uOttawa

(Group_10) Assignment_2

Prepared for

Prof: Murat SIMSEK
TA: Samhita Kuili

Faculty of Engineering, University of u Ottawa

ELG5255: Applied Machine Learning

Prepared by

Esmael Alahmady Ebrahim Ezz

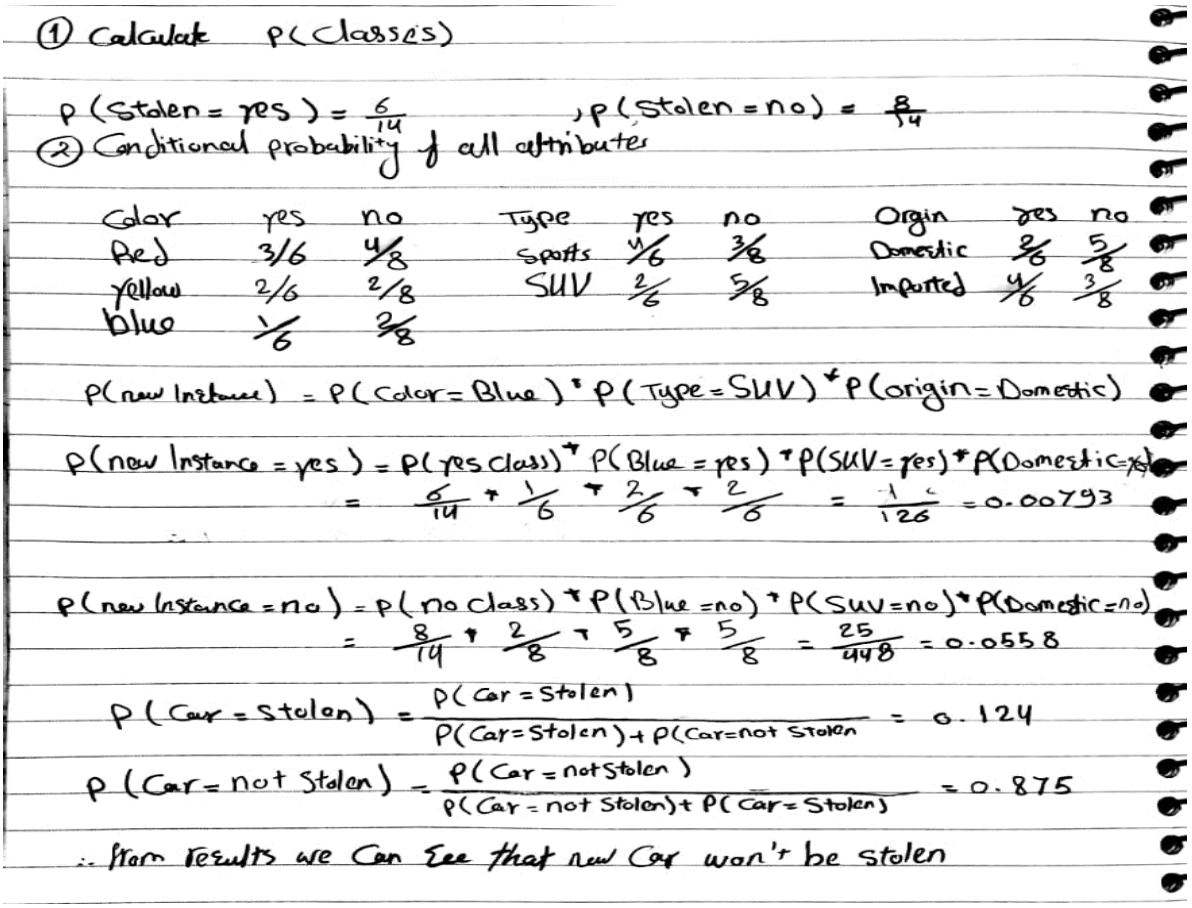
Yousef Abd Al Haleem Ahmed Shindy

Esraa Mahmoud Said Ahmed

Part_1

1.

- Firstly, we calculate the prior for the class yes (6/14) and class no (8/14)
- Then we calculate the probability of instances for each feature
- After that we calculate probability of each class (Yes or No) given the new instance (Blue, SUV, Domestic)
- The results show that probability of class **no** is bigger than probability of class yes.



2.

- We calculate the expected risk of action 1, action 2 and action 3 (Reject).
- We determine the rejection area of P(Class1|x) by comparing the expected risk of action 1 with action 3 (Reject) and action 2 with action 3 (reject)
- We choose α1 when $R(\alpha_1|x) < 2 \quad \therefore P(C_1|x) > \frac{2}{3}$
- We choose α2 when $R(\alpha_2|x) < 2 \quad \therefore P(C_1|x) < \frac{2}{3}$

$R(\alpha_i|x) = \sum_{k=1}^{K} \lambda_{ik} P(C_k|x)$

$R(\alpha_1|x) = \lambda_{11} P(C_1|x) + \lambda_{12} P(C_2|x) = 0 \times P(C_1|x) + 6P(C_2|x) = 6[1 - P(C_1|x)]$

$R(\alpha_2|x) = \lambda_{21} P(C_1|x) + \lambda_{22} P(C_2|x) = 3P(C_1|x) + 0 \times P(C_2|x) = 3P(C_1|x) = 3[1 - P(C_2|x)]$

$R(\alpha_3|x) = \lambda_{31} P(C_1|x) + \lambda_{32} P(C_2|x) = 2P(C_1|x) + 2P(C_2|x) = 2[P(C_1|x) + P(C_2|x)] = 2$

We Choose α_1 if $R(\alpha_1|x) < 2$

$\therefore 6 - 6P(C_1|x) < 2$

$\therefore P(C_1|x) > \frac{2}{3}$

We Choose α_2 if $R(\alpha_2|x) < 2$

$3P(C_1|x) < 2$

$\therefore P(C_1|x) < \frac{2}{3}$

We reject if $\frac{2}{3} < P(C_1|x) < \frac{2}{3}$ #

Part_2

a.

Initially, we divided the data into two parts: X and Y. Subsequently, we further divided these X and Y components into two sets, allocating 80% for training data and 20% for testing data. Importantly, the order of the original data was maintained during this split. Afterwards, we utilized both the Gaussian Naïve Bayes classifier and the Multinomial Naïve Bayes classifier to fit the data. For each classifier, we computed the accuracy of the test set and generated a heat map visualization of the confusion matrix.

- **Split function with maintaining order.**

```
def percent_split(X, Y , percent):
    subset_size = int(len(X) * (percent/100))
    x_train = X[:subset_size]
    y_train = Y[:subset_size]
    x_test = X[subset_size:]
    y_test = Y[subset_size:]
    return x_train, x_test, y_train, y_test
```

```
first_80_x_train, last_20_x_test, first_80_y_train, last_20_y_test = percent_split(X,Y,80)
```

```
print(first_80_x_train.shape)
print(last_20_x_test.shape)
print(first_80_y_train.shape)
print(last_20_y_test.shape)
```

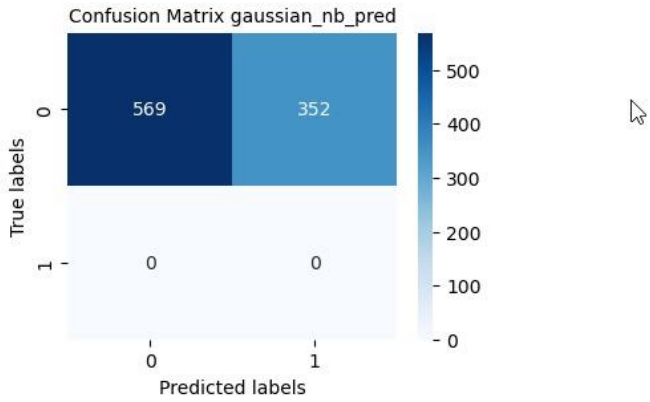
```
(3680, 57)
(921, 57)
(3680,)
(921,)
```

- **Gaussian Function**

```
def GNB(x_train, x_test, y_train, y_test):
    gaussian_nb = GaussianNB()
    gaussian_nb.fit(x_train, y_train)
    gaussian_nb_pred = gaussian_nb.predict(x_test)
    print("the Gaussian Accuracy : " , get_accuracy(y_test , gaussian_nb_pred))
    confusion_mx(y_test , gaussian_nb_pred , "gaussian_nb_pred")
```

```
GNB(first_80_x_train, last_20_x_test, first_80_y_train, last_20_y_test)
```

```
the Gaussian Accuracy :  61.78067318132465
```

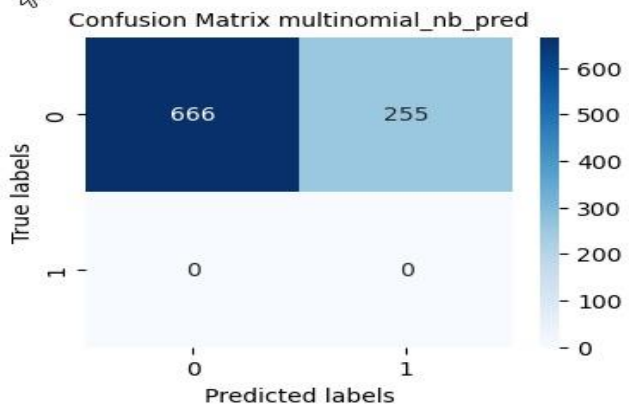


- **Multinomial Function**

```
def MNB(x_train, x_test, y_train, y_test):
    multinomial_nb = MultinomialNB()
    multinomial_nb.fit(x_train, y_train)
    multinomial_nb_pred = multinomial_nb.predict(x_test)
    print("the Multinomial Accuracy : " , get_accuracy(y_test , multinomial_nb_pred))
    confusion_mx(y_test , multinomial_nb_pred , "multinomial_nb_pred")
```

```
MNB(first_80_x_train, last_20_x_test, first_80_y_train, last_20_y_test)
```

```
the Multinomial Accuracy :  72.31270358306189
```



b.

In this experiment, we performed a random split of the X and Y datasets, allocating 80% for training data and 20% for testing data. Subsequently, we implemented both the Gaussian Naïve Bayes classifier and the Multinomial Naïve Bayes classifier to train the data. For each classifier, we evaluated the accuracy on the test set and visualized the confusion matrix as a heat map.

- **Split function with maintaining order.**

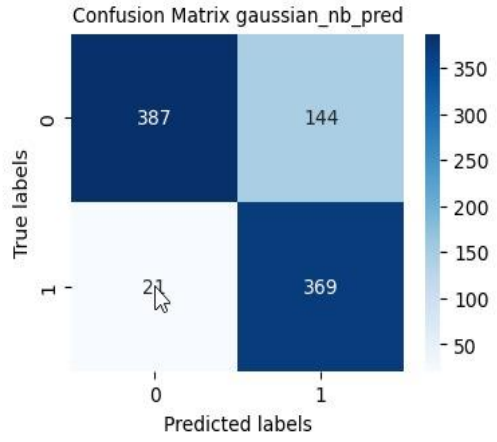
```
def split_data(x , y , train_size) :
    X_train, X_test, y_train, y_test = train_test_split(x, y, train_size = train_size , shuffle = True , random_state = 42)
    return X_train, X_test, y_train, y_test
```

```
x_train, x_test, y_train, y_test = split_data(X , Y , 0.8)
```

Gaussian

```
GNB(x_train, x_test, y_train, y_test)
```

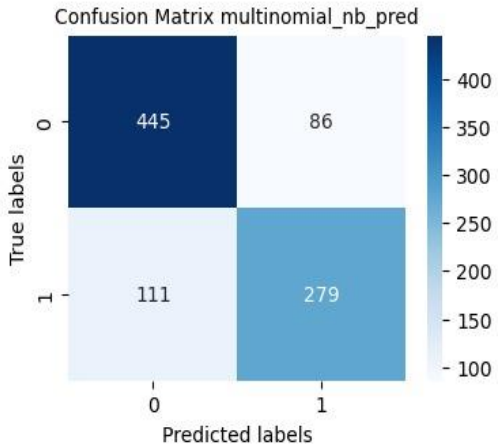
```
the Gaussian Accuracy :  82.08469055374593
```



Multinomial

```
MNB(x_train, x_test, y_train, y_test)
```

```
the Multinomial Accuracy :  78.61020629750271
```



c.

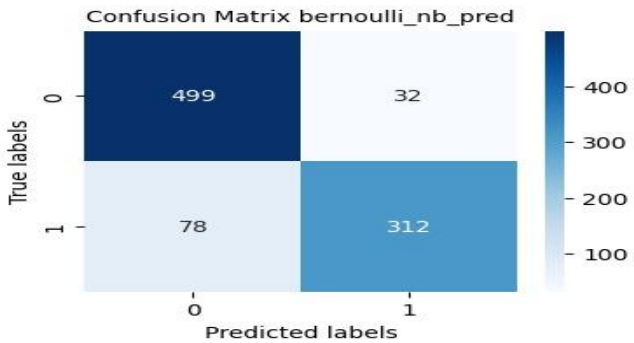
In this scenario, we selected both the random train and test sets. Our choice of the Bernoulli naïve Bayes classifier was based on the nature of the data, where values were represented as either 0 (not spam) or 1 (spam). We anticipated that the Bernoulli classifier would yield favorable results given this data format. After fitting the data and assessing the model's accuracy, we observed a noticeable improvement. We proceeded to generate a classification report and visualize the confusion matrix. The disparity in accuracy between the Bernoulli classifier and the Gaussian and Multinomial naïve Bayes classifiers can be attributed to the data imbalance. It is worth noting that the Gaussian and Multinomial classifiers are particularly sensitive to imbalanced data.

- **Bernoulli Function**

```
def BNB(x_train, x_test, y_train, y_test):
    BNB = BernoulliNB()
    BNB.fit(x_train , y_train)
    bernoulli_nb_pred = BNB.predict(x_test)
    print("the Bernoulli Accuracy : " , get_accuracy(y_test , bernoulli_nb_pred))
    confusion_mx(y_test , bernoulli_nb_pred , "bernoulli_nb_pred")
    print(classification_report(y_test, bernoulli_nb_pred))
```

- **Accuracy, Confusion matrix & classification report**

```
BNB(x_train, x_test, y_train, y_test)
the Bernoulli Accuracy :  88.05646036916394
```



	precision	recall	f1-score	support
0	0.86	0.94	0.90	531
1	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

d.

In this approach, we begin by obtaining the train and test sets in a sequential manner as described in point (a). Next, we divide the train sets into four equally sized and ordered subsets, each containing 25% of the original train data. The remaining 20% of the original data is set aside as the testing set. We then proceed to train each subset from the X train set with its corresponding subset from the Y train set. For each subset, we calculate and display the accuracy. Finally, we use a bar plot to visually represent the four resulting accuracy values.

- **split function for equal ordered sets**

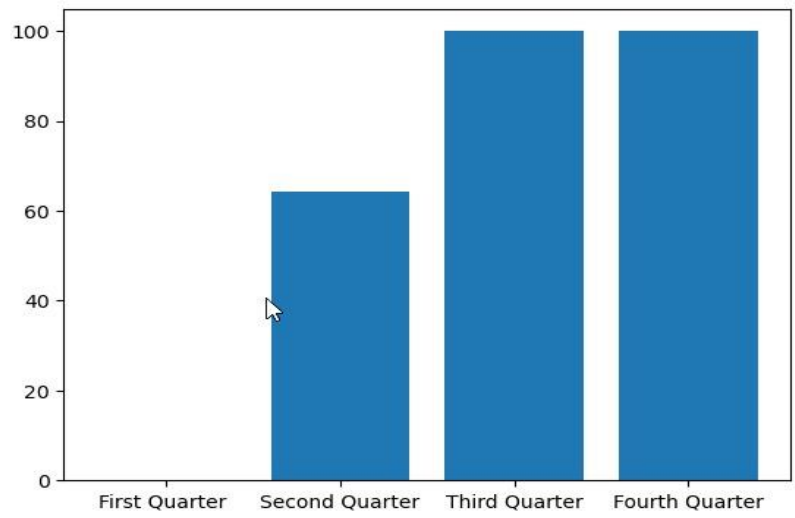
```
def split_df_4_equal_ordered_sets(x_train , y_train , sets_num):
    X_subsets = np.array_split(x_train, sets_num)
    Y_subsets = np.array_split(y_train, sets_num)
    return X_subsets , Y_subsets
```

- **Bernoulli Function for fitting multiple sets**

```
def multiple_BNB(x_train, x_test, y_train, y_test):
    results_list = []
    x_list = ["First Quarter " , "Second Quarter" , "Third Quarter" , "Fourth Quarter"]
    BNB = BernoulliNB()
    X_subsets , Y_subsets = split_df_4_equal_ordered_sets(x_train , y_train , 4)
    for subset_x , subset_y in zip(X_subsets , Y_subsets) :
        BNB.fit(subset_x , subset_y)
        pred = BNB.predict(x_test)
        result = get_accuracy(y_test , pred)
        print(f"The Accuracy : " , result)
        results_list.append(result)
    plt.bar( x_list, results_list )
```

```
multiple_BNB(first_80_x_train, last_20_x_test, first_80_y_train, last_20_y_test)
```

```
The Accuracy :  0.0
The Accuracy :  64.38653637350706
The Accuracy :  100.0
The Accuracy :  100.0
```



- **Subset_1**

In the first subset, which accounts for the first 25% of the 80% of the original data, we obtained an accuracy of zero percent. This occurred since all the values in the corresponding subset of the y train data were ones. Consequently, when training the model on this data and making predictions on the test set, we consistently obtained ones. On the other hand, the test data, representing the remaining 20% of the 80% of the original data, consisted entirely of zeros. As a result, when comparing the predicted values with the test values, there was a complete mismatch, resulting in a zero percent accuracy.

```
TRAINING :
 1    920
Name: 57, dtype: int64

TESTING :
 0    921
Name: 57, dtype: int64

 The Accuracy is :  0.0
```

- **Subset_2**

Moving to the second subset of the train data, we achieved an accuracy of approximately 65%. This improvement can be attributed to the fact that the second subset of the y train data contained a mixture of zeros and ones. Consequently, when fitting the model with this data and predicting the x test set, we obtained a combination of zeros and ones in the results.

```
TRAINING :
 1    893
0     27
Name: 57, dtype: int64

TESTING :
 0    921
Name: 57, dtype: int64

 The Accuracy is :  64.38653637350706
```

- **Subset_3 & Subset_4**

In the third and fourth subsets, we achieved a perfect 100% accuracy. This was because the y train data in these subsets consisted entirely of zeros. Consequently, when training the model on these zeros and predicting the x test set, all the values turned out to be zeros. As previously mentioned, the y test data also contained only zeros. Thus, when comparing the predicted values with the test values, they perfectly matched, resulting in a 100% accuracy.

```
TRAINING :
 0    920
Name: 57, dtype: int64

TESTING :
 0    921
Name: 57, dtype: int64

 The Accuracy is :  100.0
```

```
TRAINING :
 0    920
Name: 57, dtype: int64

TESTING :
 0    921
Name: 57, dtype: int64

 The Accuracy is :  100.0
```