ASSIGNMENT 3

SPAM OR HAM

➢ Dataset :

- The dataset I used is the Spam Email Data**(emails.csv)** set ,Which is Publicly Available on Kaggle.
- The data can be found at the following link DataSet.

- **Dataset Name: Spam Email Dataset (emails.csv)**

- **Description:**
  This dataset contains 5728 email text messages, labeled as either spam or not spam. Each email message is associated with a binary label, where "1" indicates that the email is spam, and "0" indicates that it is not spam. The dataset is intended for use in training and evaluating spam email classification models.

  The Columns of the dataset are:

- **text (Text): This column contains the text content of the email messages. It includes the body of the emails along with any associated subject lines or headers.**
- **spam_or_not (Binary): This column contains binary labels to indicate whether an email is spam or not. "1" represents spam, while "0" represents non-spam.**

➢ Preprocessing :

- For Preprocessing I implemented a function preprocess(text).

```
def preprocess(text) :

    text = text.replace("Subject",'')
    text=re.sub(r'[^a-zA-Z\s]','',text)

    text=text.lower()
    return text
```

The Function performs the following.

- text = text.replace("Subject",'')
  This line removes the word “Subject” from the emails.txt
  This is done because “Subject” is a common word in the data set that does not provide any information.

- text=re.sub(r'[^a-zA-Z\s]','',text)
  This line removes all non alphabetic characters from emails text by using a regular expression.

- text=text.lower()
  This line converts all the text to lowercase.(example : Hi and hi are treated as same)

- Preprocessing is done for standardizing the text (before extracting features) so that the models can focus on the most relevant features for classification.

➢ Feature extraction :

- I have Trained Data with 3 models.(Naive-Bayes,Logistic regression,SVM).

1)

- For the case of Naive-Bayes I used the `CountVectorizer from sklearn.feature_extraction.text.`
  Below is a code snippet.

```
vectorizer = CountVectorizer(binary=True,stop_words="english")
vectorizer.fit(train_emails)
train_vectors=vectorizer.transform(train_emails)
train_vectors=train_vectors.toarray()
train_vectors=np.array(train_vectors)
```

In Summary Each email is transformed into a binary vector where each feature represents a word from the training data.(the value of the feature is 1 if the word is present in the email
else 0).

2)

- For the case of Logistic Regression I used the `TfidfVectorizer from sklearn.feature_extraction.text.`

```
vectorizer2=TfidfVectorizer(stop_words="english")
vectorizer2.fit(emails)
LR_train_vectors=vectorizer2.transform(emails)
LR_train_vectors=LR_train_vectors.toarray()
LR_train_vectors=np.array(LR_train_vectors)
```

In Summary Each email is transformed into a vector where each feature represents a word from the training data(the value of each feature is set based on how important is the word to this email relative to all other emails).

3)

- For the case of SVM I used the CountVectorizer from sklearn.feature_extraction.text.(Binary=false)

```
vectorizer3=CountVectorizer(stop_words="english")
vectorizer3.fit(emails)
SVM_train_vectors=vectorizer3.transform(emails)
SVM_train_vectors=SVM_train_vectors.toarray()
SVM_train_vectors=np.array(SVM_train_vectors)
```

In summary it is similar to the First One exceptBinary= False in this case i.e(the value of the feature is the number of times the word appeared in the mail).

➢ Classifiers :

1) Naive-Bayes

$$\hat{p} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$\hat{p}_j^y = \frac{\sum_{i=1}^{n} \mathbb{1}(f_j^i = 1, y_i = y)}{\sum_{i=1}^{n}(y_i = y)}$$

$\hat{p}$ is the probability that an email is spam

i.e Equal to the
fraction of spam emails.

$\hat{p}_j^y$ is the probability that jth word appeared in y labeled email

i.e Equal to the fraction of y-labeled emails containing jth word.

From Bayes rule we get

$$P(y^{test} = 1|x^{test}) \propto \left(\prod_{k=1}^{d} (\hat{p}_k^1)^{f_k}(1-\hat{p}_k^1)^{1-f_k}\right) \cdot \hat{p}$$

$$P(y^{test} = 0|x^{test}) \propto \left(\prod_{k=1}^{d} (\hat{p}_k^0)^{f_k}(1-\hat{p}_k^0)^{1-f_k}\right) \cdot (1-\hat{p})$$

If $P(y^{test} = 1|x^{test}) >= P(y^{test} = 0|x^{test})$ predict 1 else predict 0.

As the values of probabilities are small and we are multiplying nearly 30,000 it is better to compare by taking logs .

$$\sum_{i=1}^{d} f_i \log\left(\frac{\hat{p}_i^1(1-\hat{p}_i^0)}{\hat{p}_i^0(1-\hat{p}_i^1)}\right) + \left(\sum_{i=1}^{d} \log\left(\frac{1-\hat{p}_i^1}{\hat{p}_i^0}\right)\right) + \log\left(\frac{\hat{p}}{1-\hat{p}}\right) \geq 0$$

The above expression can be written in the form $w^T x + b >= 0$.

If the above expression is satisfied then predict $y^{test} = 1$.

2) Logistic regression

$$\nabla \log \mathcal{L}(w) = \sum_{i=1}^{n} x_i \left(y_i - \frac{1}{1 + \exp(-w^T x_i)} \right)$$

$$w_{t+1} = w_t + \eta \cdot \nabla \log \mathcal{L}(w_t)$$

After running algorithm for certain number of iterations we get w_{LR}.

If $w_{LR}^{T} x^{test} >= 0 >$ predict $y^{test} = 1$ otherwise $y^{test} = 0$.

Observations :

- For Tuning Hyperparameters (in this case step size) I split the data into two parts in 4 :1 ratio, say train-split and test-split. Apply logistic regression for train-split and evaluate the accuracy (percentage of labels predicted correctly for both train-split and test-split. (splitted from the original data)).

- Below is the table showing the accuracy.

Feature extraction using(vectorizer used)	Step size	time	Accuracy
			test-split
Count	1e-4	60 s	99.56%
	1e-5	60 s	97.64%
	1e-6	60 s	86.12%
Tfidf	1	10 s	98.86%
	1e-1	10 s	99.47%
	1e-2	10 s	99.12%

- It is Quite clear that the Tfidf one runs faster than the Countvectorizer and also has better accuracy. This is the reason we choose Tfidf over Countvectorizer for this case.

- We Choose **step size to be 1e-1** in case of Tfidf as it is the best in that case.

Note : Splitting of original data is done only for tuning purposes.

- We will predict the test mails by training the model with entire data set.

3) SVM(Support vector machine)

```
from sklearn import svm
vectorizer3=CountVectorizer(binary=False,stop_words="english")
vectorizer3.fit(emails)
SVM_train_vectors=vectorizer3.transform(emails)
SVM_train_vectors=SVM_train_vectors.toarray()
SVM_train_vectors=np.array(SVM_train_vectors)
svm_classifier=svm.LinearSVC(C=1.0,max_iter=10000,dual=False)
svm_classifier.fit(SVM_train_vectors,labels)
SVM_train_predictions=svm_classifier.predict(SVM_train_vectors)
```

Observations :

- The Hyper parameter **C=1.0** is chosen for getting good results.
- Here Also We will predict the test mails by training the model with entire data set.(that is splitting is done only for tuning purposes)

➢ Final Classifier :

- Train All the Above Models Naive-Bayes,Logistic regression,SVM with the Entire Data set(emails.csv)

- Take step size and C value based on the Observations mentioned above.

- The Code reads all the Files named with “emails(p).txt” present in the Folder named “test” present in the Current directory.

- The Code predicts the label for each of the emails read by using the 3 models and outputs the majority of three predicted labels by above models.

- That is Final predicted label = Mode(predicted labels by the Above 3 Models).