CS464

Machine Learning

HW1 Report

N. Onur VURAL

21902330

SEC: 2

Q1: The CS 464 Case

- **The statistics show 87% of the high (H) grades, 21% of the low grades (L), and 4% of the failing (F) grades are received by the motivated students.**
- **The statistics show 13% of the high (H) grades, 79% of the low grades (L), and 96% of the failing (F) grades are received by the unmotivated students.**
- **64%, 24%, and 12% of the students got high, low, and failing grades, respectively.**

Question 1.1 [10 pts] What is the probability that a student is motivated [P (S_M)]?

P (S_M) = P(H) X P (S_M|H) + P(L) X P (S_M|L) + P(F) X P (S_M|F)

P (S_M) = $0.64 \times 0.87 + 0.24 \times 0.21 + 0.12 \times 0.04 = 0.612 = 61.2\ \%$

Question 1.2 [10 pts] If a student is motivated, what is the probability that he/she will get a high grade [P (H|S_M)]?

P (H|S_M) = $\frac{\mathrm{P}\,(S_M|\mathrm{H}) \times \mathrm{P}\,(\mathrm{H})}{\mathrm{P}\,(S_M)}$

- $\mathrm{P}\,(S_M|\mathrm{H}) = 0.87$
- $\mathrm{P}\,(\mathrm{H}) = 0.64$
- P (S_M) = 0.612

P (H|S_M) = $\frac{0.87 \times 0.64}{0.612} = 0.9098$

Question 1.3 [10 pts] If a student is unmotivated, what is the probability that he/she will get a high grade [P (H|S_U)]?

P (H|S_U) = $\frac{\mathrm{P}\,(S_U|\mathrm{H}) \times \mathrm{P}\,(\mathrm{H})}{\mathrm{P}\,(S_U)}$

- $\mathrm{P}\,(S_U|\mathrm{H}) = 0.13$
- $\mathrm{P}\,(\mathrm{H}) = 0.64$
- P (S_U) = $1 - \mathrm{P}\,(S_M) = 0.64 \times 0.13 + 0.24 \times 0.79 + 0.12 \times 0.96 = 0.388$

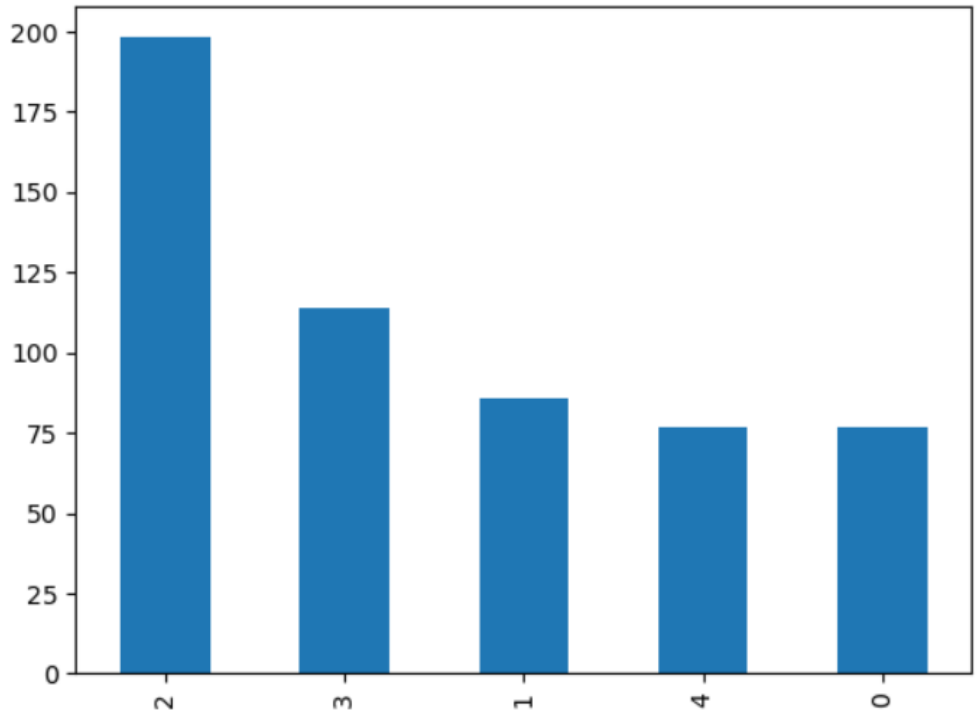
P (H|S_M) = $\frac{0.13 \times 0.64}{0.388} = 0.2144$

Q2.1

1. How are the classes distributed in the training set? Give the number of instances in each class with a suitable plot. Any plot that shows the class distribution is acceptable.

```
sports_train_df.class_label.value_counts().plot(kind='bar')
```

```
<AxesSubplot:>
```




```
sports_train_df.class_label.value_counts()
```

```
2    198
3    114
1     86
4     77
0     77
Name: class_label, dtype: int64
```

Figure 1: Class distribution graph alongside with their exact values.

- The results demonstrate in Figure 2 that in terms of number of class instances:
  2 (football) > 3 (rugby) > 1 (cricket) > 0 (athletics) = 4 (tennis)

2. Is the training set balanced or skewed towards one of the classes? Do you think having an imbalanced training set affects your model? If yes, please explain how it affects the Naive Bayes model and propose a possible solution if needed.

```
sports_train_df.class_label.value_counts().plot(kind='pie', autopct='%1.1f%%',counterclock=False,startangle=90)
```

```
<AxesSubplot:ylabel='class_label'>
```

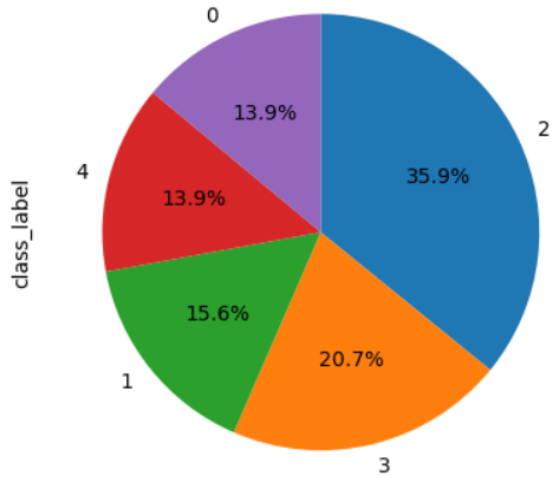


Figure 2: Percentile class distribution pie chart.

According to results, it appears that the given training set is definitely not a highly imbalanced set but there is skew for class 2. The total number of records is 552 and none of the given classes exceed 50% or completely dominate the others in a sense. However, class 2 is more than 50% of the smallest class that is 0/4. As Figure 2 suggests, classes take a relatively fair share in the pie chart. Obviously, it is a question that how imbalanced is imbalanced and relatively speaking, an informal description would be the given dataset can be considered skewed if a specific class exceeds another class by around 50%. In such classification problems, having an imbalanced dataset becomes risky as it may give bad predictions for the classes having small number of records and this is mostly because this situation affects P ($Y = y_k$) term, prior probability. When we have imbalanced dataset, Naive Bayes tends to get biased towards the majority class. Having a balanced dataset helps to balance this $P(Y = y_k)$ probability in this sense (Additional Note: A more destructive problem would be having insufficient data where a specific class has zero instance for a specific feature. Using Laplace smoothing and adding arbitrary instances would prevent getting zero in such case). To convert imbalanced dataset to a balanced dataset, over sampling (increasing minority class instances such as injecting training data copies of minority class instances) or under sampling (reducing majority classes such as removing samples) can be possible solutions.

3. Does the validation set, and the training set have similar data distributions? Regardless of the answer, if we had a bad split where validation set and the training set have different characteristics, which term in the Naive Bayes algorithm would be misleading?

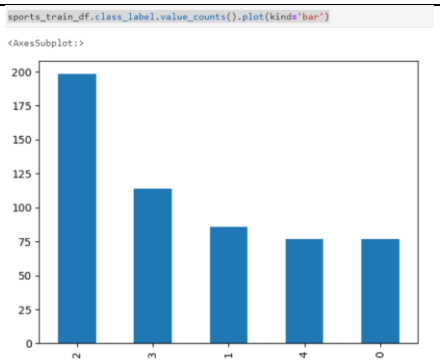
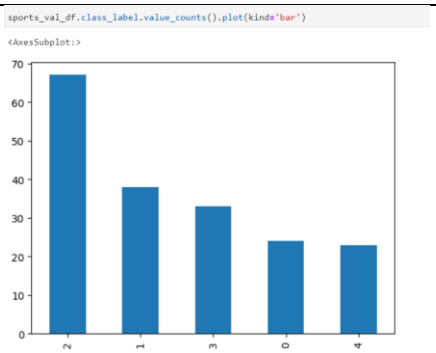
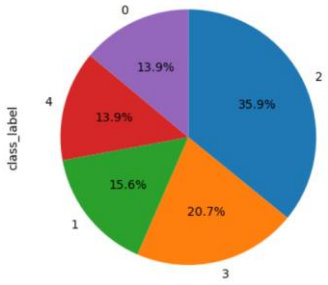
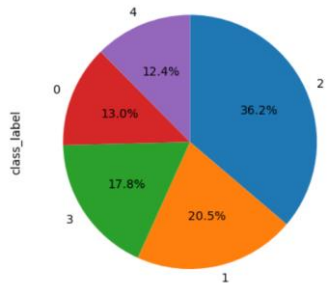
	Training Set	Validation Set
Number of instances (737)	**552 records**	**185 records**
Split (%)	**74.9%**	**25.1%**
Class distribution graph		
Class distribution values	`sports_train_df.class_label.value_counts()` 2 198 3 114 1 86 4 77 0 77 Name: class_label, dtype: int64	`sports_val_df.class_label.value_counts()` 2 67 1 38 3 33 0 24 4 23 Name: class_label, dtype: int64
Class distribution pie chart		

Figure 3: Data distribution comparison of validation set and the training set

The results in Figure 3 demonstrate that there are differences between training set and validation set in terms of data distributions. Although the highest frequency belongs to class 2 (football) in both of them, the following order of the validation set does not match with training set. In case of a bad split where validation and training sets possess different characteristics, there is the risk of $P(X_j \mid Y = y_k)^{twj,i}$ term being affected. This is due to the possibility that since the frequencies for words in accordance with class order will be changing. In a bad split this outcome will be much effective. Also, $P(Y = y_k)$ can be most likely misleading as well since the class probabilities of validation set can be much different than class probabilities of test set in such a bad split, thus not being able to effectively represent true data distribution.

4. If your dataset is skewed towards one of the classes, does this affect your reported accuracy? If yes, to what extent the reported accuracy is misleading in such an unbalanced dataset?

When we are working with a dataset that is skewed towards a specific class within, the accuracy can look high while making biased predictions at the same time. Accuracy essentially expresses the performance of our model in terms of making correct class predictions in relation to total number of predictions. When dataset is skewed, the majority classes dominating, model will begin to interpret the minority classes almost as an anomaly while always giving prediction in favor of the majority class. If the majority class is indeed very high in percentile, this situation may get to an extreme where the model never gives prediction as minority class.

CODE SECTION FOR 2.2 & 2.3

STEP 1: LOADING AND ANALYSING DATA

In [1]:
```
import pandas as pd
```

In [2]:
```
sports_train_df = pd.read_csv("bbcsports_train.csv")
```

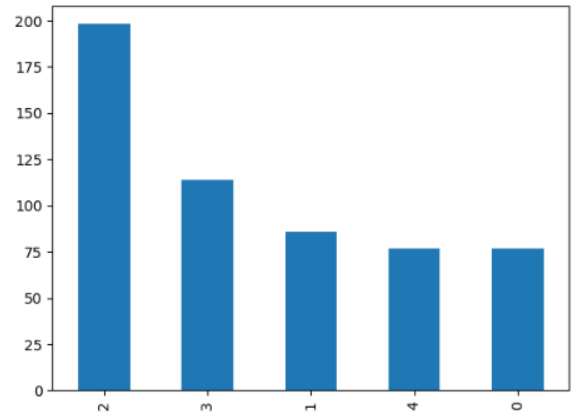
In [3]:
```
print(sports_train_df)
```

```
     claxton  hunt  first  major  medal  british  hurdler  sarah  confid  win  \
0          0     0      0      0      0        0        0      0       0    0
1          0     0      1      0      0        0        0      0       0    1
2          0     0      1      0      0        0        0      0       0    2
3          0     0      0      2      0        0        0      0       0    0
4          0     0      0      0      0        0        0      0       0    1
..       ...   ...    ...    ...    ...      ...      ...    ...     ...  ...
547        0     0      0      0      0        0        0      0       0    0
548        0     0      0      0      0        0        0      0       0    0
549        0     0      0      0      0        0        0      0       0    2
550        0     0      1      0      0        0        0      0       1    3
551        0     0      1      0      0        0        0      0       0    1

     ...  massu  mcenro  mauresmo  ameli  hip  fiveset  mario  ancic  \
0    ...      0       0         0      0    0        0      0      0
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    ...      0       0         0      0    0        0      0      0
..   ...    ...     ...       ...    ...  ...      ...    ...    ...
547  ...      0       0         0      0    0        0      0      0
548  ...      0       0         0      0    0        0      0      0
549  ...      0       0         0      0    0        0      0      0
550  ...      0       0         0      0    0        0      0      0
551  ...      0       0         0      0    0        0      0      0

     lundgren  class_label
0           0            3
1           0            1
2           0            2
3           0            2
4           0            2
..        ...          ...
547         0            2
548         0            2
549         0            2
550         0            2
551         0            2

[552 rows x 4614 columns]
```

TEST DATA: Class distribution graph

In [4]:
```
sports_train_df.class_label.value_counts().plot(kind='bar')
```

Out[4]: <AxesSubplot:>



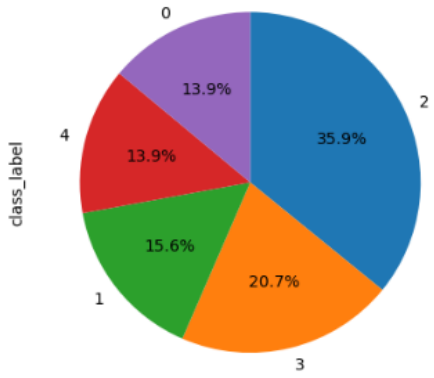
In [5]:
```
sports_train_df.class_label.value_counts()
```

Out[5]:
```
2    198
3    114
1     86
4     77
0     77
Name: class_label, dtype: int64
```

TEST DATA: Class distribution chart

In [6]:
```
sports_train_df.class_label.value_counts().plot(kind='pie', autopct='%1.1f%%',counterclock=False,startangle=90)
```

Out[6]: <AxesSubplot:ylabel='class_label'>



In [7]:
```
sports_val_df = pd.read_csv("bbcsports_val.csv")
```

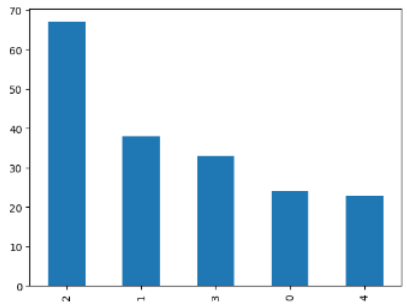
In [8]:
```
print(sports_val_df)
```

```
     claxton  hunt  first  major  medal  british  hurdler  sarah  confid  win  \
0          0     0      0      0      0        0        0      0       0    0
1          0     0      0      0      0        0        0      0       0    0
2          0     0      8      1      0        0        0      0       0    0
3          0     0      0      0      0        0        0      0       0    0
4          0     0      1      0      0        0        0      0       0    0
..       ...   ...    ...    ...    ...      ...      ...    ...     ...  ...
180        0     0      0      0      0        0        0      0       0    3
181        0     0      5      0      6        1        0      0       0    5
182        0     0      2      0      0        0        0      0       0    0
183        0     0      0      0      0        0        0      0       0    1
184        0     0      0      0      0        0        0      0       0    1

     ...  massu  mcenro  mauresmo  ameli  hip  fiveset  mario  ancic  \
0    ...      0       0         0      0    0        0      0      0
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    ...      0       0         0      0    0        0      0      0
..   ...    ...     ...       ...    ...  ...      ...    ...    ...
180  ...      0       0         0      0    0        0      0      0
181  ...      0       0         0      0    0        0      0      0
182  ...      0       0         0      0    0        0      0      0
183  ...      0       0         0      0    0        0      0      0
184  ...      0       0         0      0    0        0      0      0

     lundgren  class_label
0           0            4
1           0            2
2           0            1
3           0            2
4           0            4
..        ...          ...
180         0            2
181         0            0
182         0            4
183         0            2
184         0            2

[185 rows x 4614 columns]
```

VAL DATA: Class distribution graph

In [9]:
```
sports_val_df.class_label.value_counts().plot(kind='bar')
```

Out[9]:
```
<AxesSubplot:>
```



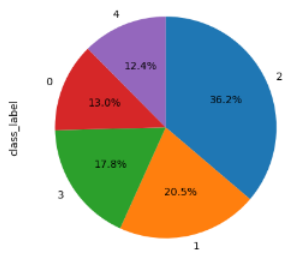
In [10]:
```
sports_val_df.class_label.value_counts()
```

Out[10]:
```
2    67
1    38
3    33
0    24
4    23
Name: class_label, dtype: int64
```

VAL DATA: Class distribution chart

In [11]:
```
sports_val_df.class_label.value_counts().plot(kind='pie', autopct='%1.1f%%',counterclock=False,startangle=90)
```

Out[11]:
```
<AxesSubplot:ylabel='class_label'>
```



STEP 2: PREPARING MODEL COMPONENTS

PI MLE

In [12]:
```
import numpy as np
pd.options.mode.chained_assignment = None
def piArray(dataset, label):
    # returns est. P(Y = yk)
    arr = np.array(dataset[label].value_counts().sort_index() / len(dataset))
    return arr
```

In [13]:
```
piArray(sports_train_df, "class_label" )
```

Out[13]:
```
array([0.13949275, 0.1557971 , 0.35869565, 0.20652174, 0.13949275])
```

THETA MLE

In [14]:
```
def thetaArray(dataset, label):
    # returns est. P(Xj |Y = yk)
    arrO = np.array(dataset.groupby([label]).sum())
    arrO = np.divide(arrO.T, np.sum(arrO,axis=1) ).T
    return arrO
```

In [15]:
```
dfTry = pd.DataFrame({'word1': [1, 5, 1, 3],
                      'word2': [3, 7, 1, 4],
                      'class_label': [1, 0, 1, 0]})
```

In [16]:
```
print(dfTry)
```

```
   word1  word2  class_label
0      1      3            1
1      5      7            0
2      1      1            1
3      3      4            0
```

In [17]:
```
marr = thetaArray(dfTry, 'class_label' )
print(marr)
```

```
[[0.42105263 0.57894737]
 [0.33333333 0.66666667]]
```

CLASSIFICATION AND ACCURACY CALCULATION

In [18]:
```
def classify(allPredictions):
    argMax = np.array(allPredictions.argmax(axis=1))
    return argMax

def accuracyResults(validLabels, validPredictedLabels):
    numTrue = 0
    for gtL, predL in zip(validLabels, validPredictedLabels):
        if gtL == predL:
            numTrue += 1
    numFalse = validLabels.size - numTrue
    print("The number of correct predictions are", numTrue)
    print("The number of incorrect predictions are", numFalse)
    accuracy = (numTrue/len(validLabels))
    print("The accuracy is %", accuracy * 100)
```

STEP 3: MULTINOMIAL NB MODEL

In [19]:
```
class MultinomialNBClassifier:

    def __init__(self):
        self.myPi = None
        self.myTheta = None

    def fit(self, df_test, label):
        self.myPi = piArray(df_test, label ) #[class0prob, clas1prob....]
        self.myTheta = thetaArray(df_test, label) #[[w1c0 w2c0...][w1c1..].....]

    def printer(self):
        print(self.myPi)
        print(self.myTheta)

    def predict(self, df_val):
        val_arr = np.array(df_val)
        val_arr_x = val_arr[:,:-1]
        val_arr_y = val_arr[:,-1]

        with np.errstate(divide='ignore'):
            dummy = np.log10(self.myTheta)
        dummy[dummy == -np.inf] = -1e308
        dummy = dummy.T
        sumTerm = np.matmul(val_arr_x, dummy)
        allPredictions =  np.add(np.log10(self.myPi), sumTerm)
        predictedLabels = classify(allPredictions)
        return predictedLabels

    def confusionMatrix(self, df, label):
        labelVar = df[label]
        predictions = self.predict(df)
        labelVarNum = labelVar.nunique() # number of class variations
        # class x class matrix
        confusionMatrix = np.zeros(shape=(labelVarNum, labelVarNum))
        for pred, exp in zip(predictions, labelVar):
            confusionMatrix[pred][exp] += 1
        return confusionMatrix
```

MULTINOMIAL NB TEST RESULTS

In [20]:
```
classifier = MultinomialNBClassifier()
classifier.fit(sports_train_df, 'class_label')
classifier.printer()
pred = classifier.predict(sports_val_df)
print(pred)
accuracyResults(np.array(sports_val_df['class_label']), pred)
classifier.confusionMatrix(sports_val_df,'class_label' )
```

```
[0.13949275 0.1557971  0.35869565 0.20652174 0.13949275]
[[8.64154857e-04 8.64154857e-05 3.19737297e-03 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 2.32842424e-04 8.20769544e-03 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 6.34256176e-05 3.86896267e-03 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 5.97446398e-03 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 9.42785840e-03 ... 2.77289953e-04
  2.77289953e-04 2.77289953e-04]]
[0 0 0 0 4 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 3 0 0
 0 2 1 2 0 0 4 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 4 0 0 2 0 0 0 0 0 2 0 0 0 0 0
 0 0 0 0 4 0 0 0 0 2 0 0 0 0 0 0 0 0 0 2 0 0 2 1 0 0 0 0 0 0 0 0 0 0 1 1 0 2
 0 0 0 0 0 0 0 2 0 0 3 0 0 0 0 2 0 0 0 0 2 0 3 0 0 1 0 0 0 0 0 0 0 2 0 0 0 2
 0 0 0 0 0 2 0 0 0 2 2 0 0 0 2 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
The number of correct predictions are 58
The number of incorrect predictions are 127
The accuracy is % 31.351351351351354
```

Out[20]:
```
array([[24., 33., 45., 30., 19.],
       [ 0.,  5.,  0.,  0.,  0.],
       [ 0.,  0., 22.,  0.,  0.],
       [ 0.,  0.,  0.,  3.,  0.],
       [ 0.,  0.,  0.,  0.,  4.]])
```

STEP 4: EXTENDING MULTINOMIAL NB MODEL

```
def thetaArrayExtended(dataset, label, alpha):
    # returns est. P(Xj |Y = yk)
    arr0 = np.array(dataset.groupby([label]).sum())
    arr0 = arr0 + alpha
    arr0 = np.divide(arr0.T, np.sum(arr0,axis=1) ).T
    return arr0

class MultinomialNBClassifierExtended:

    def __init__(self):
        self.myPi = None
        self.myTheta = None

    def fit(self, df_test, label, alpha):
        self.myPi = piArray(df_test, label) #[class0prob, clas1prob....]
        self.myTheta = thetaArrayExtended(df_test, label, alpha) #[[w1c0 w2c0...][w1c1..]........]

    def printer(self):
        print(self.myPi)
        print(self.myTheta)

    def predict(self, df_val):
        val_arr = np.array(df_val)
        val_arr_x = val_arr[:,:-1]
        val_arr_y = val_arr[:,-1]

        #with np.errstate(divide='ignore'):
        dummy = np.log10(self.myTheta)
        #dummy[dummy == -np.inf] = -10000000000
        dummy = dummy.T
        sumTerm = np.matmul(val_arr_x, dummy)
        allPredictions =  np.add(np.log10(self.myPi), sumTerm)
        predictedLabels = classify(allPredictions)
        return predictedLabels

    def confusionMatrix(self, df, label):
        labelVar = df[label]
        predictions = self.predict(df)
        labelVarNum = labelVar.nunique() # number of class variations
        # class x class matrix
        confusionMatrix = np.zeros(shape=(labelVarNum, labelVarNum))
        for pred, exp in zip(predictions, labelVar):
            confusionMatrix[pred][exp] += 1
        return confusionMatrix
```

EXTENDED MULTINOMIAL NB TEST RESULTS

```
classifierExtended = MultinomialNBClassifierExtended()
classifierExtended.fit(sports_train_df, 'class_label', alpha = 1 )
predExtended = classifierExtended.predict(sports_val_df)
print(predExtended)
accuracyResults(np.array(sports_val_df['class_label']), predExtended)
classifierExtended.confusionMatrix(sports_val_df,'class_label' )
```

```
[4 2 1 2 4 0 0 1 2 2 1 3 3 1 0 3 1 2 3 3 3 1 2 1 3 2 3 4 3 2 3 2 1 1 3 2 2
 1 2 1 2 0 1 4 3 4 4 4 2 2 2 4 2 4 0 1 0 1 2 4 0 0 2 0 3 2 1 0 2 3 0 4 0 0
 3 1 2 2 4 2 2 2 2 2 3 4 2 2 4 0 2 0 2 4 3 2 1 1 0 1 2 2 2 1 2 1 1 1 1 3 2
 1 4 0 3 2 2 3 2 2 1 3 0 1 2 3 2 3 1 1 4 2 1 3 4 3 1 2 2 3 2 3 2 2 4 3 0 2
 3 4 3 3 3 3 2 1 2 0 2 2 2 0 3 2 2 3 2 2 1 2 0 3 1 1 2 0 0 4 2 2 3 2 0 4 2 2]
The number of correct predictions are 180
The number of incorrect predictions are 5
The accuracy is % 97.2972972972973

array([[24.,  0.,  0.,  0.,  1.],
       [ 0., 35.,  0.,  0.,  0.],
       [ 0.,  1., 66.,  0.,  0.],
       [ 0.,  2.,  1., 33.,  0.],
       [ 0.,  0.,  0.,  0., 22.]])
```

Notes in regard to coding part: To mimic number -∞, the arbitrarily small value -1e308 is used as it is the assigned value for -np.inf in NumPy.

Q2.4

Comparing the two models you trained, how does the Dirichlet prior α effects your model? Also, interpret the structure of the dataset. Given that the dataset does not include stop words, why are the two models different? Explain by giving references to your results. You can also benefit from the statistical structure of the feature matrix.

The results demonstrate that Dirichlet prior α contributes to a remarkable effect in terms of greatly improving the accuracy of our classifier. It essentially smoothens the cases where a word did not occur in each class to save it from log0(-∞) effect. Essentially in the first part I also made some sort of smoothening by using an arbitrarily small number. In a sense it was a similar application but with a much smaller α value. The dataset structure also plays a role in the accuracy difference observed in my test results. Given dataset has very frequent words overall. Figure 4 shows that in test dataset, most occurred frequency in class 0 is 143, class 1 is 198, class 2 is 278, class 4 is 296 and class 4 is 136 while all of them has 0 frequency words. This occurrence frequency difference results in sharp differences among $P(X_j \mid Y = y_k)^{twj,i}$ values. *MultinomialNBCLassifierExtended* reduces those sharp differences and modifies the statistical structure of test dataset so that the probabilities become more evenly distributed among the dataset. Actually, the secret the success of higher accuracy values of the extended model as opposed to the old model is changing the statistical structure of test dataset. It increases the probability of less occurring words and eliminates zero probability cases while decreasing the probability of higher occurring words .

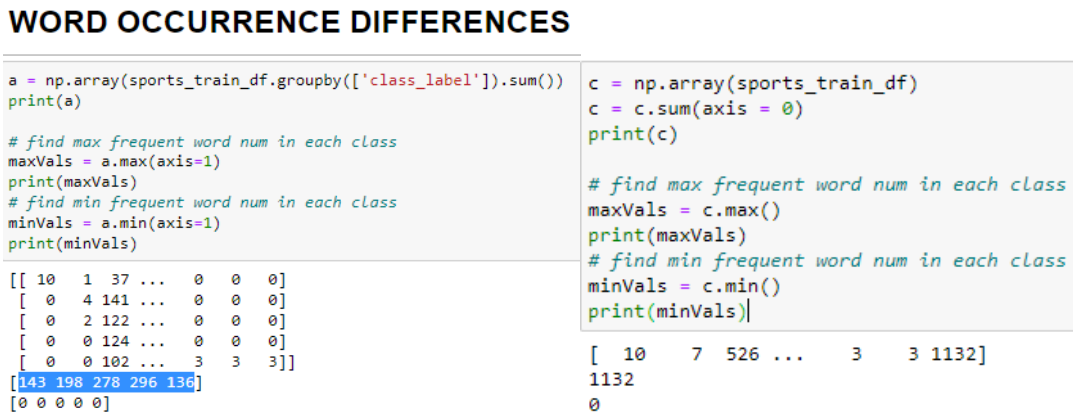


Figure 4: Word occurrence frequencies among classes and in general of test dataset