Name: Rhugaved Narmade
RRN210004

Google Collab Link:
https://colab.research.google.com/drive/1-nUscinjLWcnvtAIseBosODmhUQCeYU8?usp=sharing

To compile and run the code:
Visit the google collab link. Run the cells one by one following the instructions written over the code cell.

(You need to upload the dataset files in ZIP format as is that were given to us after running a cell that creates the folder "/content/rhugaved_data". Upload the zip for all 3 datasets in one go and then continue with the execution of cells)

Code Explanation:
Functions:
get_text_from_dataset(dataset_folders, data): Opens the extracted files and stores them in data which is a dictionary for all the 3 datasets

clean_text_document(text): Clean the original text from data. Lowers the text, removes symbols, numbers, and extra spaces, and removes hyperlinks, and emails. It also removes stopwords as they don't give any meaningful data

vocab_builder(all_text): Builds the vocabulary for a particular dataset

bow_builder(local_all_text, local_vocab): Builds the bag of words model from text and vocab of a particular dataset

get_cond_prob_multi(class_bow, class_all_text, vocab): returns the conditional probability for mulinomial naive bayes model

apply_multi_NB(class_prior, cond_prob, vocab, doc): Applies the multinomial naive bayes to a given document and returns the probability of ham or spam

multi_NB(data): primary function for multinomial naive Bayes. Applies the above functions on each dataset.

bernoulli_builder(local_all_text, local_vocab): Builds the Bernoulli model from text and vocab of a particular dataset

get_cond_prob_bernoulli(class_bernoulli_doc_occurances, class_all_text, vocab): returns the conditional probability for bernoulli naive bayes model

apply_bernoulli_NB(class_prior, cond_prob, vocab, doc): Applies the Bernoulli naive Bayes to a given document and returns the probability of ham or spam

bernoulli_NB(data): primary function for bernoulli naive bayes. Applies the above functions on each dataset.

bow_bernoulli(local_all_text, local_vocab): calculates bag of words and Bernoulli for logistic regression. Returns: bag_of_words_corpus, bernoulli_corpus

class **log_regression**(): For logistic regression functions.

__init__(self, lr = 0.1, max_iter = 64, reg_para = 0.01): takes parameters like learning rate, maximum iterations, and lamba/regularization parameter

fit(self, X, Y): Used for the training of X. Applies gradient descent in it and optimizes weights

predict(self, X): Used for predicting Y given X

lr_for_lambda(data, lambda_list): logistic regression function in which we use the validation set to look for the best lambda value. We create the bow_bernoulli models in this function for all sets: train, test, validation, and store it. We print the accuracy for each dataset for each lambda value and pick the best lambda value and use it to train the logistic regression model

lr(data, lambd): Primary logistic regression function which trains on the entire training set and gives output on the test set as well. We use the best lambda value from the above function

get_evaluation_metrics(Y, Y_pred): Returns accuracy, precision, recall, f1

In the SGD-CLASSIFIER section, we apply SGD to each dataset. Firstly we use grid search to find the optimal hyperparameters and use those hyperparameters to train and test the SGD CLassifier

Performance Metrics:

1. Multinomial Naive Bayes on the Bag of words model

Datasets	Train/Test	Accuracy	Precision	Recall	F1
hw1	train	0.984	0.953	0.991	0.972
	test	0.943	0.850	0.961	0.902
enron1	train	0.993	0.977	1.0	0.988
	test	0.938	0.890	0.926	0.90
enron4	train	0.779	1.0	0.706	0.827
	test	0.858	1.0	0.803	0.890

2. Discrete Naive Bayes on the Bernoulli model

Datasets	Train/Test	Accuracy	Precision	Recall	F1
hw1	train	0.842	0.946	0.430	0.592
	test	0.771	0.888	0.184	0.305
enron1	train	0.831	1.0	0.419	0.591
	test	0.730	0.906	0.194	0.320
enron4	train	0.921	0.934	0.962	0.948
	test	0.913	0.892	1.0	0.943

3. Logistic Regression
    a. Bag of Words model

Model: Bag Of Words model ->
Set: train, Dataset: hw1 -> Accuracy: 0.9762419006479481, Precision: 0.9590163934426229, Recall: 0.9512195121951219, F1-Score: 0.9551020408163264
Set: train, Dataset: enron1 -> Accuracy: 0.9755555555555555, Precision: 0.9918032786885246, Recall: 0.9236641221374046, F1-Score: 0.9565217391304348
Set: train, Dataset: enron4 -> Accuracy: 0.9345794392523364, Precision: 0.9717223650385605, Recall: 0.9402985074626866, F1-Score: 0.9557522123893806
Set: test, Dataset: hw1 -> Accuracy: 0.6443514644351465, Precision: 0.2619047619047619, Recall: 0.16923076923076924, F1-Score: 0.205607476635514
Set: test, Dataset: enron1 -> Accuracy: 0.5307017543859649, Precision: 0.23140495867768596, Recall: 0.18791946308724833, F1-Score: 0.2074074074074074

Set: test, Dataset: enron4 -> Accuracy: 0.7716390423572744, Precision: 0.801354401805869, Recall: 0.907928388746803, F1-Score: 0.8513189448441246

b. Bernoulli

Model: Bernoulli model ->
Set: train, Dataset: hw1 -> Accuracy: 0.9654427645788337, Precision: 0.9908256880733946, Recall: 0.8780487804878049, F1-Score: 0.9310344827586207
Set: train, Dataset: enron1 -> Accuracy: 0.9666666666666667, Precision: 0.9754098360655737, Recall: 0.9083969465648855, F1-Score: 0.9407114624505928
Set: train, Dataset: enron4 -> Accuracy: 0.9663551401869159, Precision: 0.9571428571428572, Recall: 1.0, F1-Score: 0.9781021897810218
Set: test, Dataset: hw1 -> Accuracy: 0.6903765690376569, Precision: 0.29545454545454547, Recall: 0.1, F1-Score: 0.14942528735632182
Set: test, Dataset: enron1 -> Accuracy: 0.5087719298245614, Precision: 0.10526315789473684, Recall: 0.06711409395973154, F1-Score: 0.08196721311475409
Set: test, Dataset: enron4 -> Accuracy: 0.6813996316758748, Precision: 0.7339055793991416, Recall: 0.8746803069053708, F1-Score: 0.7981330221703618

4. SGDClassifier
    a. Bag of Words model

Model: Bag Of Words model ->
Dataset: hw1, and Model: Bag Of Words model -> Accuracy: 0.68
Dataset: enron1, and Model: Bag Of Words model -> Accuracy: 0.59
Dataset: enron4, and Model: Bag Of Words model -> Accuracy: 0.74
Set: train, Dataset: hw1 -> Accuracy: 0.9028077753779697, Precision: 0.9642857142857143, Recall: 0.6585365853658537, F1-Score: 0.782608695652174
Set: train, Dataset: enron1 -> Accuracy: 0.9577777777777777, Precision: 0.9912280701754386, Recall: 0.8625954198473282, F1-Score: 0.9224489795918368
Set: train, Dataset: enron4 -> Accuracy: 0.9327102803738317, Precision: 0.937799043062201, Recall: 0.9751243781094527, F1-Score: 0.9560975609756097
Set: test, Dataset: hw1 -> Accuracy: 0.6841004184100419, Precision: 0.24390243902439024, Recall: 0.07692307692307693, F1-Score: 0.11695906432748539
Set: test, Dataset: enron1 -> Accuracy: 0.5855263157894737, Precision: 0.11538461538461539, Recall: 0.040268456375838924, F1-Score: 0.059701492537313425
Set: test, Dataset: enron4 -> Accuracy: 0.7403314917127072, Precision: 0.7615062761506276, Recall: 0.9309462915601023, F1-Score: 0.8377445339470656

b. Bernoulli

Model: Bernoulli model ->
Dataset: hw1, and Model: Bernoulli model -> Accuracy: 0.72
Dataset: enron1, and Model: Bernoulli model -> Accuracy: 0.62
Dataset: enron4, and Model: Bernoulli model -> Accuracy: 0.69

Set: train, Dataset: hw1 -> Accuracy: 0.9244060475161987, Precision: 1.0, Recall: 0.7154471544715447, F1-Score: 0.8341232227488152
Set: train, Dataset: enron1 -> Accuracy: 0.94, Precision: 0.9814814814814815, Recall: 0.8091603053435115, F1-Score: 0.8870292887029289
Set: train, Dataset: enron4 -> Accuracy: 0.9271028037383178, Precision: 0.9134396355353075, Recall: 0.9975124378109452, F1-Score: 0.9536266349583828
Set: test, Dataset: hw1 -> Accuracy: 0.7154811715481172, Precision: 0.0, Recall: 0.0, F1-Score: nan
Set: test, Dataset: enron1 -> Accuracy: 0.6228070175438597, Precision: 0.12903225806451613, Recall: 0.026845637583892617, F1-Score: 0.044444444444444446
Set: test, Dataset: enron4 -> Accuracy: 0.6887661141804788, Precision: 0.7126436781609196, Recall: 0.9514066496163683, F1-Score: 0.8148959474260679

Tuning of Hyper-parameters:
The hyper-parameters were chosen after trying a variety of them and choosing the best ones: Learning rate, lambda, number of iterations. A plot of accuracy vs. no of iterations helped in deciding where to stop iterating over the data. Lambda was chosen by checking different values on the validation set. The learning rate was chosen after trying out various values of LR like 0.1, 0.01, 0.001, etc. A hard limit was placed on the number of iterations because if we allow it to converge then the model highly overfits the training set.

• Answers to the questions:

1. Multinomial Naive Bayes on the Bag of words model gives the best performance in all the models. The reason is that NB is a generative model while LR and SGD are discriminative models. Here the dataset we have is very small and there is bias in the dataset as well, which is where generative models perform better, thus NB is better. Also, Multinomial NB performs better than Bernoulli NB because, Bernoulli only stores 1 or 0 based on the occurrence of words in the documents, where as the bag of words model stores the frequency of the words as well in the documents, which is better.

2. Yes. Multinomial NB performs much better than LR and SGD. The reason is that NB is a generative model while LR and SGD are discriminative models. Here the dataset we have is very small and there is bias in the dataset as well, which is where generative models perform better, thus Multinomial NB is better.

3. Yes. Discrete NB performs better than LR and SGD. The reason is that NB is a generative model while LR and SGD are discriminative models. Here the dataset we have is very small and there is bias in the dataset as well, which is where generative models perform better, thus Discrete NB is better.

4. No. LR doesn’t outperform SGD. SGD performs slightly better