CS5011 Machine Learning
Report on Programming Assignment 1

Abhishek A. Narwekar
EE11B129

September 2015

1 Linear Classifier

1.1 Generation of the Synthetic Data Set, DS1

Our objective was to create a 2-class data set using a choice of centroids such that there was some overlap between the data points of the 2 classes. In order to do so, I used the *mvnrnd*() function in MATLAB to generate 1000 data points for each class. Its inputs, viz. the mean vector and the covariance matrix, are generated as follows. Since the requirement was that the data points in the two classes must overlap, I chose the mean vectors of the two classes as $\mu_1 = [0,0,0,0,0,0,0,0,0,0]^T$ and $\mu_2 = [1,1,1,1,1,1,1,1,1,1]^T$. The smaller the separation between the two classes, worse the performance of the classifier. The covariance matrix, which was to be non-spherical, is chosen as follows for both classes:

$$\Sigma_1 = \Sigma_2 = \begin{bmatrix} 1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 1 \end{bmatrix} \tag{1}$$

The above choice ensured that there existed a weak correspondence between all pairs of features ($\Sigma_{ij} = 0.1$ for $i \neq j$), and that there was overlap between datasets of two classes ($\Sigma_{ii} = 1$, which happened to be the difference in the mean of each feature). The latter can be inferred from visualization for a single variable using the 68–95–99.7 rule. In this case, we can easily see that roughly 16% of the points in class 1 overlap with class 2, and vice versa.

1.2 Evaluation of Linear Regression on DS1

I made random splits of 60%-40% for training and testing respectively on the dataset, learned the coefficients of linear regression on the training samples and evaluated their performance on the test samples. To account for the randomness, I performed 10000 such random splits and observed the statistics of the misclassification over this ensemble of splits. It was observed that on an average, of 800 test points, 114.30 got misclassified. The standard deviation in the number of misclassifications was 7.70. The best and the worst classifiers misclassified 83 and 153 points respectively. For the best classifier, the weights learned were:

$$\beta = \begin{bmatrix} 0.050 & 0.057 & 0.083 & 0.065 & 0.082 & 0.067 & 0.081 & 0.089 & 0.083 & 0.081 \end{bmatrix}^T \tag{2}$$

1.3 On the Performance of the k-NN Classifier

I varied the number of nearest neighbours, k, from 1 to 50. To evaluate the k-NN classifier, I computed the number of misclassifications out of the 800 test samples for each value of k. To account for the randomness in chossing the folds, I performed this experiments over 50 instances, and have reported the average error over these instances in Table [1]. For the same set of instances, I also evaluated the performance of the linear regression model in terms of the number of misclassifications. The result is shown in Figure [1]. As we can see, linear regression performs better than k-NN till about $k = 7$, after which k-NN outperforms linear regression. For larger values of k, the performance of k improves. It eventually stabilizes for $k > 15$.

2 Linear Regression

2.1 Pre-processing the Communities and Crime Dataset

Firstly, the columns containing the features were extracted. According to the documentation, the features were in column 6 to column 127. Column 128 contained the labels. To handle the entries with NaN, I implemented two methods:

1. **Method 1 (M1): Replacement with global average**
   In this method, I replaced NaN observations with the average value of the non-NaN observations for that feature.

2. **Method 2 (M2): Replacement with "classwise" average**
   The class labels are continuous quantities normalized to lie between 0 and 1. In fact, the labels are presented to an accuracy of 2 decimals. In this method, I first assumed an inductive bias: two elements are similar if their labels are similar. This similarity was strongly imposed by binning the data into a given number of bins and clustering observations based on the bins to which they belonged. The experiments were done for 10

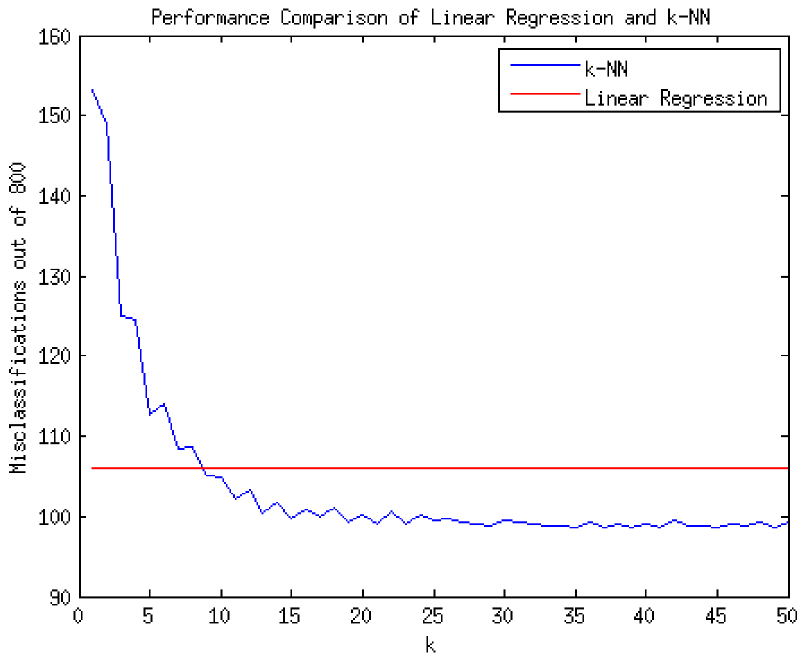

Figure 1: Variation of k-NN performance with k averaged over 50 instances and its comparison with Linear Regression

bins. Thus, observations with labels lying between 0-0.10 were binned and treated as class 1, those between 0.10-0.20 as class 2, and so on. I replaced the NaN observations in this case by the average of the non-NaN observations belonging to that class. Note that I tried using the *natural binning* obtained by truncation of labels to two decimals. However, in many cases, there just weren't any non-NaN observations in that class to replace the NaN elements with. Thus, I switched to a smaller number of bins.

2.2 Performance of Linear Regression

I performed linear regression and found the residual sum of squares upon regressing the entire data with itself. Upon creating the data with M2, linear regression gave a RSS of 32.5321. But using M2, this value dropped to 13.4238, indicating that the fit is better with M2. Since the dimension of the coefficient vector β is large, it has been stored in the following csv files: $Q2\_BETA\_LR\_M1.csv$(for M1) and $Q2\_BETA\_LR\_M2.csv$(for M2).

k	Misclassifications	k	Misclassifications
1	153.28	26	99.8
2	148.82	27	99.2
3	125.06	28	98.96
4	124.5	29	98.76
5	112.66	30	99.4
6	114.08	31	99.28
7	108.42	32	99.14
8	108.58	33	98.88
9	105.04	34	98.94
10	104.78	35	98.66
11	102.22	36	99.24
12	103.28	37	98.6
13	100.42	38	98.98
14	101.72	39	98.62
15	99.66	40	99
16	100.78	41	98.58
17	99.86	42	99.4
18	101.08	43	98.88
19	99.18	44	98.94
20	100.1	45	98.68
21	99.14	46	99.1
22	100.54	47	98.82
23	99.12	48	99.28
24	100.16	49	98.56
25	99.44	50	99.26

Table 1: Average misclassification over 50 iterations for various values of k

2.3 Performance of Ridge Regression

Firstly, I divided the data into 5 folds using a random permutation. I did not normalize it at this point. I used the *ridge*() function in MATLAB with the parameter $scaled = 0$. This indicated that the data was not normalized, and the parameters (β) estimated were to be restored to the scale of the original data. λ, which is denoted as the parameter k in MATLAB, was varied to capture the minimum in the RSS. The values of taken by λ and the corresponding RSS have been shown in Tables [2](for M1) and [3](for M2). Note that in these two tables, the first two columns correspond to coarse variations in λ which helped me zero in on the rough value of the optimum λ, while the last two correspond to fine variations. Note that these values were computed for a random permutation of the training data which gave me 5 folds. **The permutation that resulted in these folds was different for both M1 and M2.** The optimal value varies with each permutation, and thus we cannot arrive at an exact $\lambda_{optimal}$. However, one thing that we can conclude is that the RSS is significantly lower for M2 than for M1, as seen in figure [2].

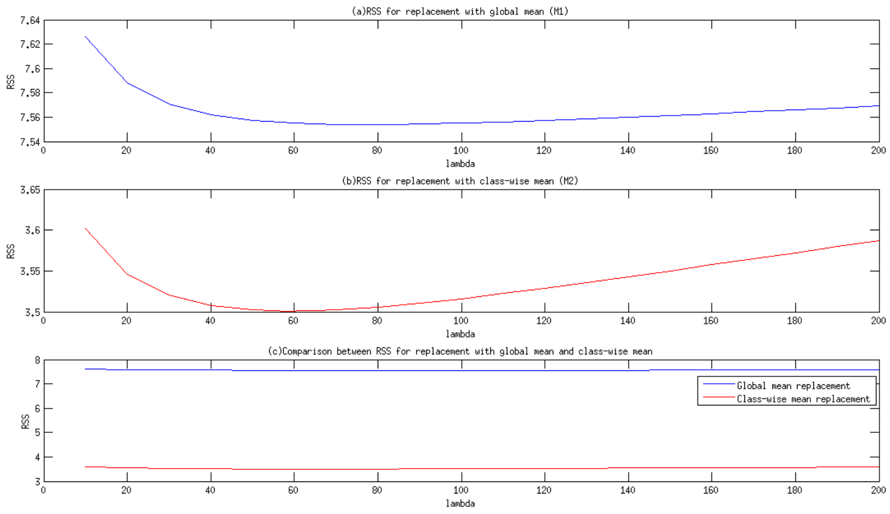

Figure 2: Variation of RSS with λ for (a)M1, (b)M2 and (c)a comparison of both

3 Feature Extraction

3.1 PCA

The raw data has been plotted in figure [3].

I used the function $pca()$ in MATLAB for performing PCA. We get three output matrices: COEFF, SCORE and LATENT. Out of these, the matrix SCORE gives us the representation of the data in the principal component space. Thus, to reduce the dimensionality of each observation in DS3 to 1, I simply chose the first column of SCORE. Now, we need to classify the data using linear regression on the single variable. The regression follows the following equation:

$$\beta_0 + \beta_1 X_{reduced} = Y \tag{3}$$

Using this, we can evaluate $\beta_0 = \frac{\mathbb{1}Y^T}{N} = 1.5$. From MATLAB, we get:

$$\beta_1 = 0.0852$$

The decision boundary is given by the following equation:

$$\beta_0 + \beta_1 X_{reduced} \quad \overset{1}{\underset{2}{\gtrless}} \quad 1.5 \tag{4}$$

λ(coarse)	RSS(λ)	λ(fine)	RSS(λ)
0	145.95	75	7.553745
10	7.6263	76	7.553735
20	7.588	77	7.553731
30	7.5707	78	7.553734
40	7.5619	79	7.553742
50	7.5572	80	7.553756
60	7.5549	81	7.553775
70	7.5539	82	7.553799
80	7.5538	83	7.553828
90	7.5542	84	7.553862
100	7.5549	85	7.553900
110	7.5559	85	7.553900
120	7.5571		
130	7.5584		
140	7.5598		
150	7.5613		
160	7.5628		
170	7.5644		
180	7.566		
190	7.5676		
200	7.5692		

Table 2: RSS for different values of λ in M1

Thus, the value of the decision boundary is:

$$X_{decision} = \frac{1.5 - \beta_0}{\beta_1} = 0 \tag{5}$$

A plot of the reduced data and the decision boundary is shown in figure [4]

The performance of classification after dimensionality reduction in PCA is given in table [4]

3.2 LDA

An LDA model was fit using the $fitcdiscr()$ function in MATLAB, followed by the *predict*() function for testing. There were 0 misclassifications, as seen in table [5]. The data points upon classification can be visualized in figure [5].

3.3 Comparison of the performance of PCA+Regression and LDA

In PCA, the data is projected along the vector with the highest variance along it in the training data. In this case, that vector isn't the class boundary, and

λ(coarse)	RSS(λ)	λ(fine)	RSS(λ)
0	4.7535	55	3.5012
10	3.6022	56	3.5011
20	3.5461	57	3.501
30	3.5205	58	3.501
40	3.5079	59	3.501
50	3.5023	60	3.501
60	3.501	61	3.5011
70	3.5025	62	3.5011
80	3.5058	63	3.5012
90	3.5105	64	3.5013
100	3.516	65	3.5015
110	3.5222		
120	3.5288		
130	3.5357		
140	3.5428		
150	3.5501		
160	3.5574		
170	3.5648		
180	3.5723		
190	3.5797		
200	3.587		

Table 3: RSS for different values of λ in M2

	Class 1	Class 2
Precision	0.5491	0.7162
Recall	0.8950	0.2650
F1	0.6806	0.3869

Table 4: Performance of classification with PCA-based dimensionality reduction followed with linear regression

	Class 1	Class 2
Precision	1	1
Recall	1	1
F1	1	1

Table 5: Performance of classification with LDA

hence, classifying using it doesn't result in the optimal performance. LDA, however, outperforms PCA+Regression in this case because the data is linearly separable. It thus successfully identifies the boundary which separates the two

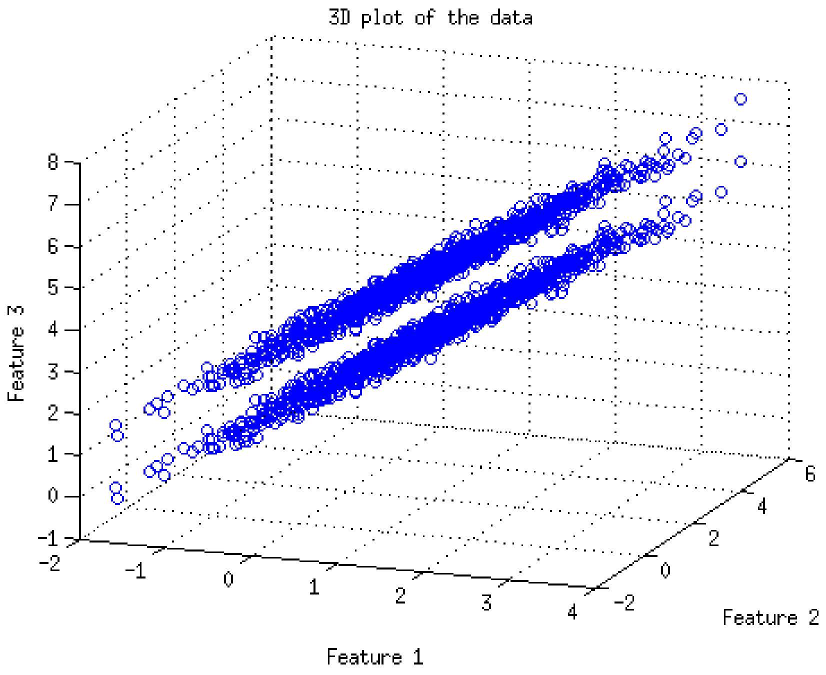

Figure 3: 3D Visualization of DS3

classes.

4 Support Vector Machines

In this question, I constructed colour histograms for each image. To evaluate the performance of the SVM parameters C and γ, I segregated the data into 5 folds. LibSVM was used to train models using the parameters.

4.1 Searching for Good Parameters

Since we need to determine the best possible values for both C and γ simultaneously (except in the linear kernel case), the search space is large. So I resorted to a *logarithmic* grid search. I first tried to estimate roughly which areas of the search space gave high accuracy. C and γ were both varied from 10^{-3} to 10^{3} logarithmically. Upon locating the combination of C and γ which gave good results, I searched in the vicinity of that point *linearly*. The search was performed in all cases with a precision of 0.001 in both C and γ.

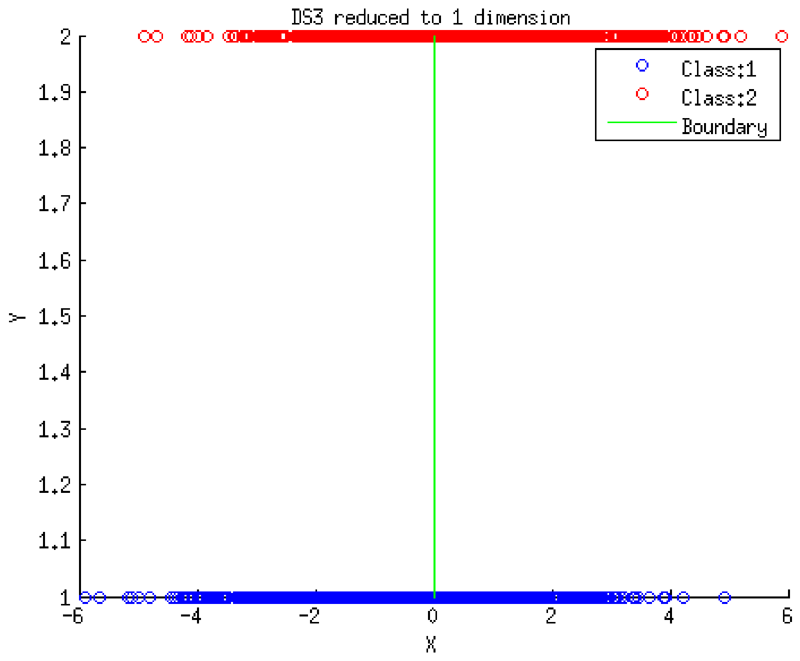

Figure 4: 3D Visualization of DS3

4.2 The optimal models and their performances

The optimal values of parameters, c and γ for various kernels are shown in Table [6]. For each kernel, the optimal model struct is stored in the tarball.

Kernel	Best C	Best γ	Average Accuracy
Linear	0.023	-	58.01
Polynomial	0.06	0.049	60.398
	0.031	0.061	60.398
Gaussian	1.55	0.015	66.4677
Sigmoid	0.0625	0.0156	51.2438

Table 6: Best parameter sets for SVM trained using various kernels

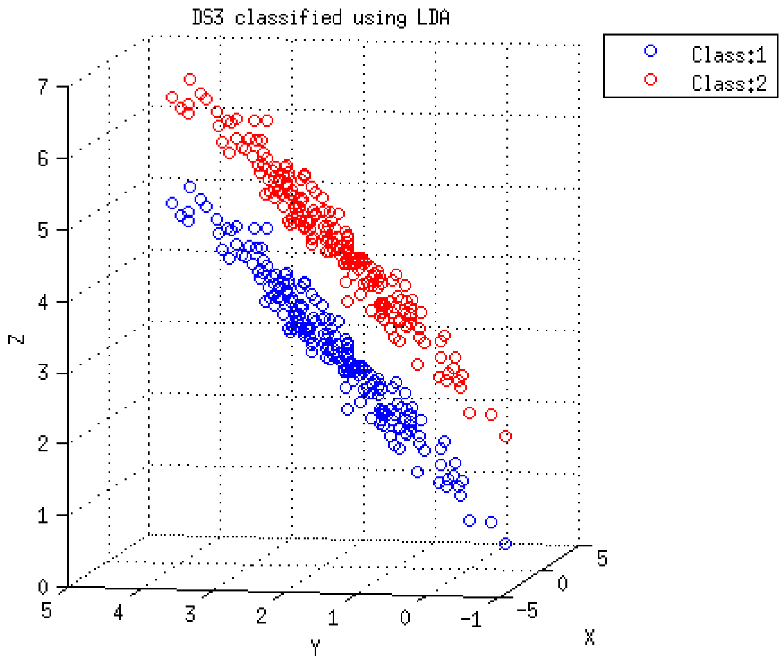

Figure 5: 3D Visualization of Classified Data Points using LDA

5 Bayesian Parameter Estimation

5.1 ML Estimation using Multinomial Likelihood

The likelihoods can be calculated as:

$$p(w|i) = \frac{n_{w,i}}{\sum\limits_{w \in V} n_{w,i}}, \quad i \in \{spam, ham\} \tag{6}$$

where $n_{w,i}$ is the number of *documents* of class i in which the word w occurs. Using these results, the likelihood of a class given a test mail can be obtained as:

$$l(i|d) = log(L(i|d)) = log(p(d|i)) = \sum_{w \in V} n_{w,d} log(p(w|i)) \tag{7}$$

Here, d is the test document and $n_{w,d}$ is the number of occurrences of the word w in d. The log helps prevent underflow when the probabilities are very small. The class with the larger likelihood is chosen. The average performance over 5-fold cross validation of this method is shown in table [7] The PR curve corresponding to this algorithm is shown in figure [6] for the spam class. As we can see, it is very near to the ideal PR curve, indicating a good performance.

	Class 1	Class 2
Precision	0.9768	0.9377
Recall	0.9498	0.9709
F1	0.9631	0.9540

Table 7: Performance of Naive Bayes Classifier with multinomial likelihood

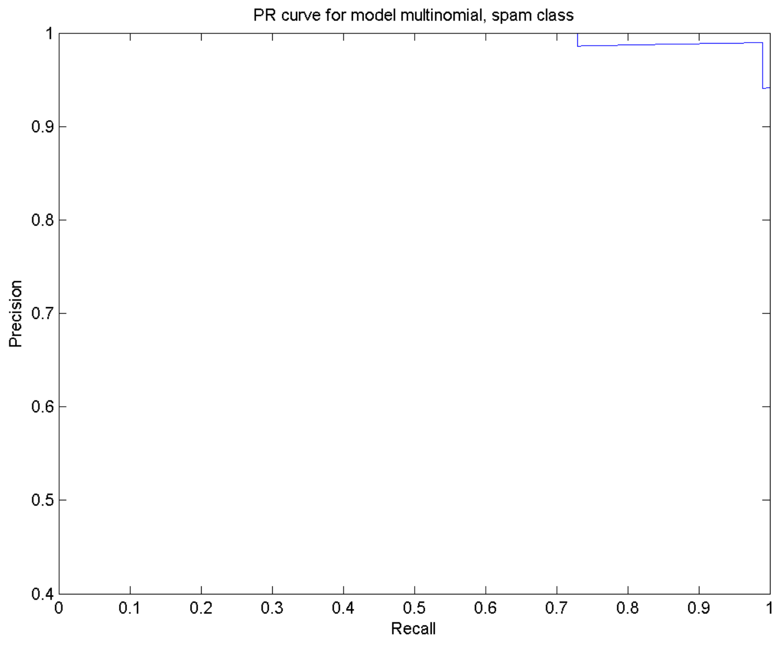

Figure 6: PR curve for spam class using multinomial likelihood

5.2 ML Estimation using Binomial Likelihood

The only difference in this case is in the computation of the likelihoods, which are given by:

$$p(w|i) = \frac{n'_{w,i}}{\sum\limits_{w \in V} n'_{w,i}}, \quad i \in \{spam, ham\} \tag{8}$$

where $n'_{w,i}$ is the number of *documents* of class i in which the word w occurs. The performance in this case is shown in table [8].

The PR curve corresponding to this algorithm is shown in figure [7] for the spam class. Here too, it is very near to the ideal PR curve, indicating a good performance. In fact, this algorithm outperforms the one in the multinomial likehood case, based on the F1 scores.

	Class 1	Class 2
Precision	0.9902	0.9447
Recall	0.9547	0.9875
F1	0.9720	0.9654

Table 8: Performance of Naive Bayes Classifier with binomial likelihood

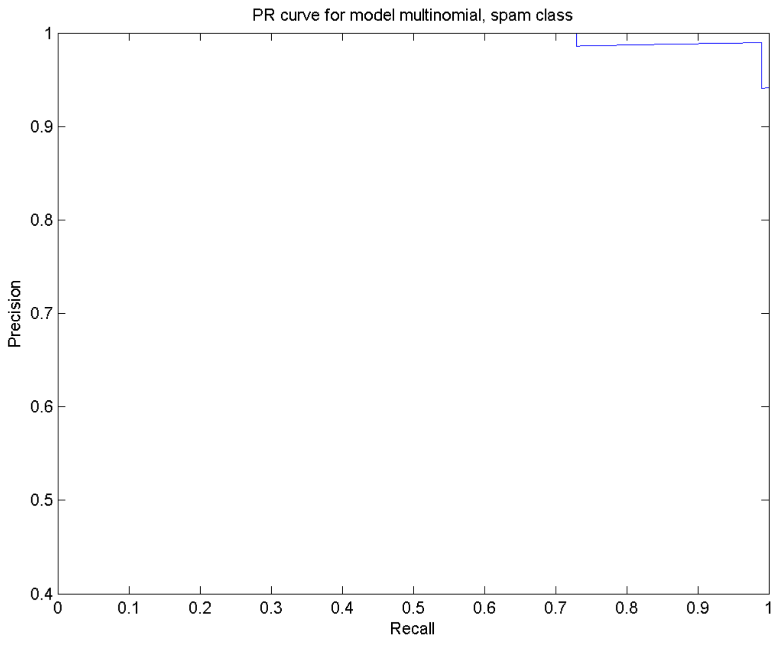

Figure 7: PR curve for spam class using binomial likelihood

5.3 ML Estimation with Dirichlet Priors

The priors for all words are given by $p \sim Dir(\vec{\alpha})$. To estimate the likelihoods, we use the fact that the likelihood terms are the conjugate priors of the Dirichlet distribution, which also follow the Dirichlet distribution. They obey the following relationship.

$$p(w|i) = E[Dir(n_{w,i} + \alpha_w)] = \frac{n_{w,i} + \alpha_w}{\sum\limits_{x \in V}(n_{x,i} + \alpha_x)}, \quad i \in \{spam, ham\} \tag{9}$$

Here, $E[\,]$ is the expectation operator. After this, we evaluate the probabilities the same way that we did in equation [7]. To inspect the variation in performance, I kept the α uniform, but increased the magnitude of each element. The results are shown in table [9]. As we can see, the F1 score of both the ham and the spam classes *decreases* as the magnitude of α increases from 1 to 5 to 50. This can be explained by the following reasoning. As seen in equation [9], a way to look at the likelihood terms is that they experience a smoothing by their corresponding terms in α. We know that smoothing (a method of dealing with the likelihood terms which are zero) worsens the performance when done in great amount.

		$\alpha_i = 1 \forall i$	$\alpha_i = 5 \forall i$	$\alpha_i = 50 \forall i$
	Precision	0.9768	0.9701	0.9437
Ham	Recall	0.9498	0.9240	0.8625
	F1	0.9631	0.9462	0.9010
	Precision	0.9377	0.9082	0.8413
Spam	Recall	0.9709	0.9625	0.9335
	F1	0.9540	0.9342	0.8847

Table 9: Performance of Naive Bayes Classifier with Dirichlet priors

The PR curves for two values of α are shown in figure [8]. As seen, the area under the curve decreases when $\alpha_i = 50 \forall i$, indicating a worsening in the performance.

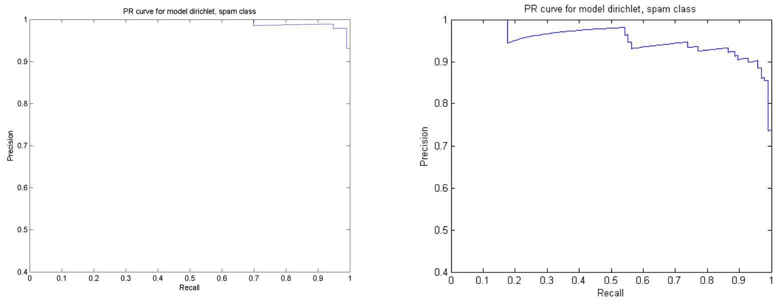

Figure 8: PR curve for (a)$\alpha_i = 1 \forall i$ and (b) $\alpha_i = 50 \forall i$

5.4 ML Estimation with Beta Priors

The priors for all words are given by $p \sim Beta(\alpha, \beta)$. Here too, to estimate the likelihoods, we use the fact that the likelihood terms are the conjugate priors of the Beta distribution, which also follow the Beta distribution. In fact, the Dirichlet distribution is a generalization of the Beta distribution. Thus, the likelihoods are given by:

$$p(w|i) = E[Beta(n_{w,i}+\alpha, \sum_{x \in V} n_{x,i} - n_{w,i} + \beta)] = \frac{n_{w,i} + \alpha}{\sum_{x \in V} (n_{x,i} + \alpha + \beta)}, \quad i \in \{spam, ham\} \tag{10}$$

Recall that the expectation of the Beta distribution is given by:

$$E[Beta(\alpha, \beta)] = \frac{\alpha}{\alpha + \beta} \tag{11}$$

This substitution has been made in equation [10]. Again, we evaluate the probabilities of the class the same way that we did in equation [7]. This time, I observed the variation in the performance as α β both vary from small to large values. Specifically, I observed the variation in performance for the following $\{\alpha, \beta\}$ pairs: $\{1, 1\}, \{1, 100\}, \{100, 1\}, \{100, 100\}$. The results are shown in table [10]. Here too, α is responsible for smoothing. As we can see, performance degrades more drastically when α increases as compared to when β increases. This is because the initial estimate of the prior distribution has a mean given by equation [11]. Since we can expect the number of occurrences of each word to be very small as compared to the total occurrences of all words, having $\beta \gg \alpha$ gives a more realistic estimate than when $\beta \approx \alpha$ or $\beta \ll \alpha$. Thus, the performance of the classifier is poorer in the latter two cases.

		$\alpha = 1, \beta = 1$	$\alpha = 1, \beta = 100$	$\alpha = 100, \beta = 1$	$\alpha = 100, \beta = 100$
Ham	Precision	0.9768	0.9798	0.9055	0.9268
	Recall	0.9498	0.9369	0.8867	0.8641
	F1	0.9631	0.9578	0.8957	0.8938
Spam	Precision	0.9377	0.9235	0.8579	0.8395
	Recall	0.9709	0.9750	0.8794	0.9106
	F1	0.9540	0.9485	0.8680	0.8730

Table 10: Performance of Naive Bayes Classifier with Beta priors

A plot of the PR curves for two combinations of $\{\alpha, \beta\}$ is shown in figure [9]. As expected, the area under the PR curve decreases when α and β increase.

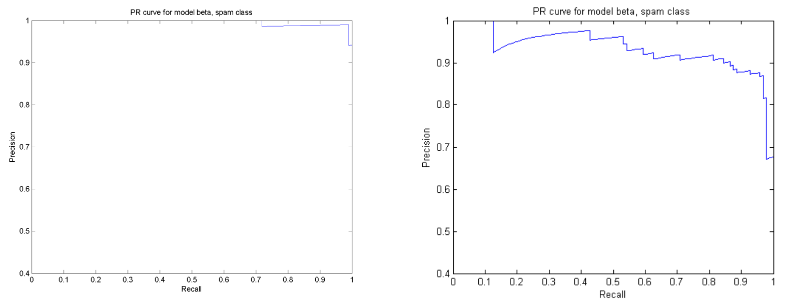

Figure 9: PR curve for (a)$\alpha = 1, \beta = 1$ and (b) $\alpha = 100, \beta = 100$