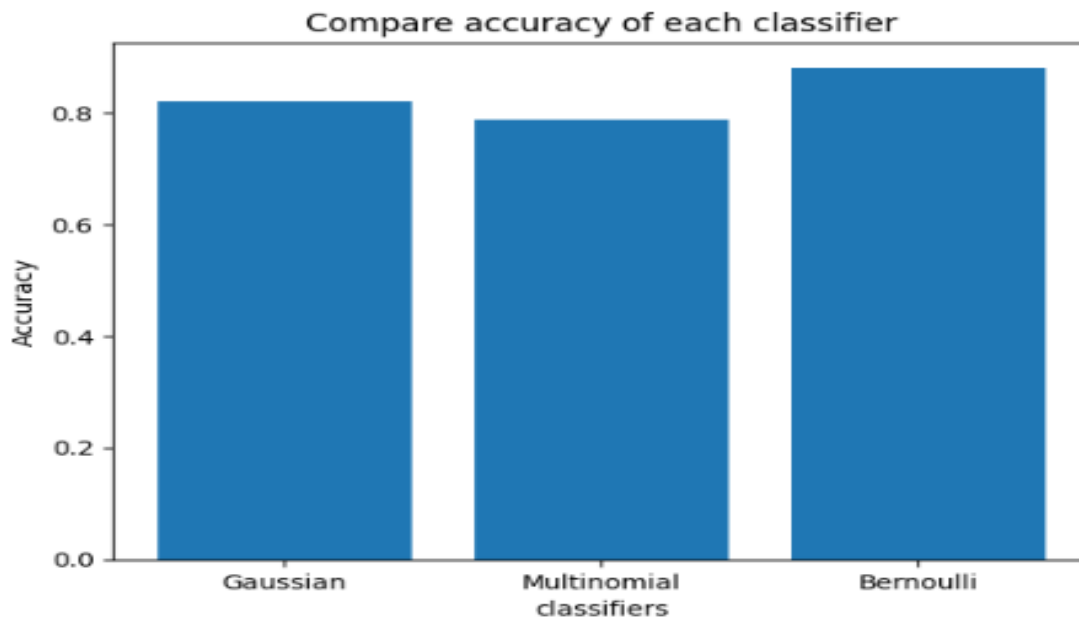
Accuracy	Confusion matrix									
0.8805646036916395	<p>Confusion Matrix for the classifier</p>  <table><tr><th>True \ Predicted</th><th>0</th><th>1</th></tr><tr><th>0</th><td>499</td><td>32</td></tr><tr><th>1</th><td>78</td><td>312</td></tr></table>	True \ Predicted	0	1	0	499	32	1	78	312
True \ Predicted	0	1								
0	499	32								
1	78	312								

Comments:

Compare between 3 classifiers:

- gaussian is used in classification tasks and it assumes that feature values follow a gaussian distribution.
- Multinomial Naive Bayes is used for discrete counts.
- Bernoulli Naive Bayes: The binomial model is useful if your feature vectors are Boolean (i.e. zeros and ones).
- Our problem is binary classification 1 spammed ,0 not spammed.
- From accuracy we found that for the same data with shuffle:

Gaussian Naive Bayes accuracy: 0.8208469055374593
 Multinomial Naive Bayes accuracy: 0.7861020629750272
 Bernoulli Naive Bayes classifier accuracy: 0.8805646036916395



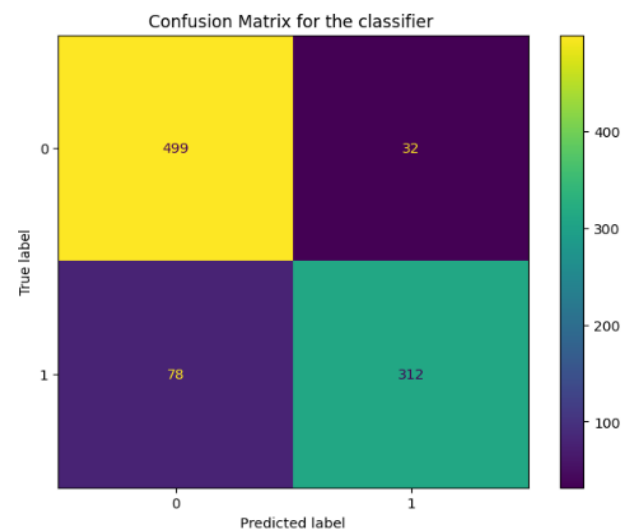
From the concept related to our problem and the accuracy of the different classifiers, we choose a Bernoulli classifier:

```
f=classification_report_bernoulli(x_trainf,x_testf,y_trainf,y_testf)
```

Bernoulli Naive Bayes classifier accuracy: 0.8805646036916395
 Bernoulli Naive Bayes classifier confusion matrix:

	precision	recall	f1-score	support
0.0	0.86	0.94	0.90	531
1.0	0.91	0.80	0.85	390
accuracy			0.88	921
macro avg	0.89	0.87	0.88	921
weighted avg	0.88	0.88	0.88	921

With confusion matrix as shown before:



(d) We split the data into subsets as following code:

```
# Calculate the number of training samples
num_training_samples = int(0.8 * len(data))
# get training and testing data
training_data=data[:num_training_samples]          # 80% training data with shape
(3680, 58)
testing_data=data[num_training_samples:]            # 20% testing data with shape
(921, 58)

# calculate the start/end of training samples:
num_training_samples_1 = int(0.25* len(training_data))      # 920
num_training_samples_2 = int(0.5* len(training_data))        # 1840
num_training_samples_3 =int(0.75* len(training_data))        # 2760
num_training_samples_4 =int(1* len(training_data))           # 3680
print(num_training_samples_1, num_training_samples_2 , num_training_samples_3
,num_training_samples_4)

# for (subset 1)
subset_1 =training_data[:num_training_samples_1 , :]        #subset1 with shape (920,
58)
x_train1 = subset_1[:, :-1]          # The shape of the x data is:  (920, 57)
y_train1 = subset_1[:, -1]          # The shape of the y(target) data is:  (920,1)

# for (subset 2)
subset_2 =training_data[num_training_samples_1 : num_training_samples_2 , :]
#subset2 with shape (920, 58)
x_train2 = subset_2[:, :-1]          # The shape of the x data is:  (920, 57)
y_train2 = subset_2[:, -1]          # The shape of the y(target) data is:  (920,1)

# for (subset 3)
subset_3 =training_data[num_training_samples_2 : num_training_samples_3 , :]
#subset3 with shape (920, 58)
x_train3 = subset_3[:, :-1]          # The shape of the x data is:  (920, 57)
y_train3 = subset_3[:, -1]          # The shape of the y(target) data is:  (920,1)

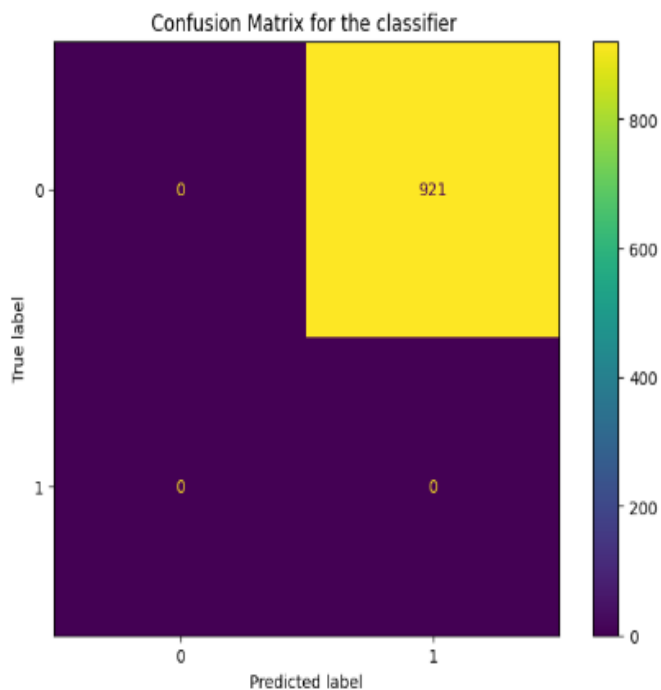
# for (subset 4)
subset_4 =training_data[num_training_samples_3 : num_training_samples_4 ,
:] #subset4 with shape (920, 58)
x_train4 = subset_4[:, :-1]          # The shape of the x data is:  (920, 57)
y_train4 = subset_1[:, -1]          # The shape of the y(target) data is:  (920,1)

# 20% of test data
x_test0 = testing_data[:, :-1]      # testing data without target      , shape (921, 57)
y_test0 = testing_data[:, -1]      # target of testing data            , shape (921,1)
```

Then, we used Bernoulli naïve classifier model to predict, we get the following accuracy:

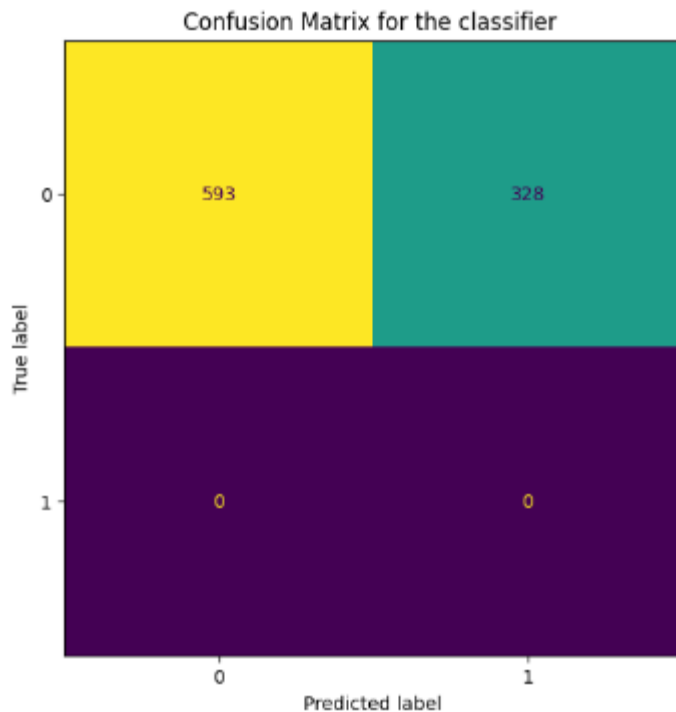
For subset_1: [from row 1 into 920]

accuracy: 0.0



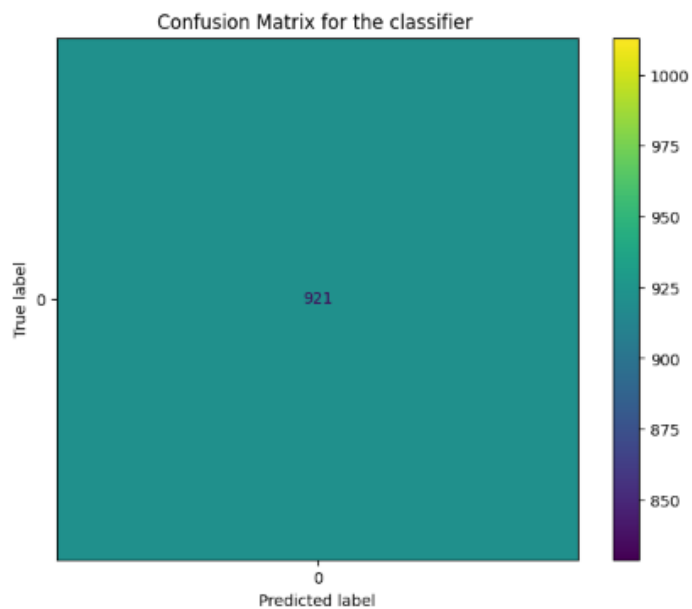
For subset_2 [from row 920 into 1840]

accuracy: 0.6438653637350705



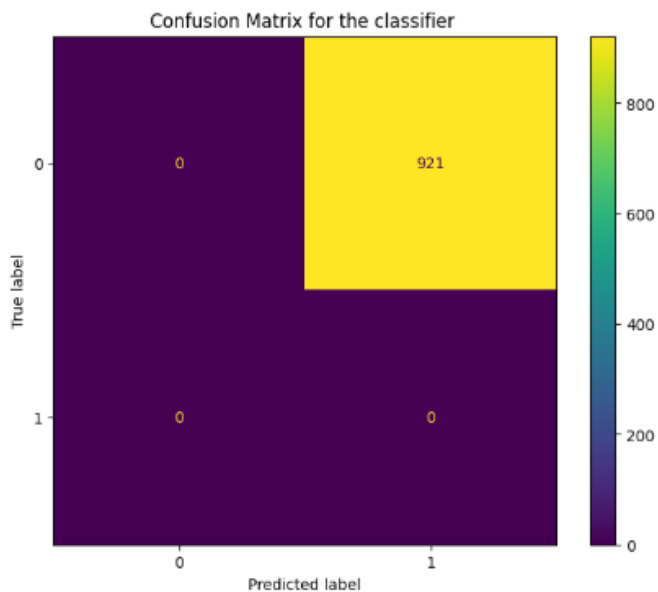
For subset_4: [from row 1840 into 2760]

accuracy: 1

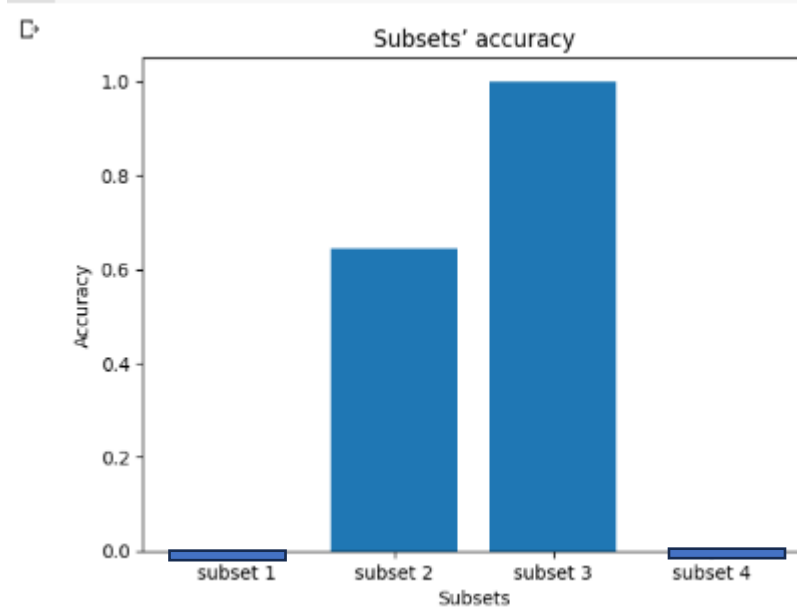


For subset_3 [from row 2760 into 3680]

accuracy: 0



Compare between all of them by bar plot as shown:



Comment:

- The previous accuracy is related to non-shuffle of the data.
- We take a deep look at the data in each subset we found:

For subset 1:

it contains only ones

```
for subset 2:
```

[illegible]

it contains one and zero data so that's way this data get a result not 0 accuracy

for subset 3:

[illegible]

it contain only zeros as the target ,so the accuracy will be the highest one

[illegible]

it contains only zeros

for subset 4:

[illegible]

[illegible]

it contains only one so the accuracy will be zero.

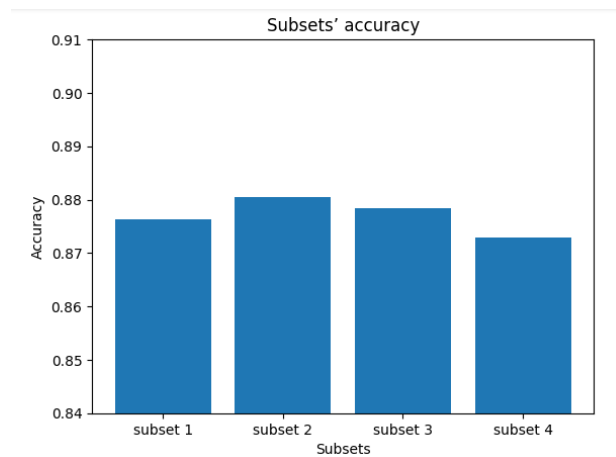
In the unupdated report you asked us to make it with shuffle data so when I make it I get this result:

```
Brenoulli Naive Bayes classifier accuracy with subset_1: 0.8762214983713354
```

```
Brenoulli Naive Bayes classifier accuracy with subset_2: 0.8805646036916395
```

```
Brenoulli Naive Bayes classifier accuracy with subset_3: 0.8783930510314875
```

```
Brenoulli Naive Bayes classifier accuracy with subset_4: 0.8729641693811075
```



That's why solving the bias (of the data) will help us to improve the accuracy of the data as it can lead to inaccurate results and it's consider as an ethics condition we must prevent any bias in our data.