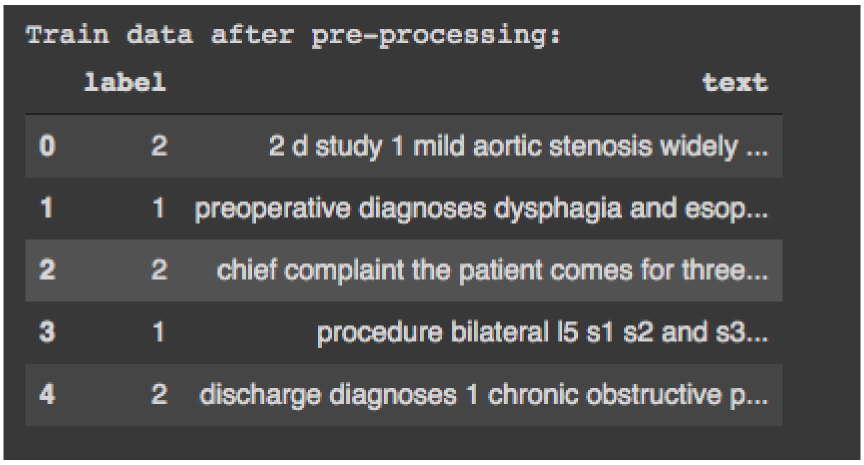
Assignment 3- ML

Zahra Parham 2122841

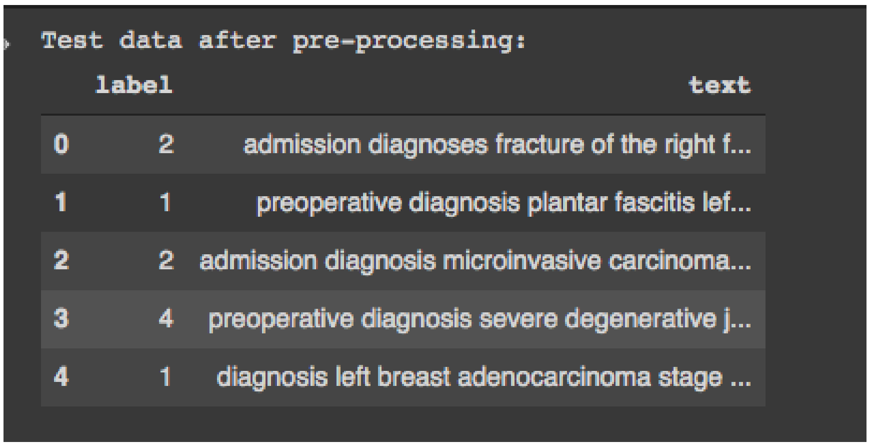
1.

In the first question, it asks for **Binary-bag-of-words** and **frequency-bag-of-word:**

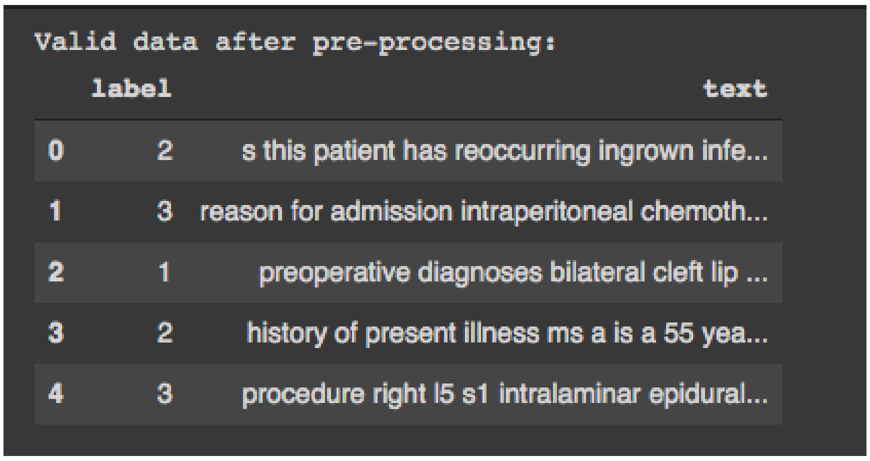
The first step is pre-processing the data by lowercase every word and removing punctuation from each corpus.



```
Train data after pre-processing:
   label                                          text
0      2          2 d study 1 mild aortic stenosis widely ...
1      1  preoperative diagnoses dysphagia and esop...
2      2     chief complaint the patient comes for three...
3      1              procedure bilateral l5 s1 s2 and s3...
4      2   discharge diagnoses 1 chronic obstructive p...
```



```
Test data after pre-processing:
   label                                            text
0      2   admission diagnoses fracture of the right f...
1      1       preoperative diagnosis plantar fascitis lef...
2      2   admission diagnosis microinvasive carcinoma...
3      4   preoperative diagnosis severe degenerative j...
4      1     diagnosis left breast adenocarcinoma stage ...
```



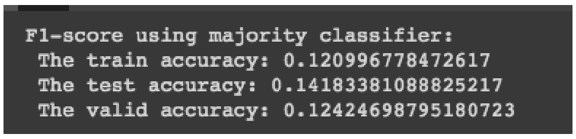
```
Valid data after pre-processing:
   label                                            text
0      2      s this patient has reoccurring ingrown infe...
1      3   reason for admission intraperitoneal chemoth...
2      1          preoperative diagnoses bilateral cleft lip ...
3      2       history of present illness ms a is a 55 yea...
4      3     procedure right l5 s1 intralaminar epidural...
```

The **second step** is creating a word vocabulary. After that, submitting the word vocabulary and test/train/valid dataset based on the instruction is given. All these datasets are located in the repository.

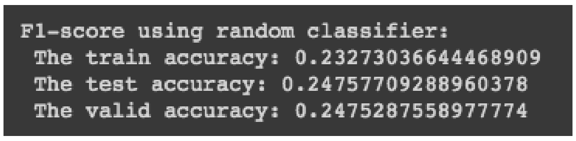
The Third Step is to convert the dataset to vector input based on two representations: a binary bag of words and a frequency bag of words

2. Binary bag-of-words:

2. a



```
F1-score using majority classifier:
 The train accuracy: 0.120996778472617
 The test accuracy: 0.1418338108825217
 The valid accuracy: 0.12424698795180723
```



```
F1-score using random classifier:
 The train accuracy: 0.23273036644468909
 The test accuracy: 0.24757709288960378
 The valid accuracy: 0.2475287558977774
```

2. b

All the required task is implemented, and you can see the result as follow:

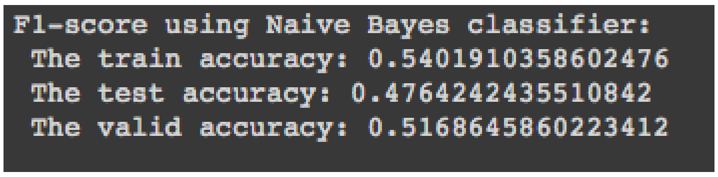
2. c

- **Bernoulli naive Bayes** has just one hyperparameter: **alpha which controls smoothing**
    - **alpha range: [0.01, 1.0] by step size = 0.01**, **Best parameter: {alpha = 0.18}**
- **Decision Tree:** The decision Tree has a lot of hyperparameters for tuning, so at first, I just started from **max_depth.**
    - **Criterion:** Decision Tree nodes are split by using impurity which is a measure of the homogeneity of the labels on a node. for impurity measure scikit-learn is implement the information gain(entropy) and gini.
        - **criterion range: ['gini', 'entropy']**, **Best parameter:{'gini'}**
    - **ccp_alpha:** is one types of pruning of Decision Trees which It makes it versatile so that it can adapt if we feed any new kind of data to it, thereby fixing the problem of overfitting. It reduces the size of a Decision Tree which might slightly increase your training error but drastically decrease your testing error, hence making it more adaptable.
        - **ccp_alpha** range: **np.arange(0.0, 1.0, 0.01)**, **Best parameter: {'ccp_alpha': 0.0009}**
    - **max_depth**: at first, I let the model decide the max_depth at first, and then by comparing the train and test scores, I look for underfitting or overfitting, and depending on the degree, I decrease or increase the max_depth.
        - **max_depth range: np.arange(1, 25), Best parameter: {'max_depth': 45}**
    - **min_samples_split and min_samples_leaf**: min_samples_split talk about the internal node, and min_samples_leaf talk about the external nodes. Both of them are used to control over-fitting. min_samples_leaf is always guaranteed no matter the min_samples_split value; according to the paper, the ideal min_samples_splits values tend to be between 1 to 40. Based on a paper, the ideal values for min_samples_split is between 1 to 20.
        - **min_samples_split range: np.arange(1,10),Best parameter: {'min_samples_split': 4}**

        - **min_samples_leaf range: np.arange(1, 10),Best parameter: {'min_samples_leaf': 4}**
    - **max_features**: use of max_features is to limit overfitting, by choosing a reduced number of features we can increase the stability of the tree and reduce variance and over-fitting.
        - **max_features range: np.arange(0.1, 0.5, 0.1),Best parameter: {'max_features': 0.4}**
- **Logistic regression:** Logistic regression does not really have any critical hyperparameters to tune. The C parameter controls the penality strength.
    - **C range: [0.001, 0.01, 0.1, 1, 10, 100,150,200, 1000]**, **Best Parameter: {'C': 100}**
    - **max_iter**: **[5000]**
- **Linear SVM:**
    - **C:** is the penalty parameter of the error term. It controls the trade-off between smooth decision boundary and classifying the training points correctly. Increasing C values may lead to overfitting the training data
        - **C range: [0.1, 1, 10, 100, 1000], Best parameter:{'C' : 1}**
    - **max_iter:** is the number of iteration for the algorithm. at first, the max_iter was low, and I got the warning that the algorithm did not converge, so I decided to increase the max_iter
        - **max_iter**: **[5000]**

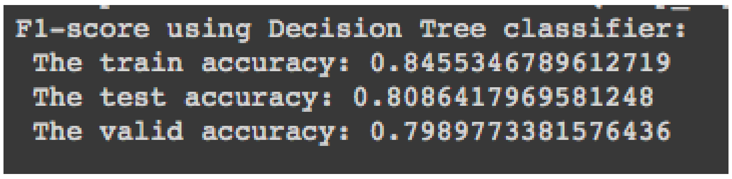
2.d

F1 score for Bernoulli Naive Bayes classifier is:



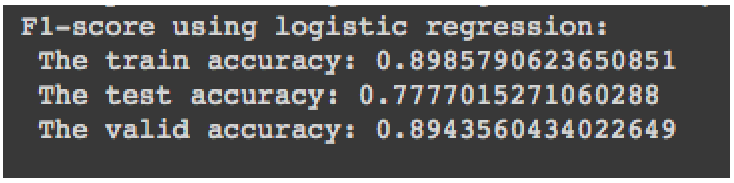
```
F1-score using Naive Bayes classifier:
 The train accuracy: 0.5401910358602476
 The test accuracy: 0.4764242435510842
 The valid accuracy: 0.5168645860223412
```

F1 score for Decision Tree classifier is:



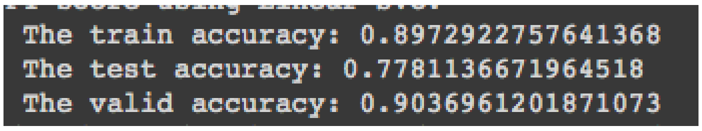
```
F1-score using Decision Tree classifier:
 The train accuracy: 0.8455346789612719
 The test accuracy: 0.8086417969581248
 The valid accuracy: 0.7989773381576436
```

F1 score for Logistic regression classifier is:



```
F1-score using logistic regression:
 The train accuracy: 0.8985790623650851
 The test accuracy: 0.7777015271060288
 The valid accuracy: 0.8943560434022649
```

F1 score for Linear SVM classifier is:



```
 The train accuracy: 0.8972922757641368
 The test accuracy: 0.7781136671964518
 The valid accuracy: 0.9036961201871073
```

2. e

Comment on the performance of different classifiers.

This is ranking of these classifier based on their F1-score on test dataset:

1- Decision Tree , Logistic regression, SVM classifier (which the score of all these algorithm are quite close to each other), Bernouli Naive Bayes.

Decision Tree perform the best between all these algorithm, cause it's tuning really well and also it could handle the curse of dimesionality very well

Naive Bayes has poor performance since it treats all features as being independent of one another so it capture much more simpler mappings between the input variables and the output classes although it make this algorithm much quicker compare to the other algorithm and it can work well on the high dimension datasets like here if the feature would be independent. Naive Bayes assumes that all predictors (or features) are independent, rarely happening in real life. This limits the applicability of this algorithm in real-world use cases.

Decision tree: in our dataset the decision tree is performing really well. at first it was overfitting but by tuning the hyper parameter it perform better. DT can make trees with nods even with absent or presence of the data. and also DT suffer from curse of dimensionality since we have it here because the number of features >> #data

Logistic regression: works really well. logistic regression is used in to understand the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic regression equation.

SVM: perform well too but it can be better the problem with this data here is:

curse of dimensionality: the number of dimensions exceeding the number of examples and SVM are affected by the lack of examples and too many features.

although SVM can really handle the curse of dimensionality very well and also it is a good algorithm for text mining

Why did a particular classifier perform better than the rest?

The difference between the classifier performance is related to the type of the data we have. The number of data for training, the number of feature we have, the independency of the features, the ratio of the number of features to the data(curse of dimensionality), type of the data(categorical, numerical), missing data. Linearity(for example Linear regression algorithms assume that data trends follow a straight line. If the data is linear, then these algorithms perform quite good. on the other hand the SVM can handle high dimensional and complex data structures.)

What was the role of the hyper-parameters in finding the best results.

the hyper parameters help us to preventing the model from over fitting or under fitting.

also control the overall behaviour of a machine learning model. The ultimate goal is to find an optimal combination of hyperparameters that minimizes a predefined loss function to give better results.

3. Frequency bag-of-words:

3. a

All the required task is implemented, and you can see the result as follow:

3. b

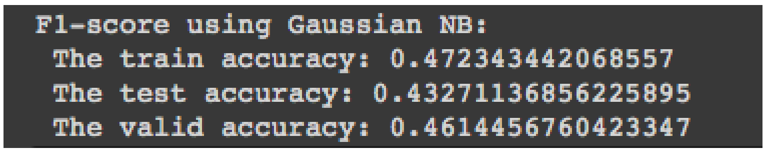
- **Gaussian naive Bayes**
    - **var_smoothing range: np.arange(0.01, 6.0, 0.01)**, **Best_parameter: {'var_smoothing': 0.02}**

- **Decision Tree:** The decision Tree has a lot of hyperparameters for tuning so at first I just started from **max_depth.**
    - **Criterion:** Decision Tree nodes are split by using impurity which is a measure of the homogeneity of the labels on a node. for impurity measure scikit-learn is implement the information gain(entropy) and gini.
        - **criterion range: ['gini', 'entropy']**, **Best parameter:{'entrpy'}**
    - **ccp_alpha:** is one types of pruning of Decision Trees which It makes it versatile so that it can adapt if we feed any new kind of data to it, thereby fixing the problem of overfitting. It reduces the size of a Decision Tree which might slightly increase your training error but drastically decrease your testing error, hence making it more adaptable.
        - **ccp_alpha** range: **np.arange(0.0, 1.0, 0.01)**, **Best parameter: {'ccp_alpha': 0.003}**
    - **max_depth**: at first, I let the model decide the max_depth at first, and then by comparing the train and test scores I look for underfitting or overfitting, and depending on the degree I decrease or increase the max_depth.
        - **max_depth range: np.arange(1, 27), Best parameter: {'max_depth': 22}**
    - **min_samples_split and min_samples_leaf**: min_samples_split talk about the internal node and min_samples_leaf talk about the external nodes. Both of them are used to control over-fitting. min_samples_leaf is always guaranteed no matter the min_samples_split value, according to the paper the ideal min_samples_splits values tend to be between 1 to 40. Based on a paper the ideal values for min_samples_split is between 1 to 20.
        - **min_samples_split range: np.arange(1,10),Best parameter: {'min_samples_split': 4}**
        - **min_samples_leaf range: np.arange(1, 10),Best parameter: {'min_samples_leaf': 4}**
    - **max_features**: use of max_features is to limit overfitting, by choosing a reduced number of features we can increase the stability of the tree and reduce variance and over-fitting.
        - **max_features range: np.arange(0.1, 0.5, 0.1),Best parameter: {'max_features': 0.4}**

- **Logistic regression:** Logistic regression does not really have any critical hyperparameters to tune. The C parameter controls the penality strength.
    - **C range: [0.001, 0.01, 0.1, 1, 10, 100,150,200, 1000]**, **Best Parameter: {'C': 150}**
    - **max_iter**: **[10000]**
- **Linear SVM:**
    - **C:** is the penalty parameter of the error term. It controls the trade-off between smooth decision boundary and classifying the training points correctly. Increasing C values may lead to overfitting the training data
        - **C range: [0.1, 1, 10, 100, 1000], Best parameter:{'C': 10000}**
    - **max_iter:** is the number of iteration for the algorithm. at first the max_iter was low and I got the warning that the algorithm did not converge so I decided to increase the max_iter
        - **max_iter**: **[10000]**

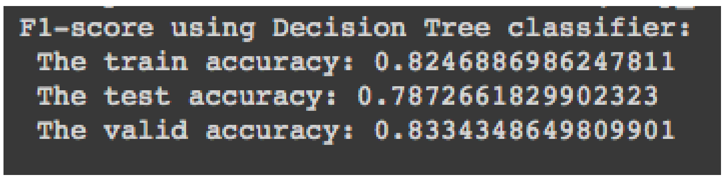
3.c

F1 score for Gaussian Naive Bayes classifier is:



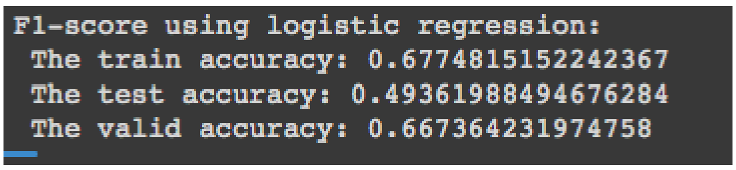
```
F1-score using Gaussian NB:
 The train accuracy: 0.472343442068557
 The test accuracy: 0.43271136856225895
 The valid accuracy: 0.4614456760423347
```

F1 score for Decision Tree classifier is:



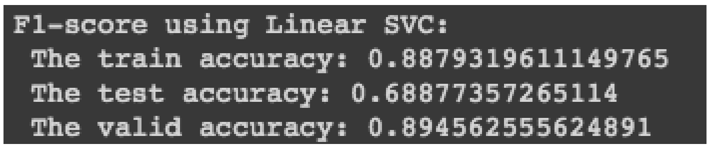
```
F1-score using Decision Tree classifier:
 The train accuracy: 0.8246886986247811
 The test accuracy: 0.7872661829902323
 The valid accuracy: 0.8334348649809901
```

F1 score for Logistic regression classifier is:



```
F1-score using logistic regression:
 The train accuracy: 0.6774815152242367
 The test accuracy: 0.49361988494676284
 The valid accuracy: 0.667364231974758
```

F1 score for Linear SVM classifier is:



```
F1-score using Linear SVC:
 The train accuracy: 0.8879319611149765
 The test accuracy: 0.68877357265114
 The valid accuracy: 0.894562555624891
```

3. d

Comment on the performance of different classifiers.

The factor that effect on the performance of classifier:

1. removing the punctuation and also replace punctuation with space really helped
2. in FBOW it measures the frequency of the word so this would be useful for consider the weighting for each word in the conpus so we can understand the importance of each word
3. the number of features exceeds the number of examples in training set. this will affect the algorithm

This is ranking of these classifier based on their F1-score on test dataset:

1- Decision Tree , Logistic regression, SVM classifier (which the score of all these algorithm are quite close to each other), Bernouli Naive Bayes.

Decision Tree perform the best between all these algorithm, cause it's tuning really well and also it could handle the curse of dimesionality very well. DT can make trees with nods even with absent or presence of the data. Decision tree's hyper parameter really helped to achieve this score. for example: ccp_alpha help to pruning the Tree which it's lead to overcome overfitting.

Naive Bayes has poor performance since it treats all features as being independent of one another so it capture much more simpler mappings between the input variables and the output classes although it make this algorithm much quicker compare to the other algorithm and it can work well on the high dimension datasets like here if the feature would be independent. Naive Bayes assumes that all predictors (or features) are independent, rarely happening in real life. This limits the applicability of this algorithm in real-world use cases.

Decision tree: in our dataset the decision tree is performing really well. at first it was overfitting but by tuning the hyper parameter it perform better. DT can make trees with nods even with absent or presence of the data. and also DT suffer from curse of dimensionality since we have it here because the number of features >> #data

Logistic regression: works poorly in this dataset. logistic regression is used in to understand the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic
regression equation.

SVM: perform well too but it can be better the problem with this data here is:

curse of dimensionality: the number of dimensions exceeding the number of examples and SVM are affected by the lack of examples and too many features.

Why did a particular classifier perform better than the rest?

The difference between the classifier performance is related to the type of the data we have. The number of data for training, the number of feature we have, the independency of the features, the ratio of the number of features to the data(curse of dimensionality), type of the data(categorical, numerical), missing data. Linearity(for example Linear regression algorithms assume that data trends follow a straight line. If the data is linear, then these algorithms perform quite good. on the other hand the SVM can handle high dimensional and complex data structures.)

What was the role of the hyper-parameters in finding the best results.

the hyper parameters help us to preventing the model from over fitting or under fitting.

also control the overall behaviour of a machine learning model. The ultimate goal is to find an optimal combination of hyperparameters that minimizes a predefined loss function to give better results.

3.e

Compare the performance with the binary bag-of-words based classifiers. Why is
there a difference in the performance?

Bag of Words just creates a set of vectors containing the existence of word occurrences in the document (reviews), while the frequency bag of words model contains information on the more important words and the less important ones as well.

here the FBOW did not perform as well as BBOW and I think it because we did not have enough training data and also if we reduce the number of features it would perform better.

Give a brief explanation comparing BBoW Naive Bayes and FBoW Naive Bayes and similarly for other models.

The performance of BBoW for Naive Bayes is better than FBow but not that much difference, I think this is happening because the number of training data it's not that much.

I expected that FBoW performs better since it store additional information for itself, here the FBOW did not perform as well as BBOW and I think it because we did not have enough training data and also if we reduce the number of features it would perform better.

to conclude both Fbow and BBow did not performs well for Naive bayes because the features are not independent in our dataset.

except logistic regression the performance of all the algorithm in FBow and BBoW are the same and again I think because of the lack of training dataset and also the number of feature we have, it was not possible for FBoW to use it's additional information.

3.f

Overall the FBoW store additional information which is the importance of each feature compare to our dataset but here it could not use it's additional information since the training data was not that much and also the number of feature compare to the number of training was quite a lot. because of that the performance of these two algorithm are quite the same.

References:

https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680