CSD316 - Assignment 2

Pranav Dahiya

1710110249

Question 1

data_generation.py generates 1000 data points randomly following the specifications given in the question with $a = 1$ and m is 5, 10 and 20. This was done 3 times in order to validate results. The data can be found in folders data_1 to data_3. The results for each subpart are given in this file whereas the weights for each dataset are in csv files in their respective folders.

The results for the batch and increment algorithm are in tables 1 and 2. The constant learning rate used was $\rho = 3 * 10^{-6}$ whereas the variable learning rate was $\rho(i) = \frac{1.25}{1000+i}$.

Table 1: Batch and Increment with Constant Learning Rate

Dataset	No of separating dimensions	Iterations
1	5	99
	10	91
	20	102
2	5	170
	10	72
	20	56
3	5	142
	10	114
	20	40

Table 2: Batch and Increment with Variable Learning Rate

Dataset	No of separating dimensions	Iterations
1	5	121
	10	191
	20	41
2	5	99
	10	94
	20	32
3	5	272
	10	70
	20	59

The datasets were made nonseparable by generating 50 more points for each dataset according to the given specifications. The python script for the same is nonseparable_data_generation.py. The pocket algorithm is implemented in pocket.py. The results for the pocket algorithm are given below in table 3. Each perceptron is trained for 500 iterations.

Table 3: Pocket Algorithm

Dataset	No of separating dimensions	No of misclassifications
1	5	86
	10	92
	20	103
2	5	88
	10	102
	20	98
3	5	83
	10	82
	20	99

The least mean square algorithm script is in least_mean_square.py. The results are given below in table 4.

Table 4: Least Mean Square Algorithm

Dataset	No of separating dimensions	No of misclassifications
1	5	109
	10	157
	20	132
2	5	86
	10	123
	20	122
3	5	106
	10	117
	20	127

Figure 1 shows the number of iterations taken to reach convergence for the batch and increment algorithm on the separable dataset as the number of separating dimensions increases. The general trend is that the iterations to convergence decrease as the number of separating dimensions increases.

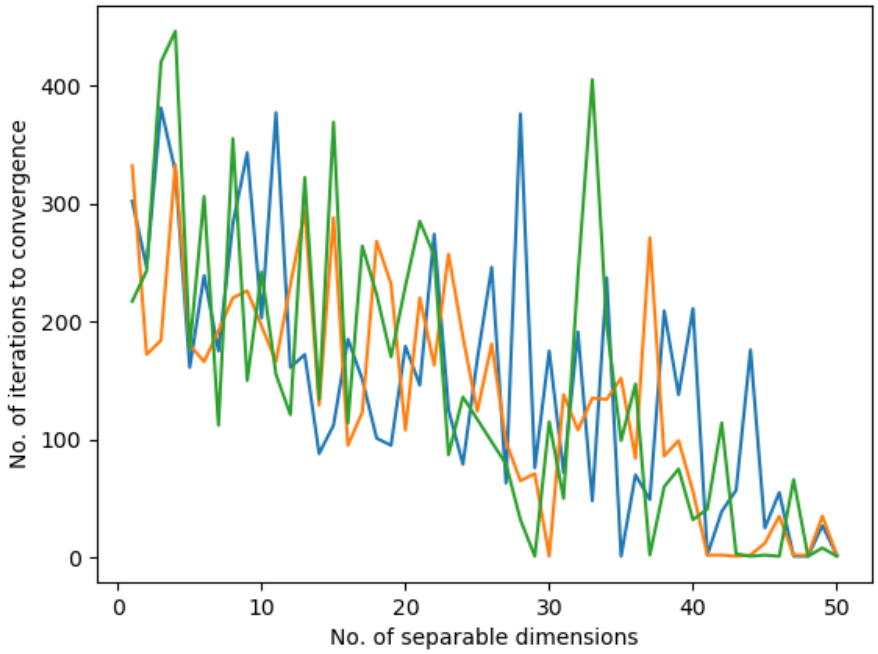

Figure 1: No. of iterations vs No. of separating dimensions

Question 2

First the entire dataset was was split into training and testing sets using train_test_split.py such that 90% of the entire dataset was randomly chosen for the training set and the rest was the testing set. The vocabulary after stop word removal, lemmatization and thresholding was generated using vocabulary.py. The thresholds of the vocabulary were tuned again for training the preceptron. The lower bound was set to 22 occurrences in the dataset and no upper bound was set to 1290. When choosing an appropriate threshold, a higher accuracy on both the train and test sets was preferred. Then 6 linear classifiers were trained using the least mean square algorithm (spam_filter_weights_1.pickle - spam_filter_weights_12.pickle). The python code for the same can be found in spam_filter.py. The results are shown in table . A comparison of the mean performance of the linear classifier vs the naive Bayes classifier are shown in table .

Table 5: Performance of the linear classifier on lingspam

	TP	**FP**	**TN**	**FN**
1	44	8	232	6
2	42	7	233	8
3	43	4	236	7
4	46	12	228	4
5	45	10	230	5
6	46	12	228	4
average=	13.58	25.25	214.75	36.42
stadard deviation=	5.63	12.75	12.75	5.63

Table 6: Comparison of Performance of LMS and Naive Bayes on the lingspam dataset

Performance Metric	**LMS**		**Naive Bayes**	
	Mean	σ	Mean	σ
TPR(Recall)	0.89	0.03	0.45	0.10
FPR	0.04	0.01	0.00	0.00
TNR	0.96	0.01	1.00	0.00
Precision	0.84	0.05	0.98	0.03
Accuracy	0.95	0.01	0.91	0.02
F1	0.86	0.01	0.61	0.09

The performance of the linear classifier is better than the naive Bayes classifier with regards to TPR, Accuracy and F1 score. However, naive Bayes was better in terms of FPR, TNR and Precision.