# Principles of Machine Learning (CS4011)

Programming Assignment 3

Ameet Deshpande - (CS15B001)

November 10, 2017

# 1 Question 1 - Clustering

## 1.1 Visualizing the Plots

### 1.1.1 Aggregation

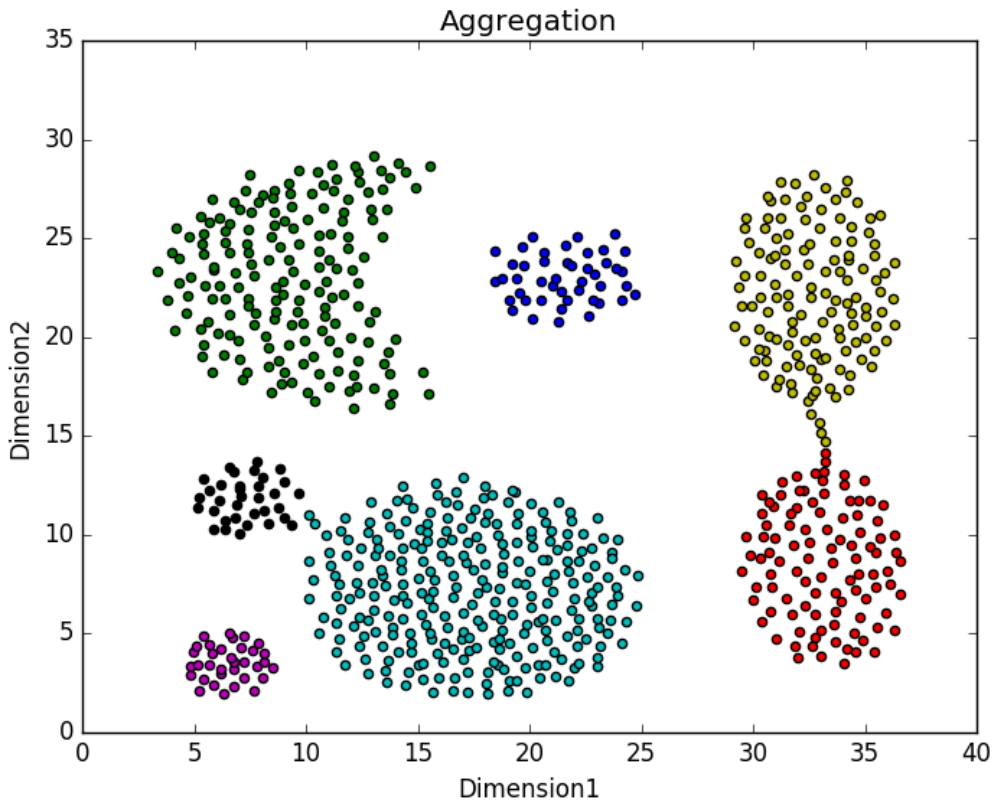


Figure 1: Aggregation Dataset

- **K-Means** algorithm will mostly do well on this dataset. We can see that all the clusters are approximately convex in shape. This means that the centroid

will actually lie inside the cluster for all of them. The K-Means algorithm can be run several times and it should be seen that at $K = 7$, there will be an "elbow" or a "knee". A nice way to look at if $K - Means$ can give the right answer is to check if the final visual answer is a solution that can be achieved from the K-Means initialization. If we initialize the centroids of all the clusters correctly, we can see that the points belonging to the cluster will be closest to that cluster itself. Thus, K-Means algorithm can converge to these clusters as the right solution. However note that since the cluster sizes are very different, there will be some points from the teal cluster which will get assigned to the purple and black cluster.

- **DBSCAN** algorithm will work well on this dataset for the right values of *eps* and $minPts$. The Red cluster and the yellow cluster have some points close to each other. The parameter minPts should thus be chosen to be slightly large, and the *eps* value small. This will ensure that the linkages between the clusters do not cause the two clusters to coalesce. The same thing goes for the teal and the black cluster. This algorithm however does run into problems with black-teal and yellow-red clusters. This is because there is a small region which is densely connecting the two clusters.

- **Hierarchical Clustering with Single Link** Hierarchical Clustering uses a distance metric to calculate clusters. As single link chooses the closest points and then clusters, it can be seen that the yellow and red cluster will get combined first as they have points which are very close to each other. In fact, the red and yellow point seem to be one of the closest points in the diagram. There will thus not be any way to differentiate between the two clusters. Other clusters will however be clustered among themselves first and then with other clusters. The black and the teal cluster are also pretty close and they will be the second pair of clusters to be coalesced. This method will therefore not do a good job in separating the yellow-red clusters and the black-teal clusters.

- **Hierarchical Clustering with Complete Link** Complete link uses the farthest points in two clusters to join two clusters. This will mostly work very well in this case. This is because it will force all the true clusters to get clustered first. The yellow and the red cluster will most probably not be in the same cluster. Complete link will however favor smaller clusters to get clustered first. This is because larger clusters will inherently have larger distances for complete link and smaller distances for single link. It will thus do well on this dataset as there are no very small clusters as such. Indeed, this performs the best out of all four methods.

### 1.1.2 Compound

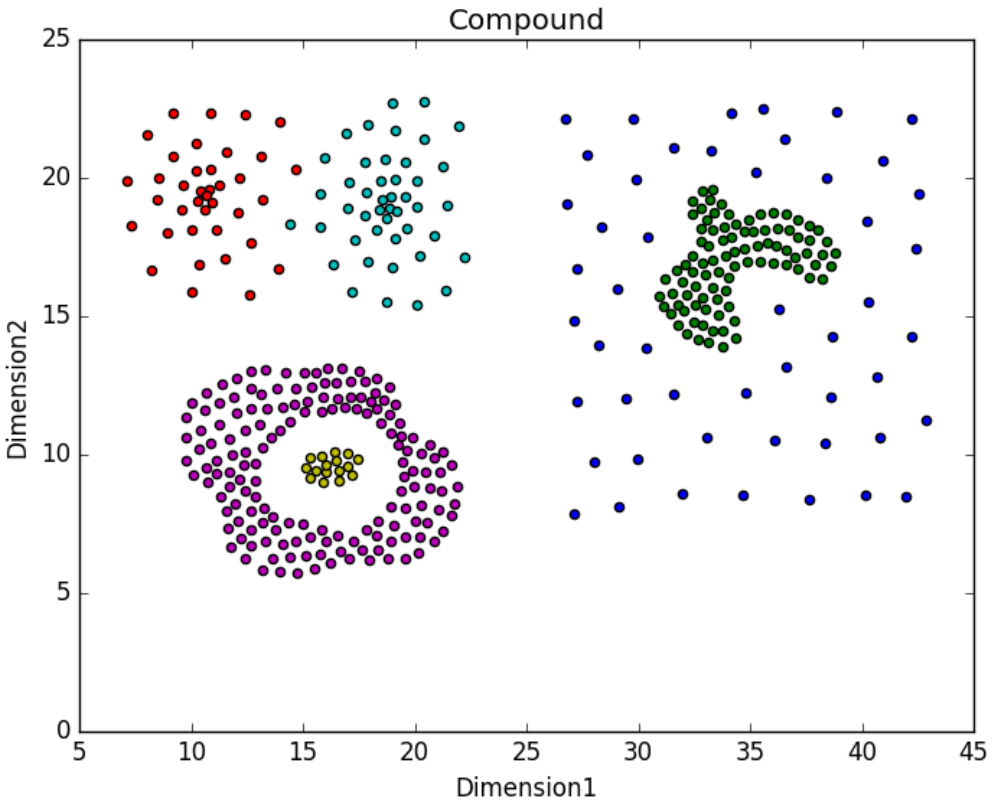


Figure 2: Compound Dataset

- **K-Means** algorithm will not work in this case. It will do a good job in clustering the red and the teal cluster as their boundary is convex in shape and the density is continously spread inside the boundary. The dense core that they possess is where the expected centroids will be. There will however be problems in clustering the purple and yellow clusters. This is because, the centroids of the true purple cluster and the true yellow cluster are very close to each other. The algorithm will not be able to capture that and it will end up splitting both the purple and yellow clusters into two parts or will consider them as part of the same cluster. The same will happen in the case of the blue and the green clusters. KMeans will thus not do a good job.

- **DBSCAN** algorithm This algorithm will wprk only for the purple and green cluster, unlike the KMeans algorithm. It will however fail to cluster the blue cluster well as the points are very sparse. The outer regions of the red and the teal cluster will also get classified as outliers. If we increase *eps* or decrease *minPts* to combat the same, we will end up clustering the red and teal cluster together as their boundary points are close to each other. DBSCAN does well

on the clusters that KMeans does bad on and may not work for the blue cluster at all and classify the whole blue cluster as outliers.

- **Hierarchical Clustering with Single Link** None of the points from the blue cluster will get combined with Single Link Clustering. The green cluster will get combined first and the red and teal clusters will combined almost well enough. But the purple and yellow clusters will have the same problem and will probably get split into two clusters which splits both the yellow and the purple clusters or will get clubbed into the same cluster. The blue points will one by one get added to the Green cluster because they are very spread out among themselves. This method will thus do well on the green, red and teal clusters but it wont work with the blue, red and purple clusters. The sparseness of the blue cluster is what causes the issue.

- **Hierarchical Clustering with Complete Link** In this case, I suspect that Single Link and Complete Link will give similar clusters. This is because the yellow cluster is very spatially small. This will result in it getting coalesced first. The red and teal clusters are such that their densities seem to be becoming lesser as we move away from their cores. They will thus grow inside out. **One main difference** with respect to Single Link Clustering is that the blue cluster will get arbitrarily split into several clusters. This is because, like before the green cluster will get clustered first. Now since complete link is being used, the distance between blue points and green cluster increases. This results in blue points getting clustered together. The blue points will most likely get split into two halves. This might be desirable as compared to the previous case where the blue cluster got merged with the green cluster completely. Points from the purple cluster will get added to the yellow clusters as there are yellow points closer to the purple points as compared to diametrically opposite purple points.

### 1.1.3 D31

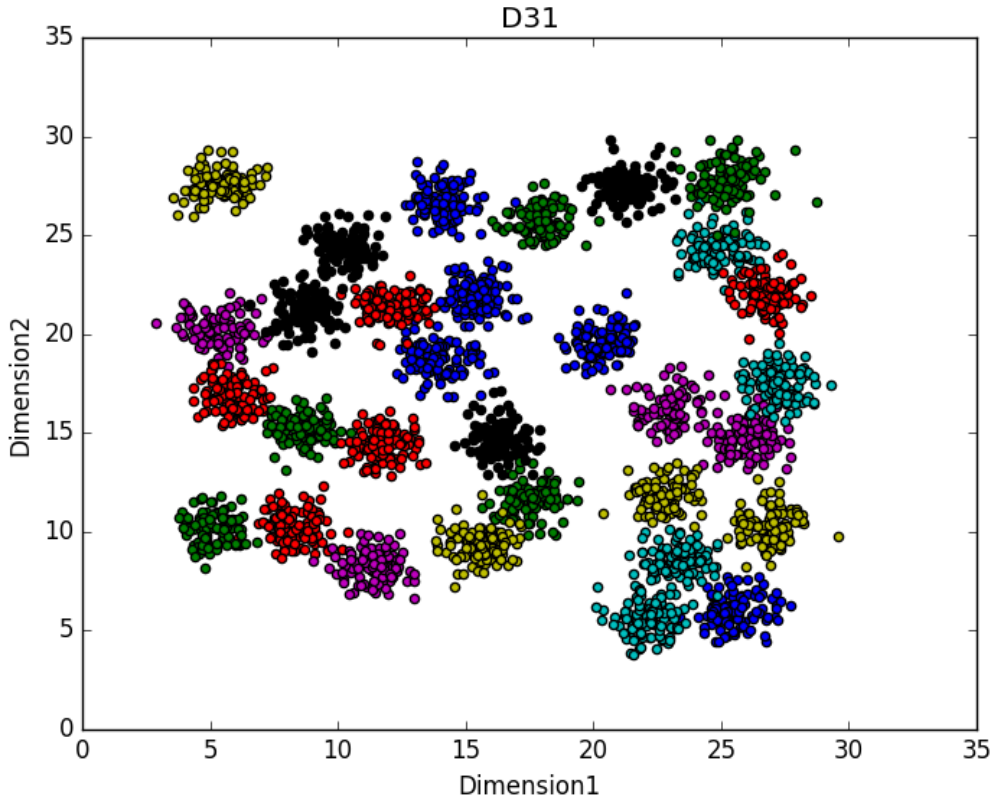


Figure 3: D31 Dataset

- **K-Means** algorithm should perform very well on this dataset because the clusters are all nice and convex in shape. This allows the centroids to converge properly to the centers of the clusters. Also, because of the fact that the clusters are all of similar sizes, KMeans will choose the right points for the right clusters.

- **DBSCAN** algorithm will also do well in this dataset because of the densely connected regions in the clusters. However, there are clusters like the teal and the purple ones which are very close to each other. Also, the clusters in the left are also close to each other. Thus, to get the right clusters, we need to do a Grid Search. Indeed, after a gridsearch is performed, we do get the correct clusters.

- **Hierarchical Clustering with Single Link** should perform very well in this dataset. Given the fact that the clusters all have a dense core and they are separated to a decent extent, Hierarchical Clustering will be able to find the clusters. Note also that Hierarchical Clustering is similar to DBSCAN or KMeans in some

cases. Both these algorithms perform well, hence this will also most probably perform well.

- **Hierarchical Clustering with Complete Link** should give very good results in this dataset. The clusters are compact and are all of similar sizes. This ensures the fact that complete link distances are actually distance measure between clusters, which is not the case with larger and skewed clusters. Indeed, this algorith gives a score of about 95%, which is very good for a dataset size of 3000 points.

### 1.1.4 Flames

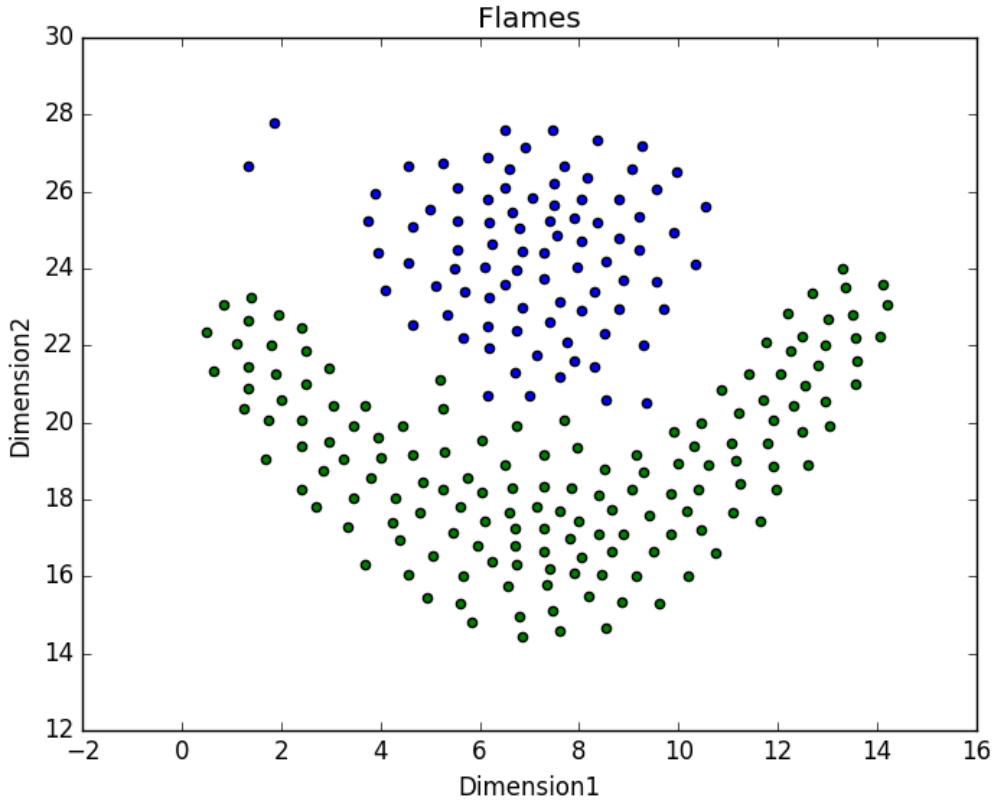


Figure 4: Flames Dataset

- **K-Means** will do only a satisfactory job in this case. It will be able to retrieve the blue cluster well, but it will run into issues in the Green Cluster as the centroid is not really a representative point of the same. It will thus include some blue points in the green cluster and thus cause some wrong clustering. In other initializations, it could include a few green points in the blue cluster. Other algorithms perform better.

- **DBSCAN** algorithm should do well on this dataset. Parameter tuning will be slightly tricky in this case because the region between the blue and the green cluster is not completely sparse and has a dense band of green followed by blue points. But with correct parameters, it should also be able to detect the outliers. Note that experimentally it can get a purity value of above 90% after wrongly classifying a few points as outliers.

- **Hierarchical Clustering with Single Link** should again do well on this dataset. The fact that both clusters are dense and not very concave, allows single link to traverse through the dense regions. Also since the cluster sizes are not highly imbalanced, Single Link will work very well. Infact in practice, it can get purity values of about 95%.

- **Hierarchical Clustering with Complete Link** will not work very well because there are points near the boundaries which are in a dense region. Complete Link will force those points to be part of the same cluster and this in the final cluster there will be significant number of wrong clusterings.

#### 1.1.5 Jain

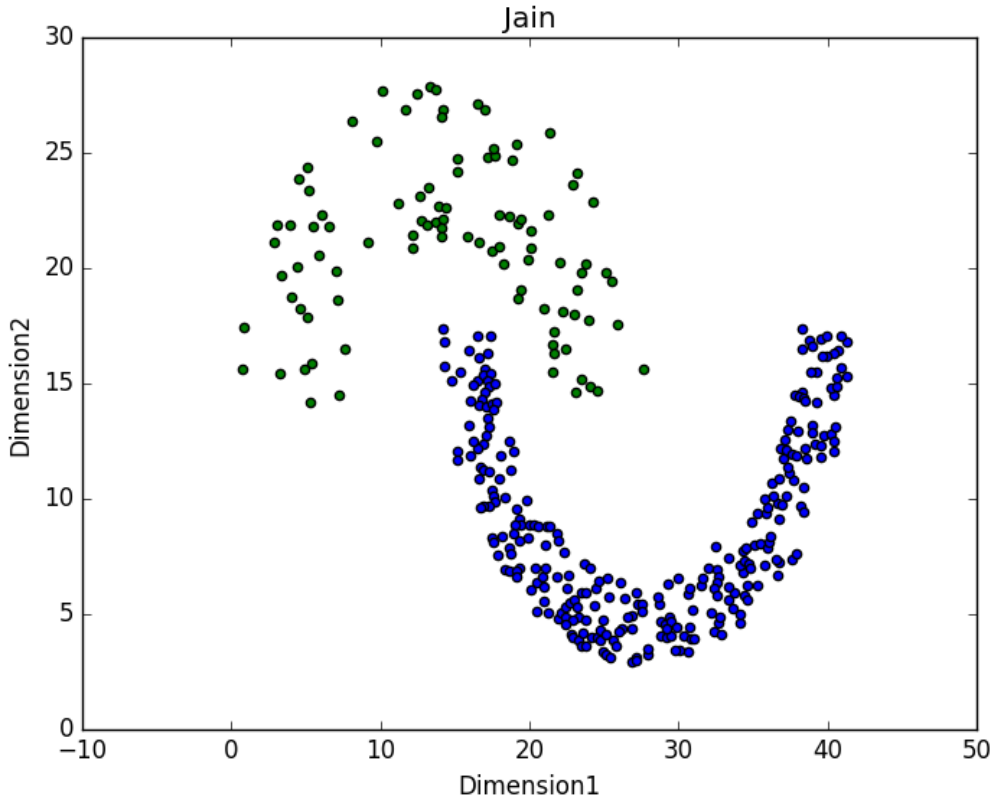


Figure 5: Jain Dataset

- **K-Means** is not expected to do very well on this dataset, but it still manages to get one complete cluster. It will end up retrieving the blue cluster completely, but it will not capture the non-linear boundary very well. As a result, some of the points from the blue cluster will end up getting classified in the green cluster. Following is an example of the run.

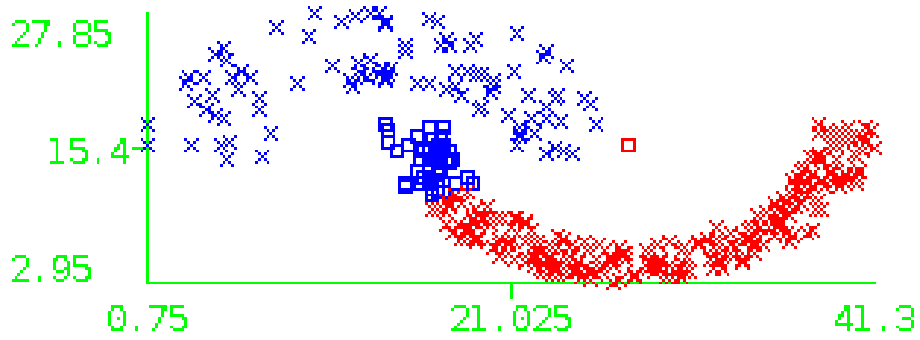


Figure 6: Jain Dataset

- **DBSCAN** algorithm is expected to do very well on these clusters because they are density connected between each other. There is also a considerable gap between the two clusters and choosing the right values of $eps$ and $minPts$ will ensure that the right clusters are retrieved. We thus expect almost perfect clustering. As can be seen in the $4^{th}$ subpart of the question, DBSCAN indeed does a good job.

- **Hierarchical Clustering with Single Link** does very well on this dataset. This is because of the densely connected regions which are utilized by single link distances. Note that there are many cases where **DBSCAN** and **Single Link** perform well/bad. They both use local densities in a way to cluster. This algorithm thus performs very well. A small catch however is to give $numOfClusters$ as some value greater than 2. In that case, it will retrieve some small outlier clusters, which when evicted would have given almost perfect clustering.

- **Hierarchical Clustering with Complete Link** performs very similar to KMeans algorithm. Unlike Single link, it is not able to capture the non linear boundaries very well. It thus ends up wrongly clustering a few points in the edge of the blue and the red cluster. This is because complete link will force some points across the boundary towards another cluster to be more similar "distance" wise. In practice, it actually performs very similar to KMeans algorithm.

### 1.1.6 Path-Based

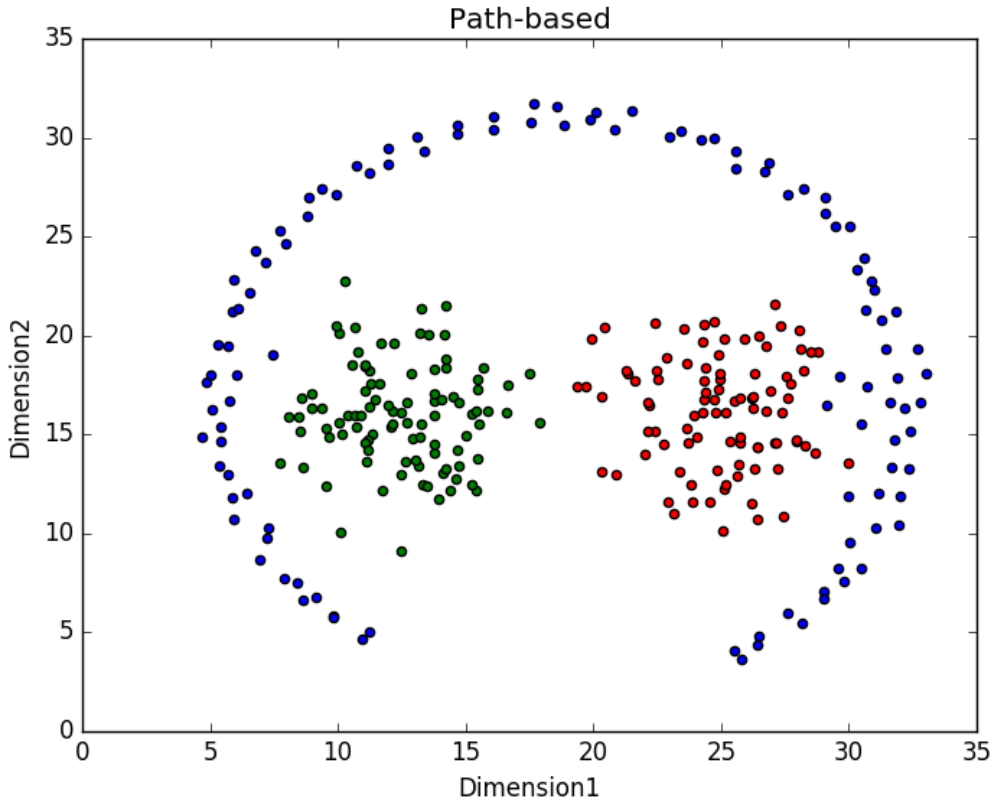


Figure 7: Path-Based Dataset

- **K-Means** algorithm will be able to retrieve the green and the red clusters. But the blue clusters' centroid will be somewhere in the middle and it is not a good representative point for the same. This forces the blue cluster to get arbitrarily divided into 2 clusters. As a matter of fact, this algorithm gives about 75% purity. It thus performs decently well. But it does not retrieve the blue cluster completely.
- **DBSCAN** algorithm is expected to do well on this dataset. But the prpblem here is that a lot of red points are close to the blue points. This forces the two clusters to get clubbed into one. To get around this, we need to reduce the value of *eps*. After tweaking the model, we get a purity of about 75%
- **Hierarchical Clustering with Single Link** actually performs the best on this dataset. The green and the red clusters are convex in shape and thus the clusters can be retrieved using the same very easily. Retrieving the blue cluster will be a problem. Like the other methods, that cluster gets split.
- **Hierarchical Clustering with Complete Link** works similar to Single Link. Green and Red Clusters will be retrieved well. There will however be problems

with the Blue Cluster. The purity for the previous method and this method is about 80 and 75 percent.

### 1.1.7 R15

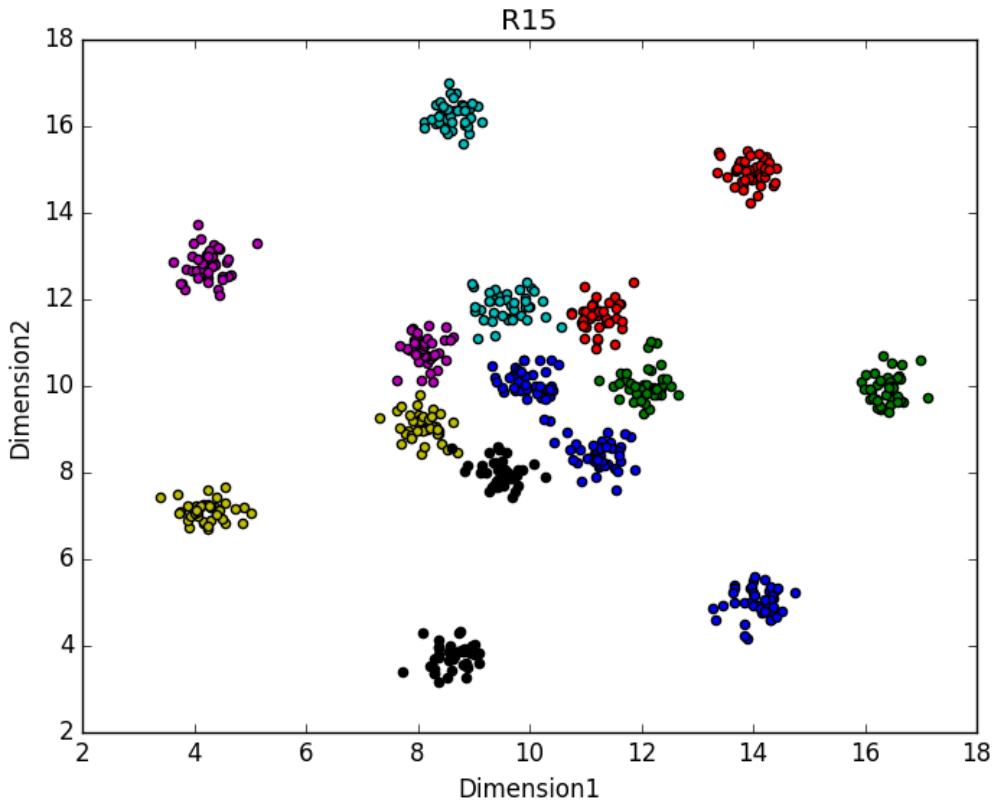


Figure 8: R15 Dataset

- **K-Means** algorithm will work very well on this dataset because of the conves shapes of the clusters. Pretty much all the clusters are well separated and their centroids are very good representative points. KMeans is expected to give very close to 100% purity, which it does in practice.
- **DBSCAN** algorithm
- **Hierarchical Clustering with Single Link** will be able to cluster the outer 7 clusters, but it will not be able to cluster the inner 8 clusters very well due to the fact that the points are very close to each other. It will thus do only a mediocre job on the inner 8 clusters and a pretty good job on the outer 7 clusters.
- **Hierarchical Clustering with Complete Link** will be more robust that Single link in this case. It does end up classifying the outer 7 clusters properly.

At the same time, the inner 8 clusters are clustered very well because of the fact that complete link uses farthest point. The scattered points near the boundaries of the clusters which are in some sense outliers, will not affect complete link. It indeed ends up getting a purity $\approx 99\%$

### 1.1.8 Spiral

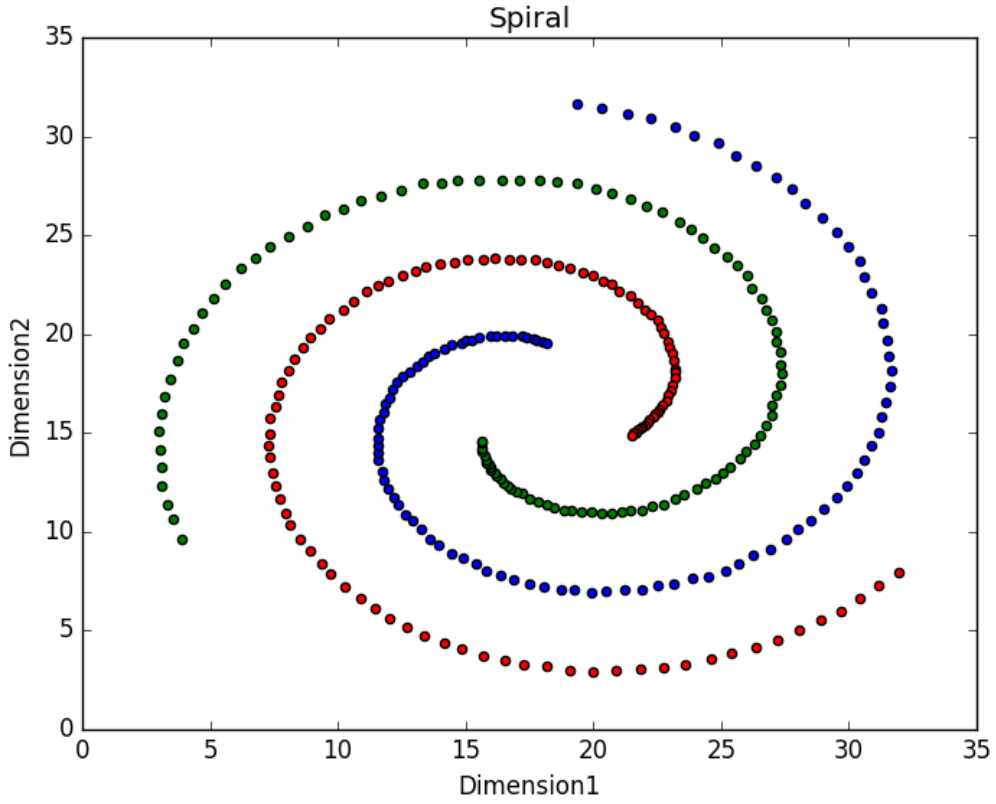


Figure 9: Spiral Dataset

- **K-Means** algorithm will fail very badly on this dataset. This is because the centroid of the true cluster are nowhere close to any of the points. The centroid of the blue cluster is actually very close to many green points as well. The nonlinear decision boundary will cause problems with KMeans.
- **DBSCAN** algorithm will do very well on this dataset. This is because the classes are very well separated amongst each other. There are large distances between any two points of difference classes. This, by choosing $minPts$ as 1 and choosing $eps$ as the distance between the points neaar the end of the spiral line, it is possible to recover the clusters completely.
- **Hierarchical Clustering with Single Link** does very well on this dataset. This is because, Single link distance looks at the closest points and then chooses

which cluster to assign to. Thus, it clustering the highly dense points near the centre. It then continues to cluster all the points. All along, since the Single Link distance is being used, the spiral is covered. We thus end up getting a 100% purity.

- **Hierarchical Clustering with Complete Link** does very bad on this dataset. It is quite the opposite of Single Link in this case. It starts off similar. But once the clusters grow larger in size, the complete link forces red and green, blue and green clusters to combine. This happens because there are many regions where the data points are close to each other. The purity ends up becoming something like 35%

## 1.2 KMeans with R15 Dataset

Weka does not automatically do multiple initializations for us. I thus use the Command Line Interface so that we can automate the seeding procedure.

### 1.2.1 Setting K=8

As can be seen in the R15 Dataset, there are 15 different clusters. When we set $K = 8$, we expect the outside 7 clusters to be captured, and the middle 8 clusters to be captured as 1 cluster. We run the algorithm multiple times and report the overall cluster purity.

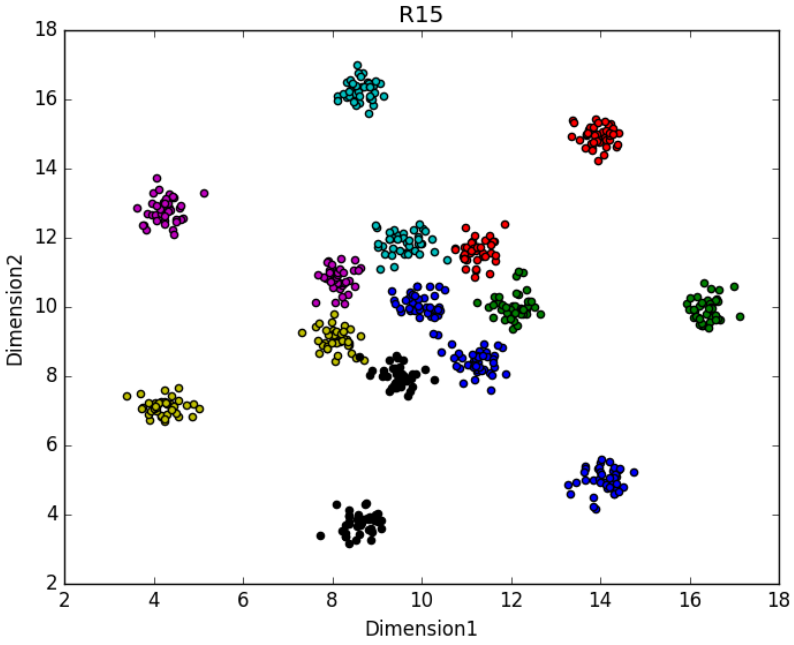


Figure 10: R15 Dataset

The algorithm was run for 100 different seed values between 1 to 100. The overall purity value is listed below.

| Seed | Purity |
|---|---|
| 2 | 53.33 |

This is expected because, the 7 clusters in the middle are all wrongly classified with respect to the true clusters. The following figure shows the clustering when $K = 8$.

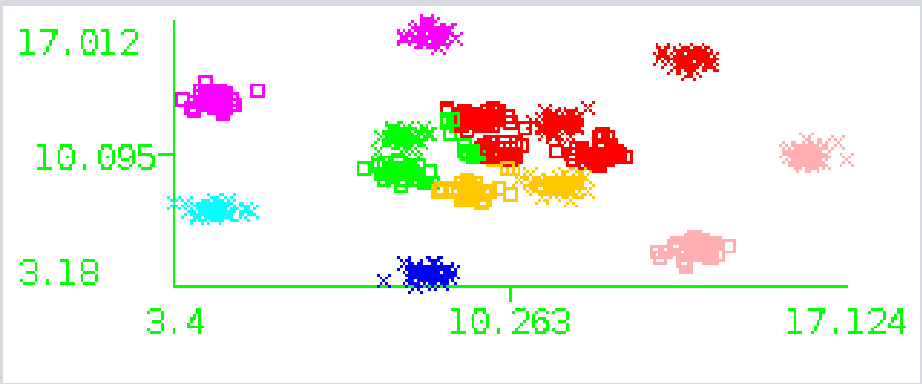


Figure 11: R15 Clusters for K=8

### 1.2.2 Varying K values for R15

As suggested in the question, the K values are varied from 1 to 20. 100 different random initializations for each K Values to get the best results. The following is the graph for the k values.

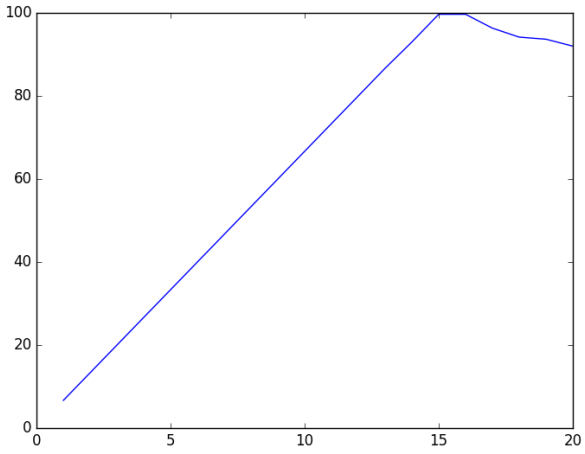


Figure 12: R15 Dataset

It is very interesting to see that when enough number of initializations are performed, the knee in the curve is very clearly visible. We can clearly see that the

slope is almost constant till the value $K = 15$. At that value, the Purity has reached 100% (As a matter of fact, it is 99.96%). This graph shows characteristics of a typical graph. As we increase the number of clusters, the purity keeps increasing. Once it hits the correct number of clusters, the slope of the graph reduces. In this case, the slope $\approx 0$. Since Weka uses a Rand Index kid of Purity to average the values, there is slight decrease in the value after the right number of clusters are hit because points which are supposed to be in the same cluster are going to different clusters. The following clusters show the best clustering that was done for the value of $k = 15$. Same colour does not mean the same cluster. There are exactly 15 clusters in this retrieved clustering. The blue cluster on the other hand is pretty dense and we can see that it will be retrieved very well. The gap between the blue cluster and the green cluster becomes very less near the edge of the blue cluster. We need to be wary of that.

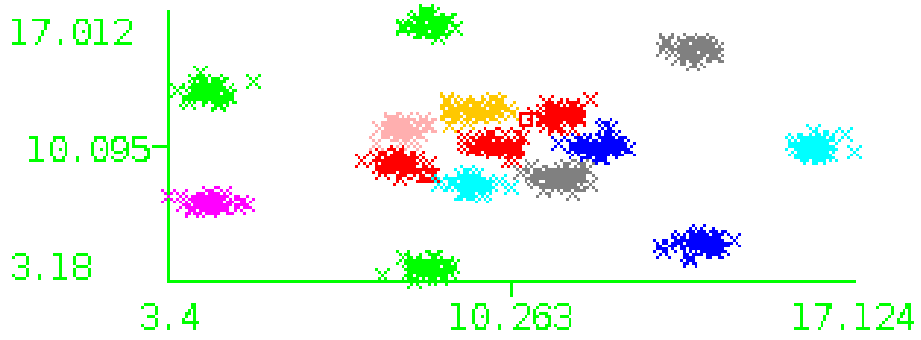


Figure 13: Clustering

## 1.3 DBSCAN with Jain Dataset

A few observations are in order before we start choosing the right values of *eps* and *minPts*. The image is included below for convenience. We can see that the green cluster is very small. Thus, having a large *minPts* or a small *eps* might end up splitting the green cluster into several clusters. At the same time, having a large *eps* will cluster all the data points in the same clusters, as happens in the following case, when $eps = 0.9$ and $minPts = 6$.

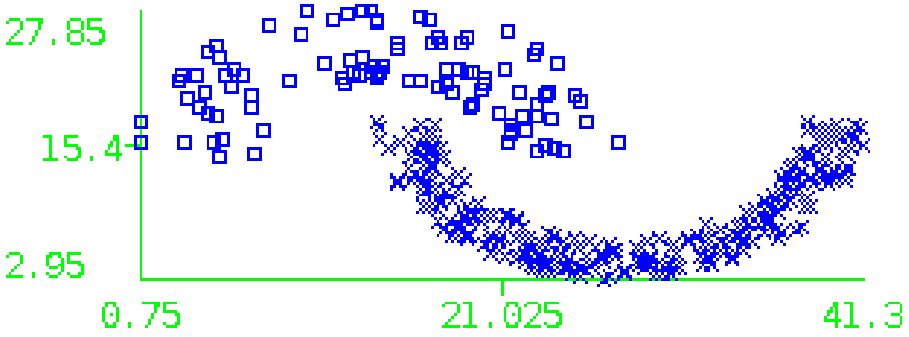


Figure 14: Clustering

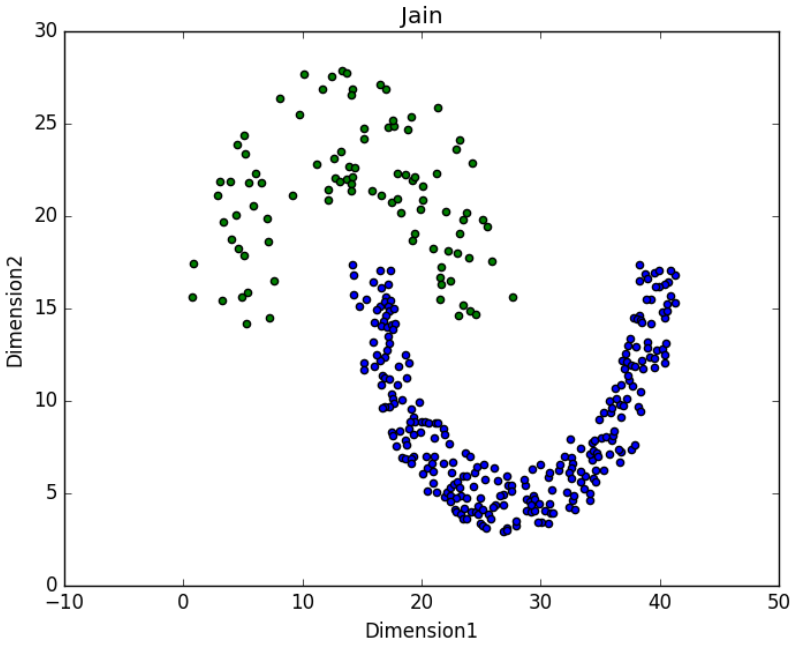


Figure 15: Jain Dataset

When $minPts$ is equal to 6, as we keep decreasing $eps$, there is initially only one cluster. After $eps = 0.05$, the data gets split into 4 clusters, which is not ideal in our case. This indicates that $minPts$ should probably be reduced. We use $minPts = 1$ to experiment with different values of $eps$ and then narrow down our search to a smaller range of $eps$. The following is the graph for Cluster purity with respect to $eps$ values. Note that for values $eps > 0.1$, it returns only 1 cluster and we are thus not very interested in the same. We narrow our search to $eps \in \{0, 0.1\}$. The first graph shows the variance of the number of clusters as we change the value of $eps$.

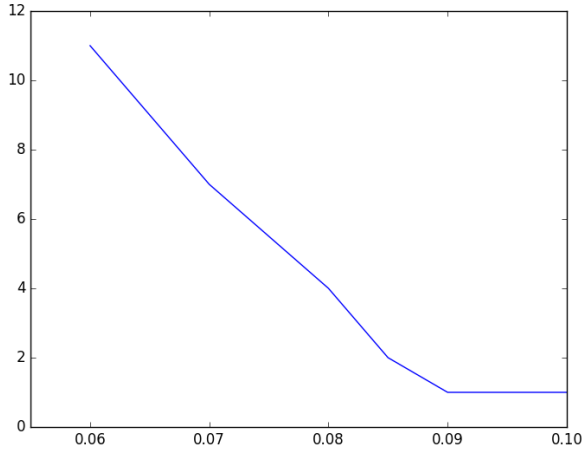


Figure 16: Number of Clusters

The following is the graph that illustrates illustrates how the Purity of cluster varies as we change the value of *eps*. Note that the sharp increase of purity allows us to choose the optimal value of 0.08.

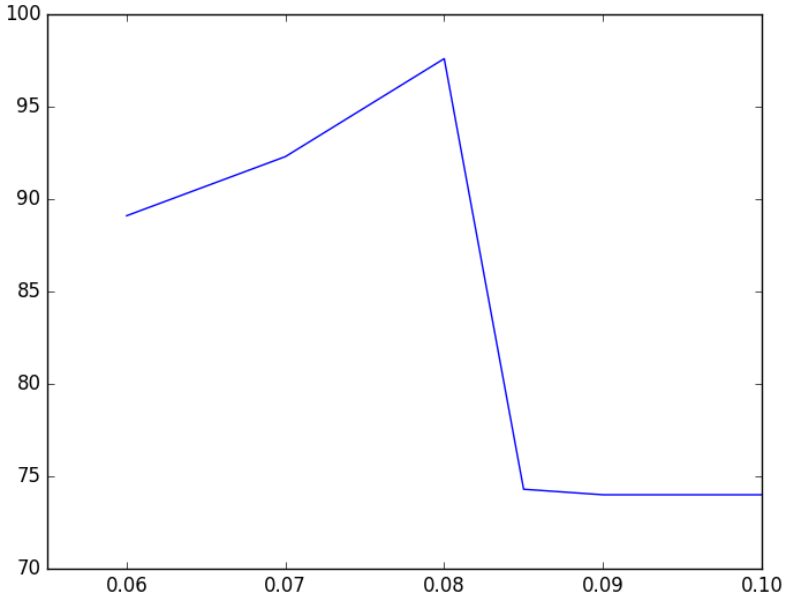


Figure 17: Purity vs Epsilon

An interesting fact to note is that when $eps = 0.08$, there are actually 4 clusters returned by the algorithm whereas there are just two clusters in the original dataset. This is mainly because of the sparse regions in the Green cluster. The two clusters

other than the main clusters can actually be considered as outlier clusters. The following graph clustering gives us a good idea about how DBSCAN is performing.

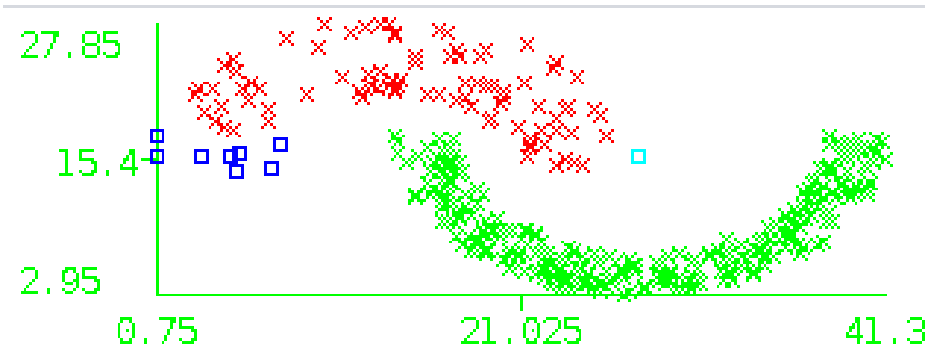


Figure 18: Purity vs Epsilon

From the figure it can be seen how the clusters are captured very well by DBSCAN. Only a few outliers near the edge of the data miss out.

We now change the value of *minPts* and *eps* together to see how the clusters change. We observe that for *minPts* values greater than 5 the clusters start splitting up. We thus explore for *eps* values near 0.08 and $minPts \in \{1, 2, 3, 4, 5\}$. The following surface plot shows how the Purity varying with both the values.

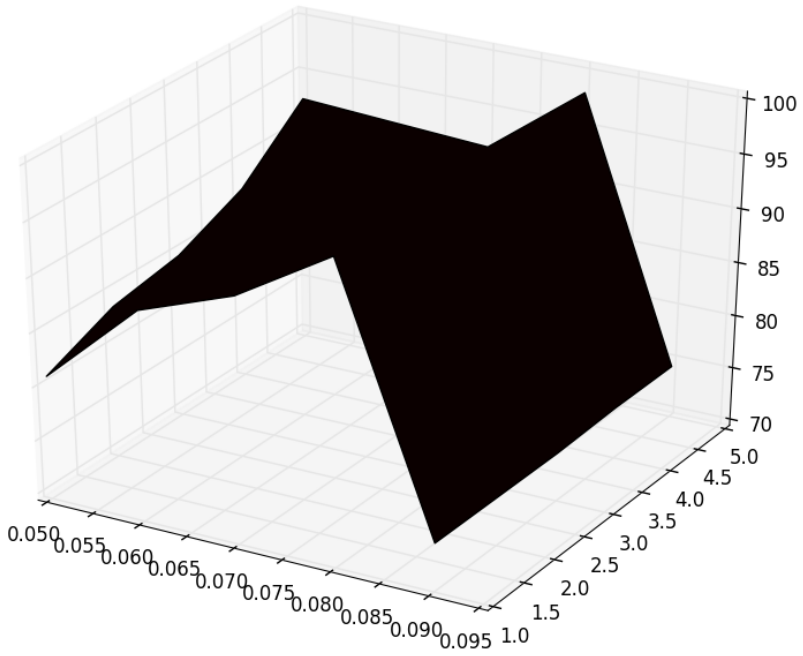


Figure 19: Surface Plot

The following is a table illustrating the same data.

| Values | 0.05 | 0.06 | 0.07 | 0.08 | 0.09 |
|---|---|---|---|---|---|
| 1 | 81.0 | 89.008 | 92.23 | 97.588 | 74.0 |
| 2 | 83.65 | 90.35 | 93.03 | 97.856 | 74.0 |
| 3 | 84.72 | 90.89 | 93.57 | 97.856 | 74.0 |
| 4 | 87.4 | 92.23 | 91.421 | 98.13 | 74.27 |
| 5 | 92.5 | 91.96 | 91.43 | 98.13 | 74.3 |

From the above table we choose the best purity value of 98.13%, which is obtained with $eps = 0.08$ and $minPts = 4$.

| Eps | minPts |
|---|---|
| 0.08 | 4 |

Using those values, there are only 3 clusters obtained. The following is the Clustered data.

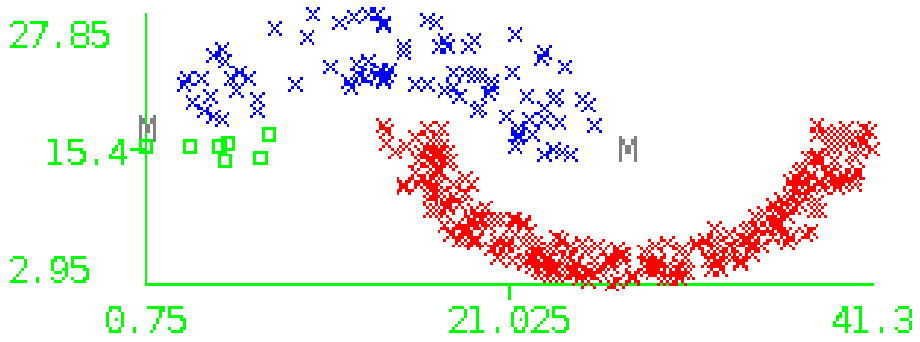


Figure 20: Best Clusters

## 1.4 Path-Based, Spiral and Flames

### 1.4.1 Path-Based

We first experiment with Hierarchical Clustering. Path Based is expected to have 3 different clusters. Following is the figure for reference. Giving $numClusters = 3$ apriori is not the right thing to do. We thus run Hierarchical Clustering on different for that parameter. We check the cluster assignments. If the cluster assignments are close to the actual assignments (even if there are more clusters than the true clusters) we accept that as a solution. The commandline script is used to iterate over a large number of cluster values to get the best cluster. After running the algorithm for different values of the number of clusters, for **Single Link**, 80.667% purity is obtained, with 7 retrieved clusters. The model is also penalized for retrieving more number of clusters than required. The following is the cluster assignment.

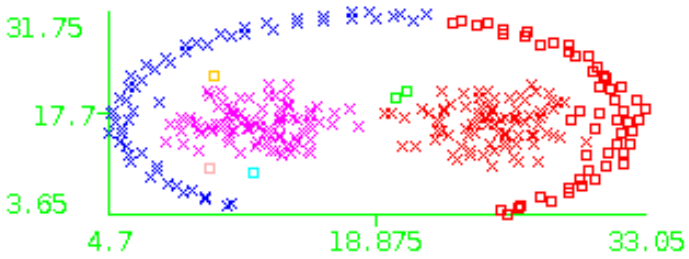


Figure 21: Single Link

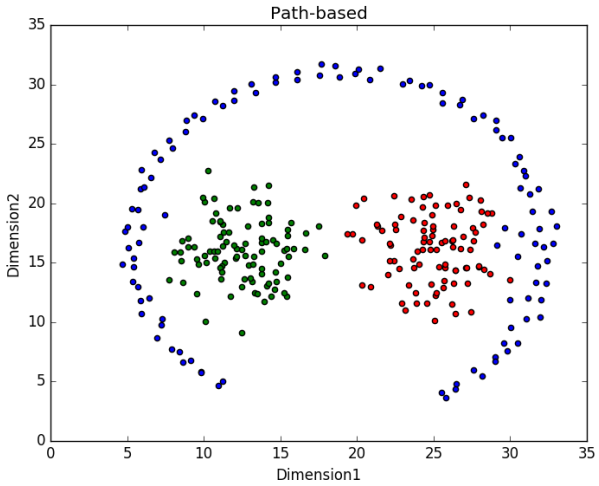


Figure 22: Path Based

We run the same kind of tests on all the different kinds of linkages and report the best values (reporting all clusters will be voluminous. The script does the work).

The following table summarizes all the different techniques used with Path-Based dataset.

| Model | Purity | Number of Clusters |
|---|---|---|
| Hierarchical - Single | 80.667 | 7 |
| Hierarchical - Complete | 70.667 | 3 |
| Hierarchical - Average | 73 | 3 |
| Hierarchical - Mean | 70 | 3 |
| Hierarchical - Centroid | 73.3 | 3 |
| Hierarchical - Ward | 75.33 | 3 |
| Hierarchical - AdjComplete | 64.5 | 5 |
| Hierarchical - Neighbour Joining | 80.667 | 7 |
| DBSCAN | 73.6 | 6 |

Note that *NEIGHBOUR_JOINING* and Single link actually gave the best answer. The best possible cluster for Hierarchical case is as shown before.

DBSCAN gives the following cluster assignments.

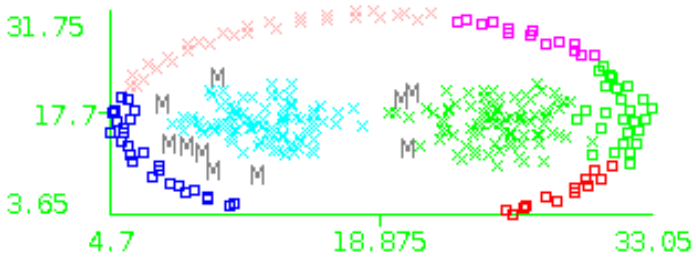


Figure 23: Path Based

### 1.4.2 Spiral

As noted before, Hierarchical Clustering with Single Link is supposed to work wonders in this case and give almost perfect clustering. DBSCAN will also give very good clustering as it moves only through the highly dense regions. The following table summarizes the results.

| Model | Purity | Number of Clusters |
|---|---|---|
| Hierarchical - Single | 100 | 3 |
| Hierarchical - Complete | 38.141 | 3 |
| Hierarchical - Average | 38.141 | 3 |
| Hierarchical - Mean | 61.859 | 3 |
| Hierarchical - Centroid | 40.384 | 3 |
| Hierarchical - Ward | 59.61 | 3 |
| Hierarchical - AdjComplete | 61.85 | 5 |
| Hierarchical - Neighbour Joining | 40.38 | 3 |
| DBSCAN | 100 | 3 |

In both the cases of Hierarchical with single link, the following is the cluster assignment.

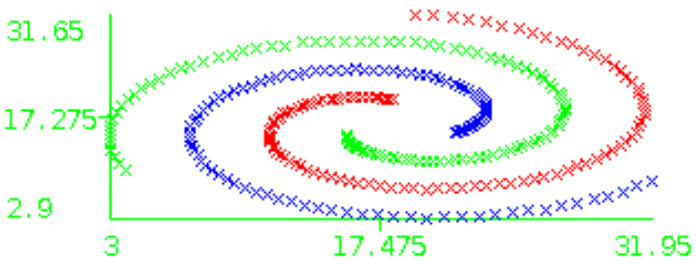


Figure 24: Spiral Dataset

### 1.4.3 Flames Dataset

This dataset looks like it will perform well with Hierarchical Clustering. Near the boundary of the green and blue clusters, the density of the points remains almost

the same. For DBSCAN to work, we would thus have to tune the parameters very carefully.

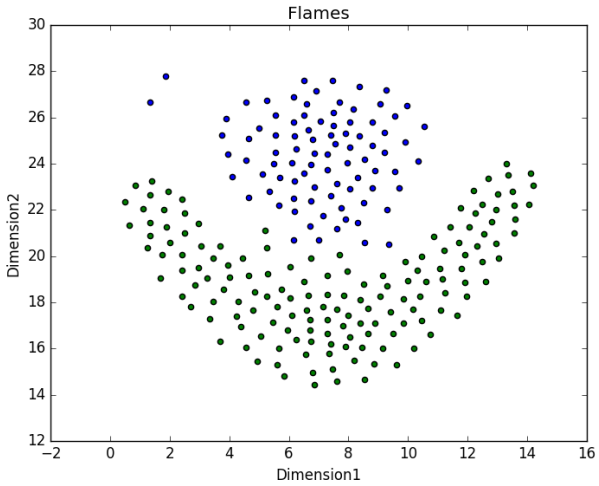


Figure 25: Flames Dataset

The following are the results for the Clustering techniques. *Ward* linkage does the best in Hierarchical Clustering.

| Model | Purity | Number of Clusters |
|---|---|---|
| Hierarchical - Single | 95 | 11 |
| Hierarchical - Complete | 69.583 | 3 |
| Hierarchical - Average | 83.33 | 2 |
| Hierarchical - Mean | 92.08 | 2 |
| Hierarchical - Centroid | 86.25 | 3 |
| Hierarchical - Ward | 100 | 2 |
| Hierarchical - AdjComplete | 64.16 | 2 |
| Hierarchical - Neighbour Joining | 63.75 | 1 |
| DBSCAN | 92.5 | 5 |

As can be seen, after tuning DBSCAN well, with $minPts = 2$ and $eps = 0.06$, we get a purity of 92.5%. Ward linkage however does almost perfect clustering. But noe that there are outliers in the dataset. DBSCAN does a good job in finding outliers in the dataset. Hierarchical Clustering however is not exactly able to the outliers. The following is the clustering for Hierarchical with Ward Linkage.

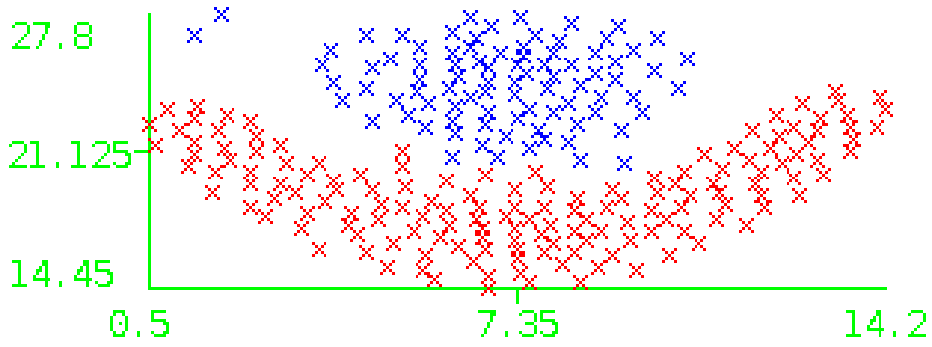


Figure 26: Ward

The following figure illustrates the clustering in the case of DBSCAN. $M$ points denote the outliers.

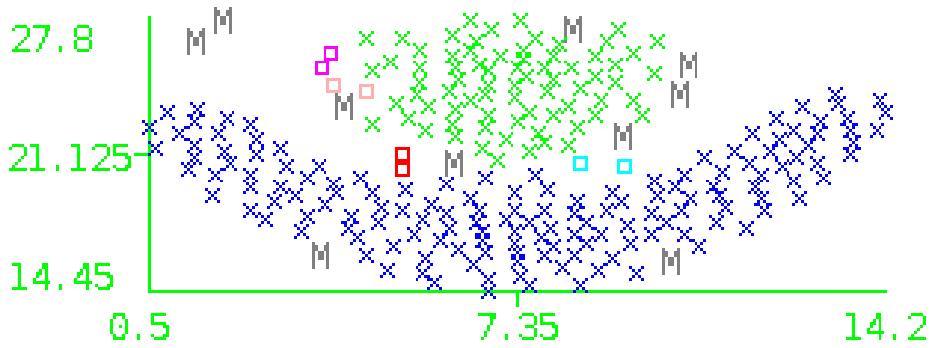


Figure 27: DBSCAN

## 1.5 D31 dataset

In this question, because there is no explicit mention of which evaluation metric to use, we use sklear's *v_measure* function which is essentially the harmonic mean of *completeness_score* and *homogeneity_score*.

### 1.5.1 KMeans Clustering

When the algorithm is run on several values with the number of clusters as 31, the best score achieved is as follows.

| 96.8 |
|---|

With the value of $K = 31$, all the clusters are retrieved. The following is the visualization.

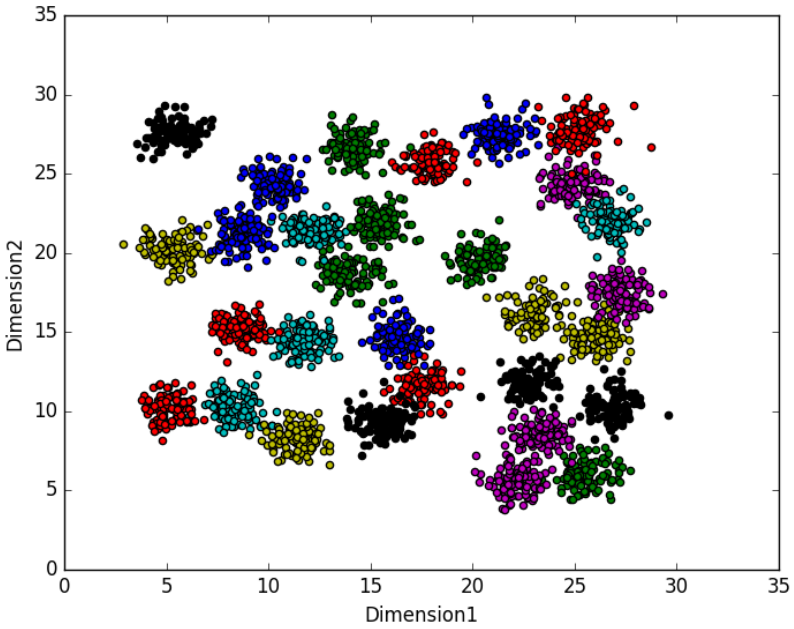


Figure 28: KMeans

When $K = 32$, one of the clusters will get split into two because there are more number of centroids now. The following diagram shows the same. Notics that one cluster is now comprised of green and black points both.

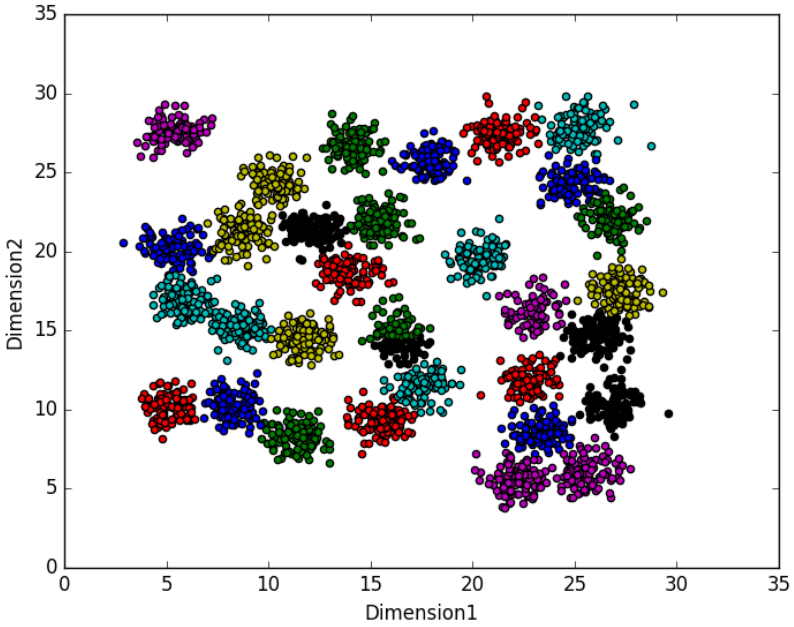


Figure 29: KMeans

As and when we increase K, the number of different clusters it gets split into increases. This results in random clustering of points. The following plot shows the same for $K = 45$

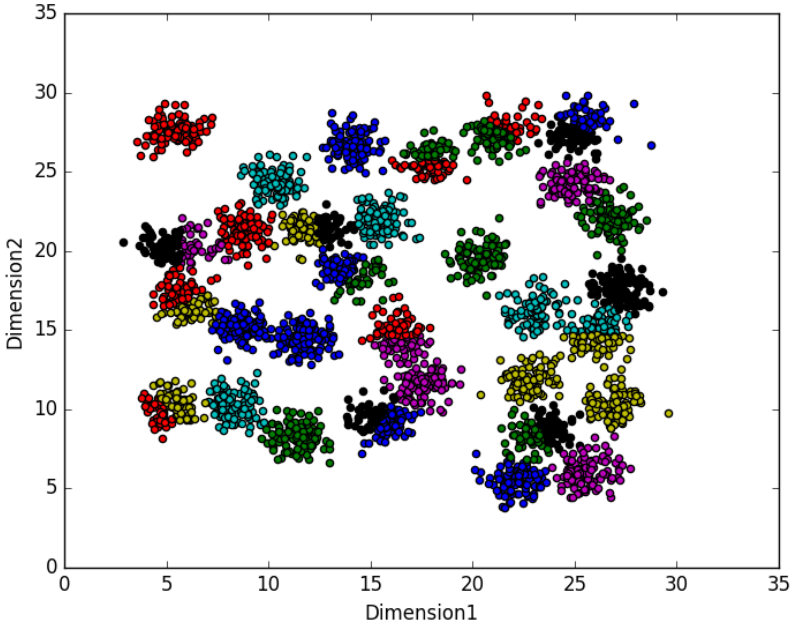


Figure 30: KMeans with K=45

### 1.5.2 DBSCAN algorithm

From the Diagram it can be seen that a lot of clusters have dense regions near the boundaries. Due to this, we will have to tune the parameters of DBSCAN well to make sure it does not consider all the points together. The following is the clustering for default values. We can see that a lot of clusters are getting coalesced.

| Score | Epsilon | MinPts |
|---|---|---|
| 83.7 | 0.5 | 5 |

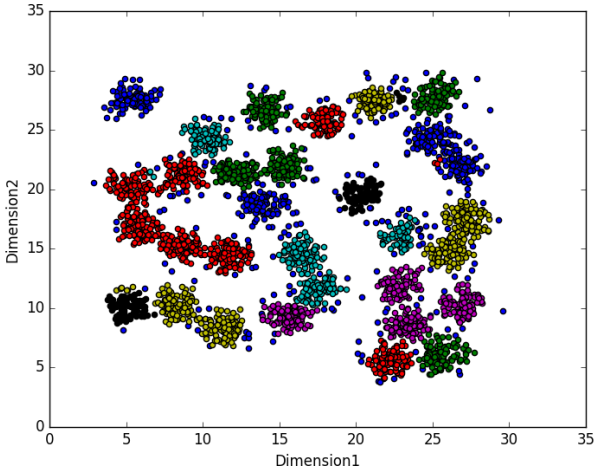


Figure 31: Default Values

We do a grid search to get the optimal values of the parameters. After doing the same, the following are the results. Note the differences between this and the default value clustering.

| Score | Epsilon | MinPts |
|---|---|---|
| 92.4 | 1.3 | 66 |

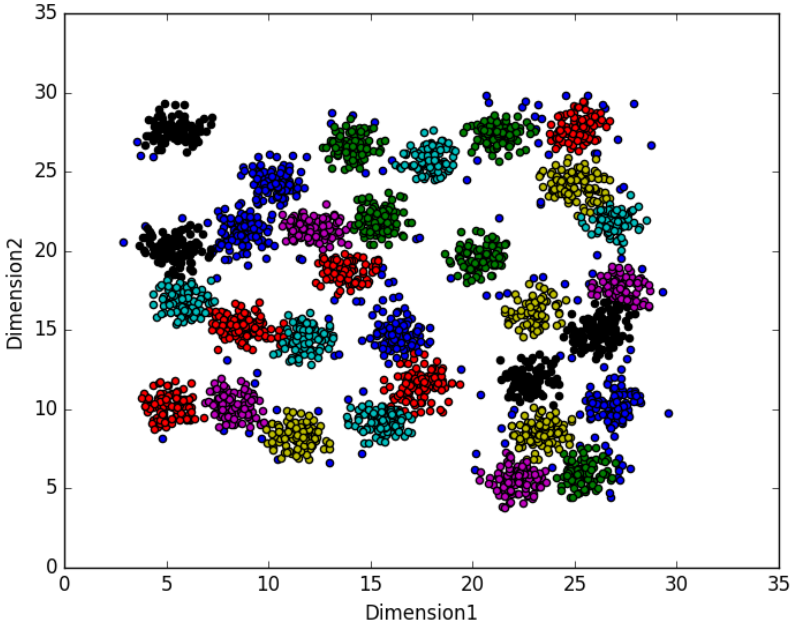


Figure 32: Optimal Values

### 1.5.3 Hierarchical Clustering

We now check hierarchical Clustering with Ward Linkage. This performs very well on the dataset and gives results as follows.

| 95.08 |
|---|

The clustering is almost perfect. Note that Ward Linkage uses squared distance amongst the cluster to check for clustering. This will results in compact clustering which is ideal in this case. The following is the figure for clustering.

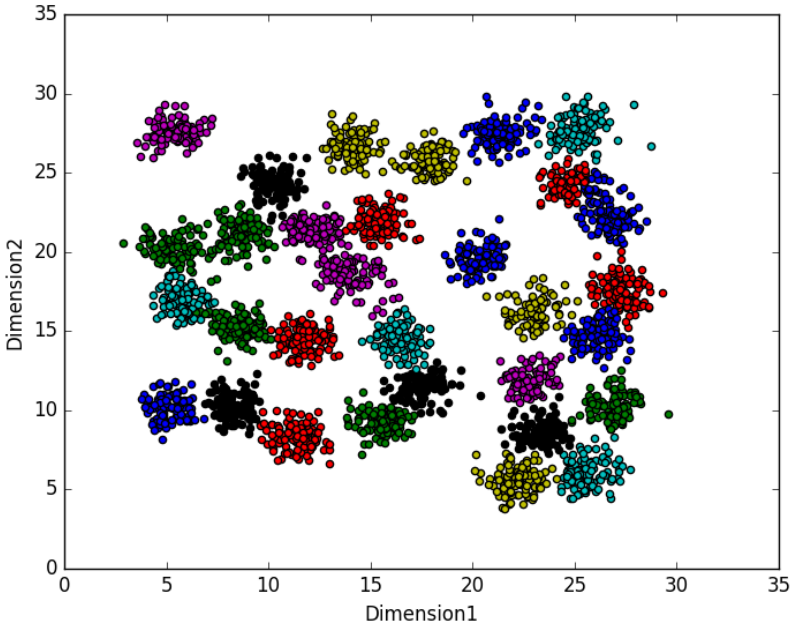


Figure 33: Hierarchical Clustering with Ward Linkage

# 2 Question 2 - Naive Bayes

## 2.1 Generating the train-test matrices

- The text files that are provided have numbers in them which indicate what that word is. The subject of the email is to be ignored. We generate two large matrices, the first one being a data matrix of size $p \times k$ and the second being the labels, $p \times 1$. $k$ is the vocabulary size and will thus have to be fixed after going over all the documents. Here we assume that the test datasets available are the only ones which will be used to test. For online learning with Naive Bayes, we will have to denote such new words using some token like, $¡TOK¿$. We also note that the ordering of the words do not matter as we are assuming a unigram language model. The bag-of-words approach assumes that there is

no extra information that the ordering can give.
One implementation note. It is usually a common practice to use Sparse matrices to represent data in Natural Language Processing applications. This is because the vocabulary size is huge and many words end up not appearing in the documents. For our purpose, there are about 110 files in each directory and there are 10 directories. A normal matrix should work in our case. We use the sparse matrix however because the normal matrix takes about 30 seconds to load while the sparse matrix take about 0.30 seconds.
After analyzing the text data provided, it can be seen that the vocabulary size overall is 24634 words, but the largest token number is 24747. This is because there are probably a few tokens that appear only in the subject. Since we have decided to ignore the subject, the words do not appear in the vocabulary. However, since the difference between the two number is very small (113), we choose to keep the vocabulary size as 24747 for all purposes. The total number of documents are 1099. Thus, the size of the whole data matrix will be $1099 \times 24747$. Note that we are keeping all the documents together so that we can perform cross-validation easily.

## 2.2 Multinomial Likelihood

- For the Multinomial case, we need to estimate the priors $p(C)$ and we need to estimate $p(w|C)$. The latter is calculated by just using the counts associated with that word. The following formula can be used to calculate the conditional probability.

$$p(w|c) = \frac{W_{cw} + 1}{\sum_{w'}(W_{cw'} + 1)}$$

  The notation has been borrowed from the Information Retrieval book that has been suggested. Note that Add-1 smoothing is performed here.

- We first calculate the class priors using simple counts. We note that $p(spam) \approx 0.43$ and $p(ham) \approx 0.56$. This ratio is almost constant across all the 5 splits. The datasets are not unbalanced and that is good news.

- At prediction phase, we just choose the class which has a higher likelihood $\prod p(c|w)$. We need to make sure that we take a log so that we don't run into underflow problems. The following table illustrates the accuracy scores of the classifier averaged over 5 folds.

| Class | Precision | Recall | F-Score |
|---|---|---|---|
| Spam | 0.951 | 0.973 | 0.961 |
| Ham | 0.978 | 0.961 | 0.969 |

- As can be seen, this naive model itself does very well. Note that in the case of Spam Classification, we usually require the recall of spam emails to be very high as we do not want to miss any spam emails. We also want the recall of

ham emails to be 1 as we do not want any legitimate email to get classified as spam as it could potentially contain important information.

- The PR curves can be obtained by varying the threshold used to decide which class the document belongs to. Usually, the threshold for ratio of $\frac{P(spam|d)}{P(ham|d)}$ is set at 1. But by changing the values, we can get different Precision and recall. As can be seen from the curve, the Precision score decreases as the recall increases. This establishes the precision recall trade off.

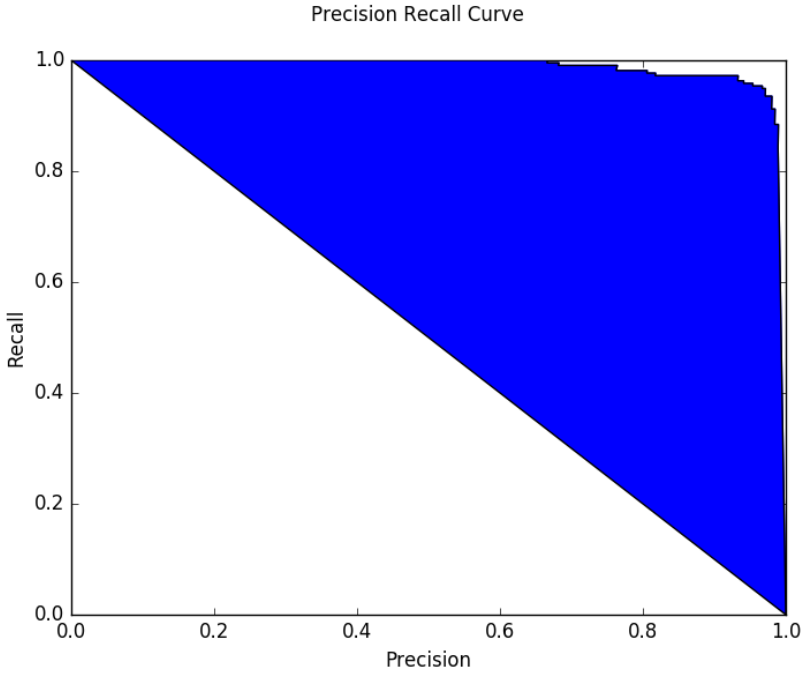


Figure 34: Multinomial

## 2.3 Bernoulli Likelihood

- As can be seen from the problem, there are essentially two ways of estimating the conditional probability $p(w|c)$. The way the Multinomial Model does it is to count the fraction of words in all the documents belonging to class $c$. This is thus a generative model where each word in the new (test) document is generated.

- In the Bernoulli Model however, it is a multivariate Bernoulli distribution. There are thus $k$ distributions, where $k$ is the number of words in the vocabulary. Each distribution is given by $BERNOULLI(i, p)$, where $i$ is the indicator variable. We follow the same convention of 1 indicating ham and 0 indicating spam. This distribution mainly estimates the fraction of documents that contained the word when the class was spam/ham.

- This model is different from the Multinomial model like the following. There might be very large documents where the word occurrences might be very large.

These will affect the Multinomial Model more than the Bernoulli model because the Bernoulli model cares only about the presence/absence of the word in the document. It is thus less affected by variation in length of documents. This also means that this model makes more mistakes on long documents.

- The priors that are calculated in the previous case can be reused here. They will remain the same as it is an empirical estimation. The conditional probabilities will however end up becoming

$$p(w|c) = \frac{N_{cw} + 1}{(N_c + 2)}$$

. Smoothing has already been performed. Here $N_{cw}$ represents the number of documents in class $c$ in which the word $w$ appeared. $N_c$ represents the total number of documents belonging to the class $c$. There can be easily estimated.

- The precision scores for the two classes are given below.

| Class | Precision | Recall | F-Score |
|---|---|---|---|
| Spam | 0.983 | 0.835 | 0.902 |
| Ham | 0.886 | 0.988 | 0.934 |

- The PR curve is plotted like in the previous case. The main difference with respect to the previous graph is the lower recall of Spam articles and higher precision of the same. We can thus choose a different threshold and see how the precision varies.

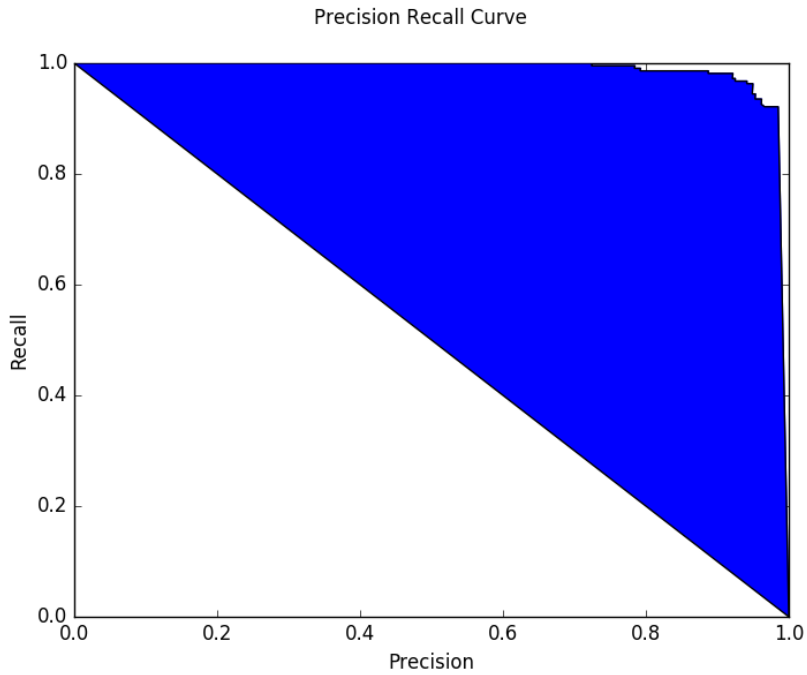


Figure 35: Bernoulli

- By choosing different thresholds, we can get the following scores.

| Class | Precision | Recall | F-Score |
|---|---|---|---|
| Spam | 0.94 | 0.96 | 0.95 |

  Using the curve, we can thus locate points which give more suited values for us and use that corresponding threshold at test time. Note how these scores are much better than in the previous case.

## 2.4 Bayesian Parameter Estimation with Dirichlet Prior

The Dirichlet Distribution for variables $z_1, z_2, \ldots, z_k$ are characterized by $k$ different parameters, $\alpha_1, \alpha_2, \ldots, \alpha_k$ such that $z_1 + z_2 + \cdots + z_k = 1$ to make it a probability distribution. It is characterized by $f(z_1, z_2, \ldots, z_k; \alpha_1, \ldots, \alpha_k) \propto \prod_{i=1}^{n} z_i^{\alpha_i - 1}$. This is actually thus considered one kind of smoothing because in the prior information we can tell the fact that a word does appear. Even if the word does not appear in the training set, the probability of the same will not be 0 and will instead be some value defined by the prior.
Note that we are assuming the Dirichlet distribution only over the priors of the words in vocabulary and not for classes. Dirichlet is a multi variate generalization of Beta distribution. Since the Dirichlet Distribution is conjugate to the Multinomial, we get the following formula for the conditional probabilities.

$$p(w|c) = \frac{W_{cw} + (\alpha_i - 1)}{\sum_{w'}(W_{cw'} + 1) + \sum_i(\alpha_i - 1)}$$

The same formula can thus be written as the following.

$$p(w|c) = \frac{W_{cw} + (\alpha_i - 1)}{\sum_{w'}(W_{cw'} + 1) + \sum_i(\alpha_i) - K}$$

Here as usual, $K$ denotes the number of words in the vocabulary. Note that we are using the MAP estimate here. The previous two cases which did not have have any priors and thus we could use the Maximum Likelihood Estimates. Now that we have priors, we can take the MAP estimate. Note how this is very similar to smoothing. Having $\alpha_i = 2$ enables the Laplace Smoothing that we have considered so far. Theory developed here is borrowed from here. We assume that the Dirichlet Priors are the same for every class. Intuitively, we have pseudo counts for each word in a given class. When we run the Dirichlet code with $\alpha = 2$, we get the exact same results as that of Multinomial Naive Bayes. This is because, it is just Add-1 smoothing.

| Class | Precision | Recall | F-Score |
|---|---|---|---|
| Spam | 0.951 | 0.973 | 0.961 |
| Ham | 0.978 | 0.961 | 0.969 |

As the values of $\alpha's$ are all increased, The precision, recall and F-Score keeps falling. This happens because the priors are stronger than in the previous case where 0 word count was given to all the words, and at the same time the priors are uniform, which is obviously not the case in real life. Thus, we end up with worse evaluation metrics.

### 2.4.1 Varying the parameters

There are about 24747 different words in the vocabulary. There are those many number of alphas. It is obviously impossible to vary all those parameters and then check how the precision and recall is varying. We thus need to come up with heuristics. Not all words in the dataset are predictive of the class a document will belong to.

- Words which are very frequent in the English language will appear in both the Spam and the Ham dataset. This however does not provide any support about which class the document actually belongs to. Words like *this*, *the*, *an* fall under this category.
- Another set of words which are very infrequent are also not very predictive of the class. These are chance words. Words like *stupendous*, which might have by chance come only in Spam emails, may get a very high probability score for $p(w|Spam)$. This is however just a chance. It is highly likely however that if the word *stupendous* appears in any of the documents, it will get classified as spam.

Considering the above points, we use some heuristics to pick which words we want to vary the parameters for. We only choose the words which have high $p(w|Spam)$ and $p(w|Ham)$ scores. We also omit words which have high scores for both these probabilities. These are in general referred to as $StopWords$. Also, the priors over both the classes will be different. There are thus two sets of $\alpha's$, one for each class. Intuitively what this means is that, we need to give different priors for different classes. We might want to increase the probability of $p(free|Spam)$, but not $p(free|Ham)$.

After using the heuristics defined before, following are the words obtained for Spam and Ham.

Spam Words: [46, 58, 63, 75, 79, 81, 83, 85, 124, 174, 1846]
Ham Words: [67, 69, 75, 79, 81, 83, 85, 117, 126, 174, 252]

The overlap between the words can clearly be seen. After removing the overlapping words, we get the final words as the following.

Spam Words: [46, 58, 63, 124, 1846]
Ham Words: [67, 69, 117, 126, 252]

This result is very interesting because, when we consider 20 words instead of 10 words and perform the same experiment and extract the words, the probability difference between the words is pretty large. The following table compares the $p(w|spam)$ and $p(w|ham)$.

| Spam | Ham |
|---|---|
| 0.0129441010945 | 0.000854449537317 |
| 0.0185408970065 | 0.0017123583845 |
| 0.00430549321548 | 0.000127994465104 |
| 0.00535856092256 | 0.000515437170285 |
| 0.00323516210337 | 0.000124535155237 |

Final Spam words used are $\{46, 54, 58, 774, 6175\}$.

This shows that these set of words are important for Spam and they don't appear in Ham much. We change the corresponding $\alpha's$ for this and increase the pseudo counts and see how the precision recall etc change by plotting the PR Curve. In the first experiment, we increase the pseudo counts of corresponding alphas in the Spam class and do not change anything for the Ham class. The following are the values obtained.

| Class | Precision | Recall | F-Score |
|---|---|---|---|
| Spam | 0.957 | 0.973 | 0.964 |
| Ham | 0.978 | 0.966 | 0.972 |

These values are actually better than the values with no Dirichlet Prior Distribution. Note the $\alpha$ values are set to be 1000 because those are approximately the number of documents we are looking at. This puts things on equal footing. The following is the PR Curve for this case. It is very close to the axis, which means that our priors are very good and the heuristic we used to determine the priors has worked very well.

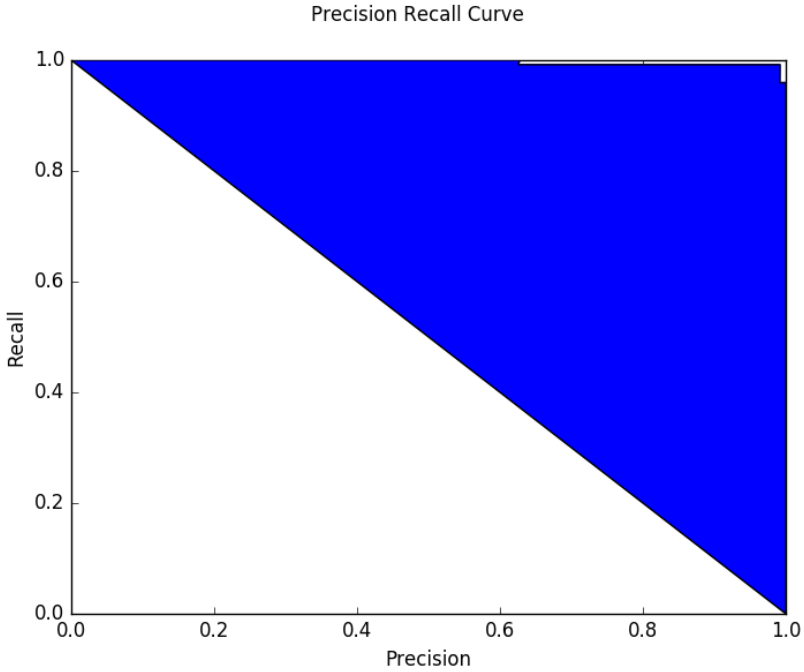


Figure 36: Dirichlet with Spam prior varying

We now vary only the ham priors and check what the results are.

| Class | Precision | Recall | F-Score |
|---|---|---|---|
| Spam | 0.941 | 0.975 | 0.958 |
| Ham | 0.98 | 0.953 | 0.966 |

As can be seen, this alone does not do any better than no prior case. The following is the PR Curve for this case.

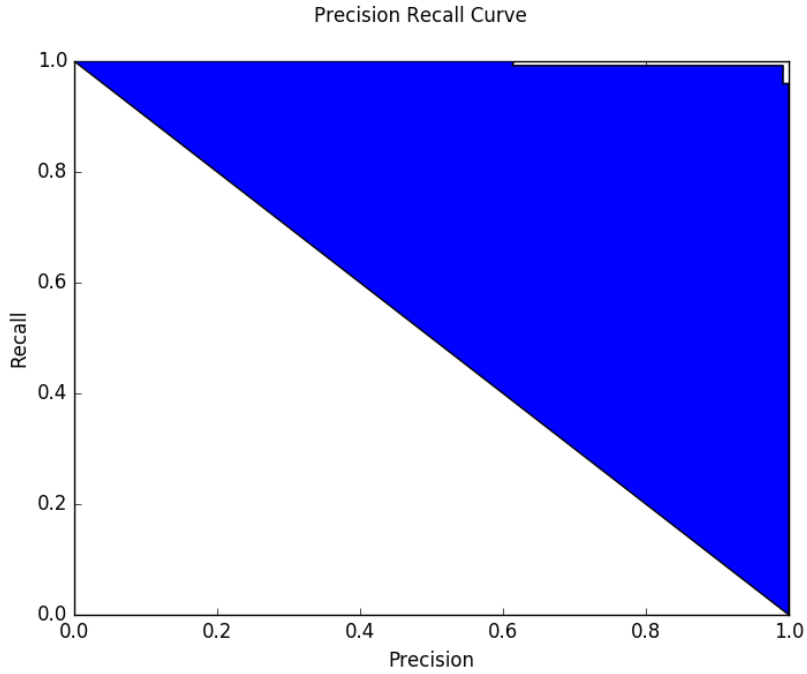


Figure 37: Dirichlet with Ham prior varying

We now vary both these parameters and check the values. Both the $\alpha's$ are changed.

| Class | Precision | Recall | F-Score |
|---|---|---|---|
| Spam | 0.968 | 0.958 | 0.963 |
| Ham | 0.968 | 0.975 | 0.971 |

**We see that there is an improvement with respect to the normal case mainly because of the spam words that are present. This means that spam words are more indicative of the document than the ham words. Words in ham can in general also appear in spam, but not the other way round.** This allows us to exactly determine which words are very important to spam. Notice how the corner is different compared to other curves.

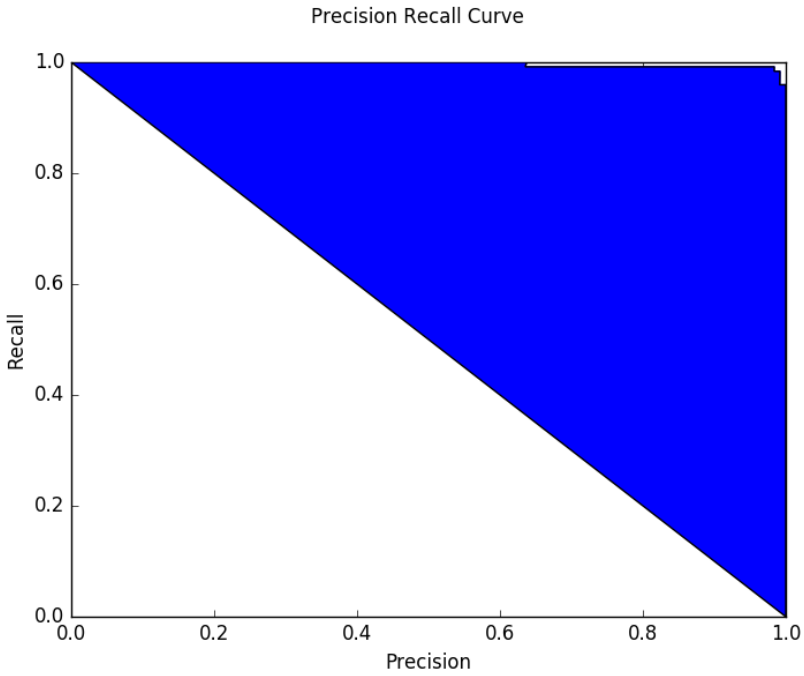


Figure 38: Dirichlet with both priors varying

## 2.5 Bayesian Parameter Estimation using Beta Distribution Prior

In this part of the question, we are basically varying the class priors to check how the precision and recall vary. We use the $Beta(\alpha, \beta)$ function to give the priors. Like in our coin toss example, the previous probability for Spam and Ham Classes were,

$$p(C) = \frac{N_c}{N}$$

. After using the priors, the equation becomes,

$$p(C) = \frac{N_c + (\alpha - 1)}{N + \alpha + \beta - 2}$$

. The code is modified to incorporate this.

### 2.5.1 How the parameters have to be varied

These parameters will dictate how many spam articles we have seen so far and how many ham articles we have seen so far. This may depend on the domain knowledge, or the user's previous history, the person's job etc. Checking user history and seeing how many spam and ham emails are received is probably one good way to check and assign these probabilities. The following graph shows the statistics for real world.

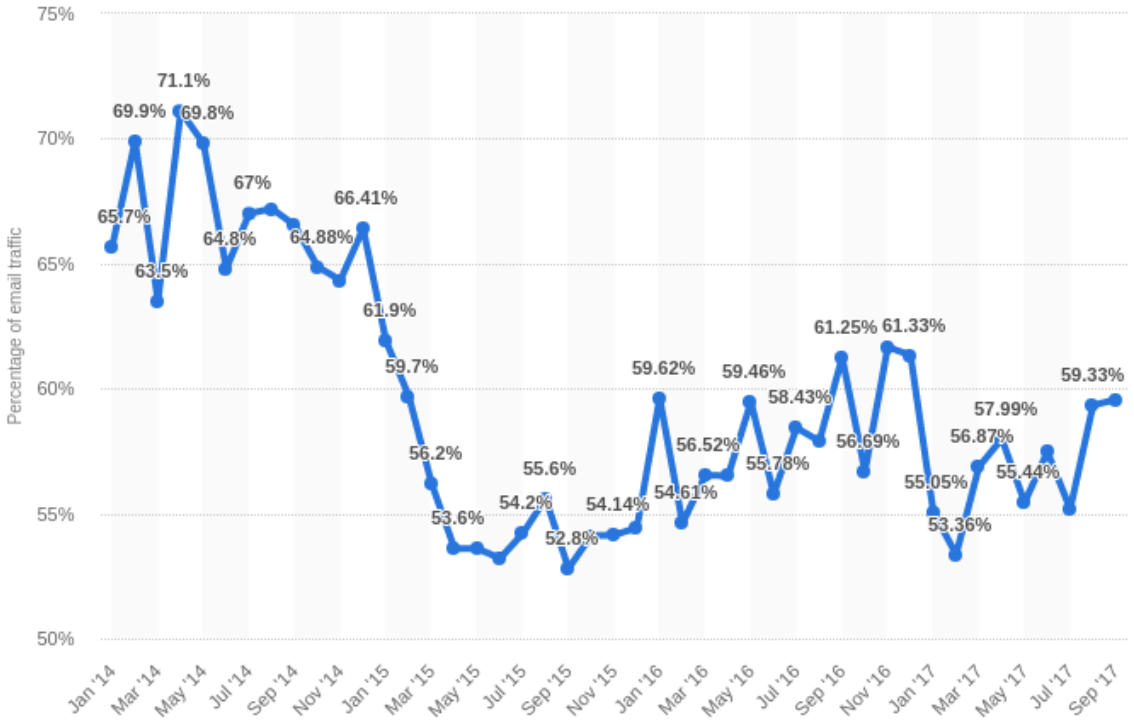


Figure 39: Spam trends

The above figure gives us some kind of idea about how to vary the value of $\alpha$ and $\beta$. Based on the dataset, 43.6% of the emails are spam. If we keep the supplied priors in this exact same ratio, we will still get the same precision and recall values. This can be seen by giving $\alpha$ as 43000 and $\beta$ as 56000. The answers for evaluation metrics are exactly the same as in the case of normal multinomial. If we give a very large value of $\alpha$, the recall for Spam articles will be very high, but the precision will be very low. For $\alpha = 1000000, \beta = 0$ the following are the scores.

| Precision | Recall | F-Score |
|---|---|---|
| 0.929 | 0.979 | 0.953 |

At the same time, increasing $\beta$ ensures that precision for spam class is very high, but recall is low. For $\alpha = 0, \beta = 1000000$ the following are the scores.

| Precision | Recall | F-Score |
|---|---|---|
| 0.968 | 0.954 | 0.961 |

It can be seen how the precision has increased but the recall has decreased. We now have control over both the parameters. The following two are the plots for the first and second cases. As can be seen, the area under the curve seems to be larger for the second curve. Increasing the value of $\beta$ seems to affecting the probabilities more.

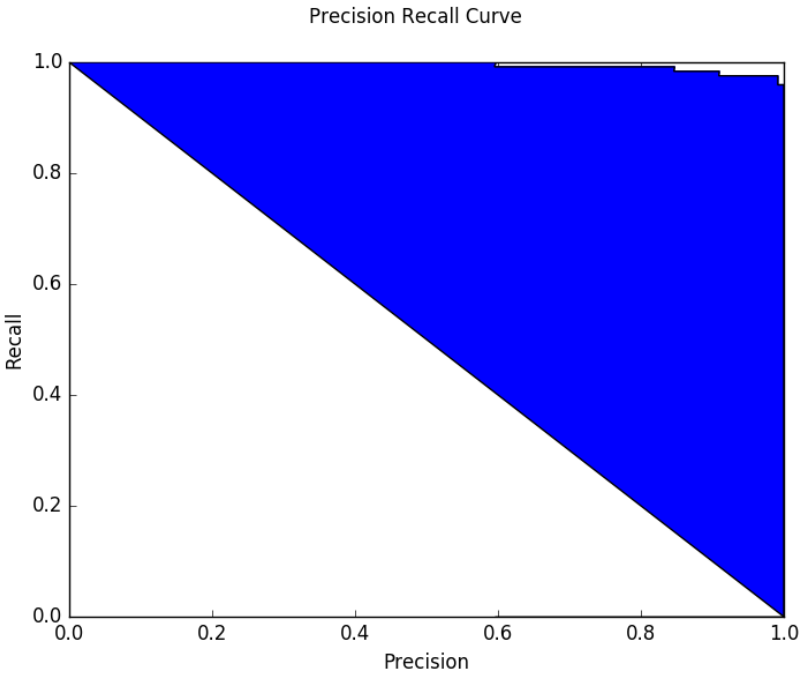


Figure 40: Beta Function with alpha varying

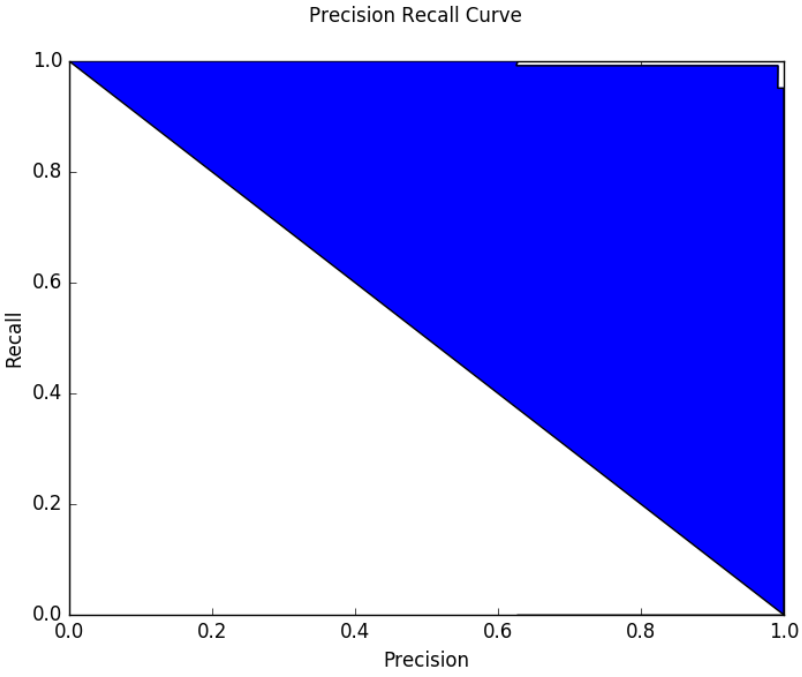


Figure 41: Beta Function with beta varying

We check how different beta's affect the F-Score of the classifier. The value of $\beta = 0$ will represent the normal multinomial case. The following graph shows how the F-Score varies with respect to $\beta$ on the logarithmic scale. $\beta = 10000$ gives the

best score for the classifier. Note that it reaches F-Score upto 0.965, by just changing the priors supplied.

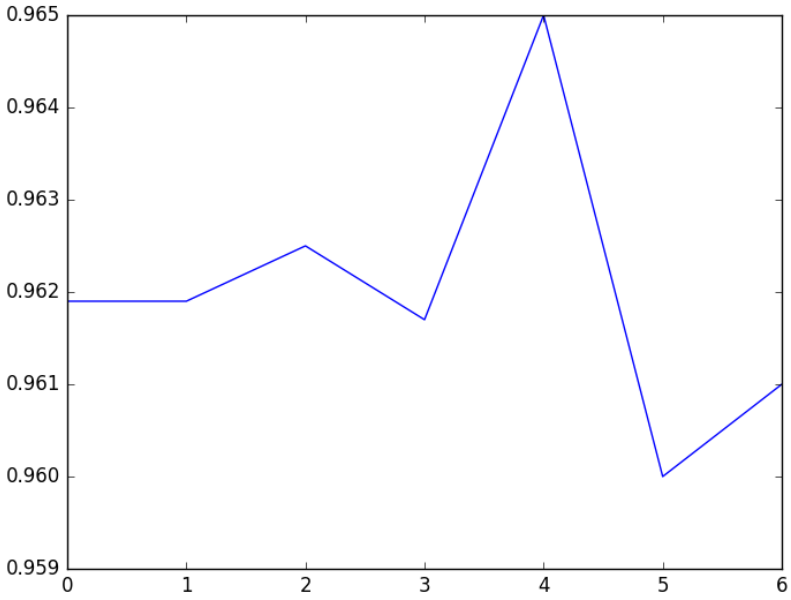


Figure 42: F-Score vs log(beta)