Principles of Machine Learning (CS4011)

Programming Assignment 2

Ameet Deshpande - (CS15B001)

October 10, 2017

1 Question 1 - Principal Component Analysis

- This question expects us to perform Principal Component Analysis on the dataset to get one Principal Component. The direction is used to project the dataset and Linear Regression is performed using an Indicator Variable. The Class Labels are 1 and 2. Thus, like in the previous assignment, the *threshold* for the Linear Regressor to decide which class an instance belongs to is $\frac{(1+2)}{2}$, which is 1.5.

- Care has been taken, not to use the Test Data to find the Principal Component. This is because, we need to follow the rule that no Test Data has to be seen before verification using the same. Using it to find the Principal Component will leak the test data. It would also be wrong to perform PCA once each on the Training and Test data. This is because of the variance in the data which will give different Principal Components for both the datasets. The procedure used is thus the following.

  - ⋆ Use the training data to find the Principal Component of the same.
  - ⋆ Project the training data on the Principal Component and learn a Linear Regressor for this 1 dimensional problem (excluding bias term).
  - ⋆ At prediction or testing time, use the same Principal Component to project the Test Data and use the learnt Linear Regressor to predict the value.
  - ⋆ Depending on the value of the indicator variable, (≥ 1.5 or < 1.5) predict the class.

- PCA uses the whole data to find the principal component. Unlike LDA, it does not take into account the class labels while finding the direction. It thus ends up finding the direction with maximum variance, disregarding completely the class labels. Below are $3D$ plots of the dataset which show how this property of the method gives low accuracies. *matplotlib* has been used to generate them.

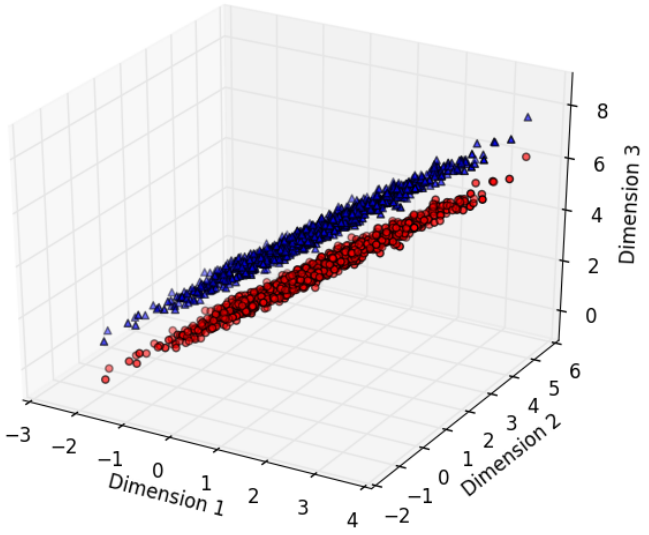


Figure 1: View 1

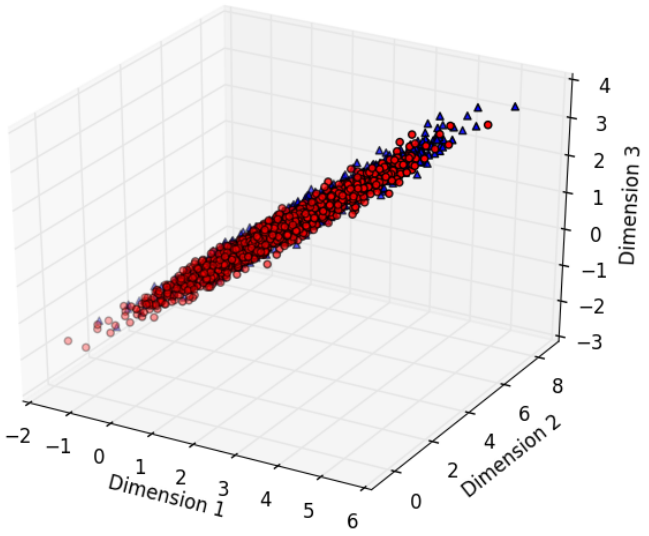


Figure 2: View 2

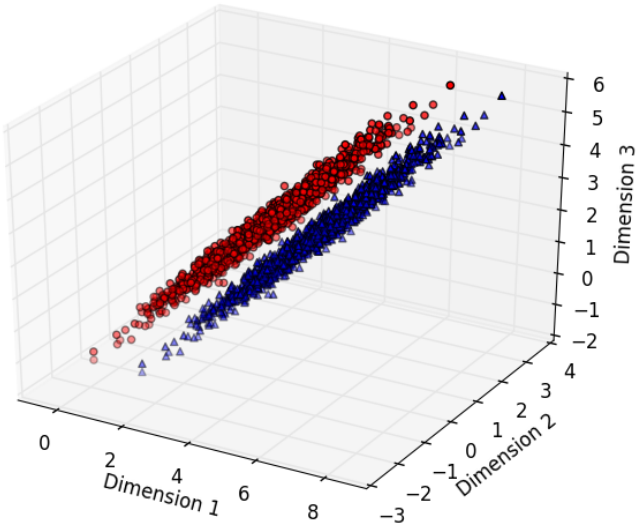


Figure 3: View 3

- As can be seen from the plots, the classes seem to have similar distributions and their class centers seem to be separated in one dimension. View 2 gives a very interesting plot. This is because, most of the data points lie along one line in it. Since PCA is expected to find such a line with High Variance, it will most likely find this as the Principal Component. The problem it will run into here is that, this direction has almost no predictive power as to which data point belongs to which class as, for every blue point, there is a red point with the same value across that direction and hence the classifier will not be able to differentiate between the same. This can be seen from the accuracies that the model gives. Accuracy of Linear (Logistic) Classifier which uses all 3 dimensions has been attached to show relative performance.

Model	Class	Precision	Recall	F-Score
PCA	Class 1	0.611	0.615	0.613
PCA	Class 2	0.613	0.61	0.611
Logistic	Class 1	1.000	1.000	1.000
Logistic	Class 2	1.000	1.000	1.000

- Given that the classes are balanced, a model that always predict class label as 1 will give precision, accuracy and F-scores of 0.5. The PCA model is thus clearly performing poorly, and the reason, as explained before is that PCA finds the direction of maximum variance disregarding the class labels completely. It so happens that on this dataset, both the classes have data samples along that

direction and thus predictive power is very less. We can thus understand from this that PCA will be a good dimensionality reduction technique for Regression tasks where there is no concept of labels, but classification tasks where label information has to be used before dimensionality reduction is where it fails.

- The following is the plot which shows the curve learnt by the linear model and the dataset is plotted as well. The plot clearly shows how the predictive power of the learnt dimension is very low. Each x coordinate has almost equal numbers of data points from the two classes. x axis represents the projected value and y axis represents the class label (1 and 2).

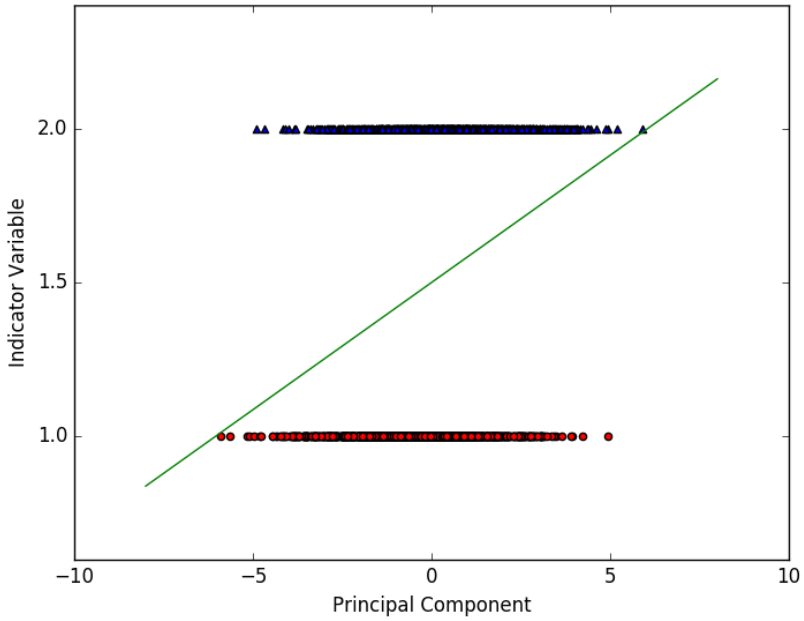


Figure 4: Projected Dataset

2 Question 2 - Linear Discriminant Analysis

- LDA performs much better than PCA in this case. Intuitively this can be seen from the 3-D Dataset plot itself. It looks like it is generated from Multivariate Gaussian Function. LDA's inherently assume that the classes conditioned densities are Multivariate Gaussian with the same covariance matrix. Thus, LDA is anyway expected to do very well. The $3D$ plots will be the same as in the previous question, so only two of them are repeated here.

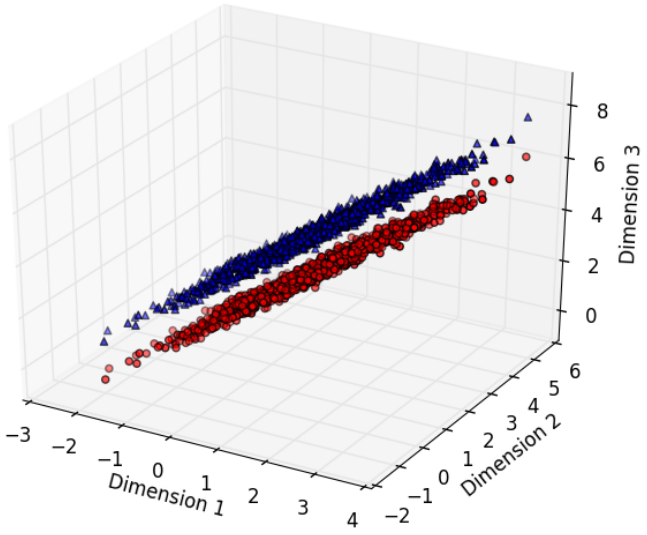


Figure 5: View 1

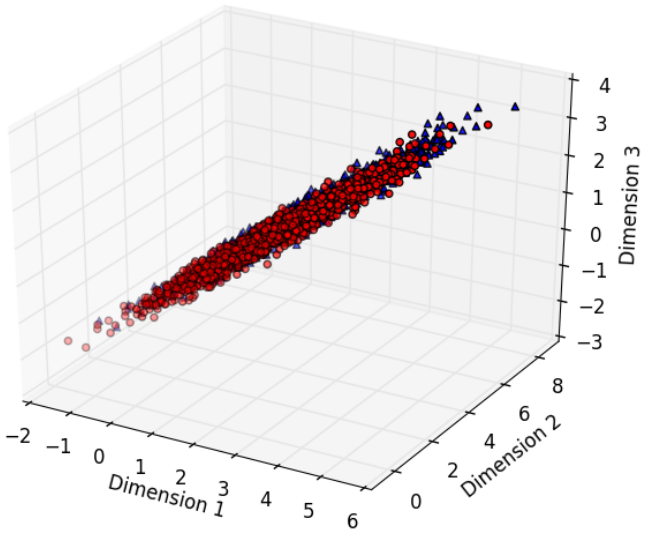


Figure 6: View 2

- The procedure followed to build a classifier is very similar to previous question. *sklearn's* LDA function actually has an inbuilt *predict* function that can be

used. But to make comparisons between PCA and LDA easier, the following steps are followed.

- ⋆ Learn 1 direction to project the data on, using only the Training Data
- ⋆ Learn a Linear Regression Model using this one feature and bias term.
- ⋆ At prediction time, use the same direction to project the test data.
- ⋆ Use the learnt model to classify test data points.

- The results obtained are tabulated below.

Model	Class	Precision	Recall	F-Score
LDA	Class 1	1.000	1.000	1.000
LDA	Class 2	1.000	1.000	1.000
PCA	Class 1	0.611	0.615	0.613
PCA	Class 2	0.613	0.61	0.611

- LDA clearly learns a different direction as compared to PCA. This can clearly be seen in the plot below which shows the projected dataset and the classifier boundary.

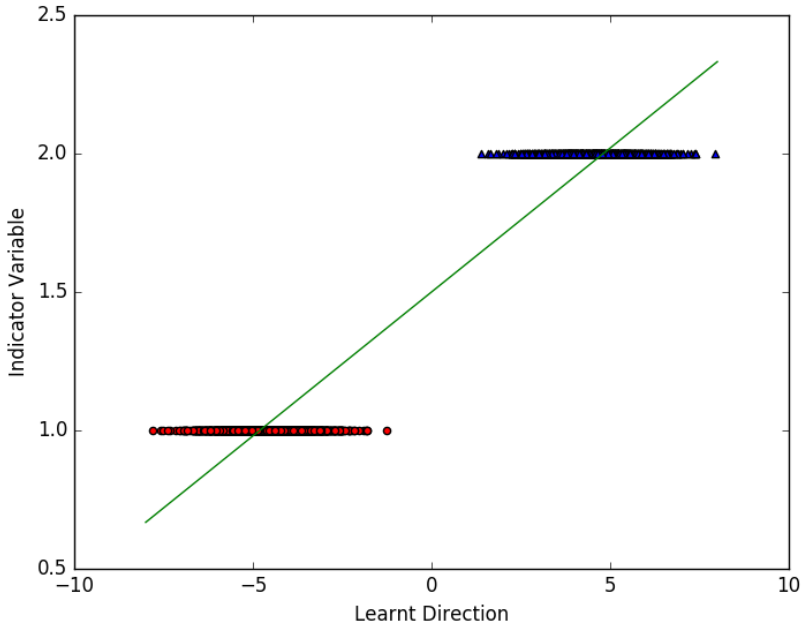


Figure 7: Projected Dataset

- As can be seen from above, LDA learns a direction which maximizes the distance between the centers of the two classes. It can thus achieve perfect classification

even on Test Data. From these two experiments it is apparent that LDA technique performs better than PCA. Both the techniques are used for Dimensionality Reduction, but LDA specifically uses the class labels to learn the direction. Very intuitively, the fact that LDA is using more information than PCA and that they have the same objective should mean that LDA perform better in some scenarios. The inferences that can be drawn from the experiments are highlighted below.

- ⋆ Dimensionality reduction can help save computation power. From Question 2, it is apparent that we were able to achieve perfect classification with just one feature. Though running a Logistic Regression model (Extra Experiment in Question 1) would have given the same results, using LDA reduced the number of featured to $\frac{1}{3}$. It is thus a good idea to perform a Dimensionality reduction to make the model more efficient.
- ⋆ Visualizing the data may give very meaningful insights. Of course it is not easy to plot data when the feature space has more than 3 dimensions, but crucial features can still be plotted to see which kind of dimenstionality reduction to use. In this case, it was visible that the data was generated from a gaussian-like distribution.
- ⋆ The main inference is that the choice of the Dimensionality Reduction technique has profound effects on the model. More points about the differences between the two techniques used are discussed below.

- **From the table it is apparent that LDA did much better than PCA.** Following are the reasons for the same.
  - ⋆ First step to understand it is that PCA and LDA can give completely different directions, which is actually happening in this case. The following is a case when the learn almost orthogonal directions. PCA learns the blue solid line and LDA learns the blue dashed line. (Figure borrowed from Quora).

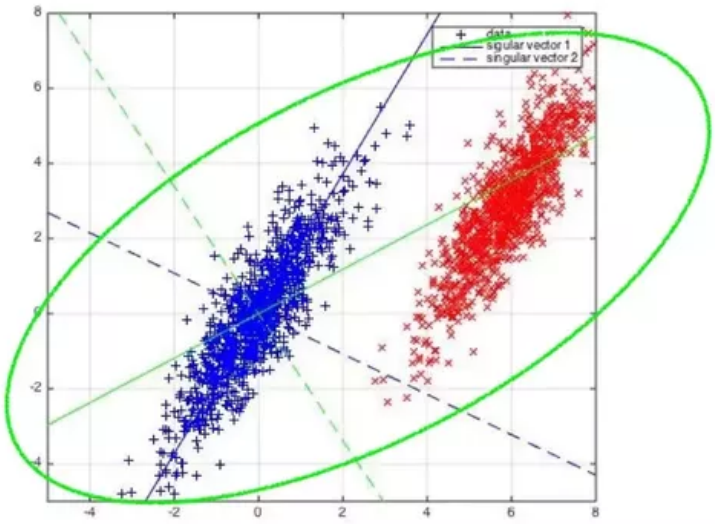
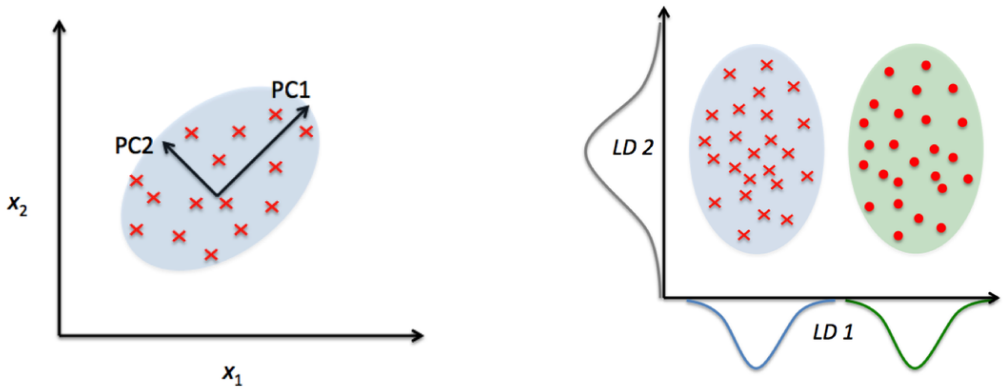


Figure 8: PCA vs LDA

⋆ The reason why this happens is that PCA and LDA find the directions in different ways. **PCA is label agnostic** as mentioned before. LDA however takes the labels into consideration. PCA is thus just a method which finds the direction of maximum variance of data samples. In Figure 2, it can be seen than most of the data lies along the same direction. PCA thus finds this direction and projects the data on it. However, this direction offers no predictive power as there are as many red points on that line, as there are blue, and there are both similarly spread. It's performance is thus very close to that of a model which guesses randomly. LDA on the other hand is a model which takes the distance between the class centers into account. Along the direction that PCA has learnt, the distance between class centers is almost 0 as the datasets are overlapping. LDA thus does not learn that direction. However, in a direction almost orthogonal to it like in Figure 5, the distance between the centers is larger. LDA thus finds this direction. **LDA thus not only tries to maximize the variance along the direction learnt, but it also tries to maximize the distance between the centers** and thus learns a more predictive feature. Below is a figure borrowed from a Blog of how PCA and LDA differ in the direction they choose. Note how direction $PC2$ is actually the second one picked by PCA, but the first one picked by LDA.



(a) PCA

(b) LDA

Figure 9: Directions learnt

3 Question 3 - LDA, QDA and RDA

- The Iris Dataset which is downloaded is loaded into a DataFrame, and only the mentioned two features are extracted. Following is a scatter plot of the data provided. *matplotlib* was used to generate the same. Only the training data has been used to plot the data. Also note that a test-train split is performed on the data. 66% of the data is used as training data and 33% is used as the test. That gives 100 points for training the classifiers.

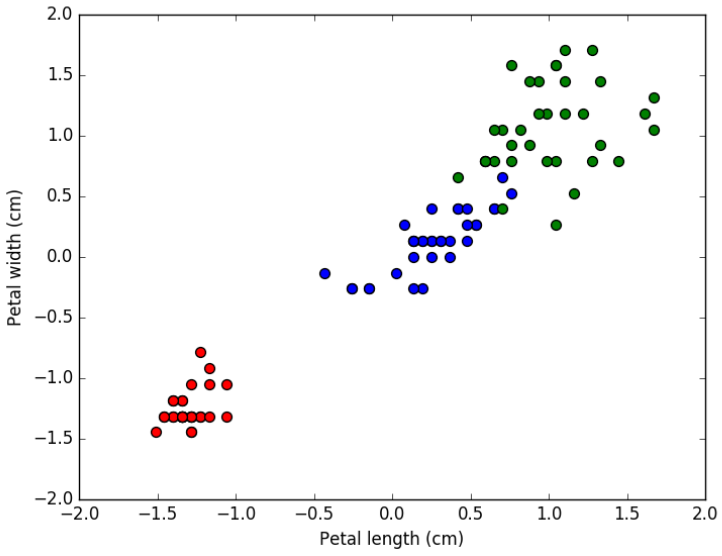


Figure 10: Iris Data Scatterplot

- As can be seen, there is one class which can be linearly separated from the other

two classes. However, there is some overlap present between the *blue* and the *green* classes, which represent Iris virginica and Iris versicolor. The red class represents Iris setosa. Thus the best classification boundary between the blue and green classes need not be linear. The following experiments will give more insights into the same.

- The method that is used is that, LDA or QDA is fit to the data. The function fits gaussian to the data. Since this is a discriminant based method, it is very natural to use One-vs-One classifiers. Also, given that there are only 3 classes both the one-vs-one scheme and the one-vs-rest scheme will give the same number of classifiers. We thus use the one-vs-one approach to draw classifier boundaries.

3.1 LDA

- LDA assumes the classes are fit with gaussians of the same covariance matrix. They thus end up giving linear boundary classifiers. The sklearn LDA function does a one-vs-rest fit and gives classifier boundaries for all the classes. These boundaries are visualized and test set predictions are made to check how good the model is. The discriminant function is as follows.

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + log(\pi_k)$$

- The following plot shows the classifier boundary for LDA. The colours indicate which class the data sample belongs to. We do not expect LDA to be able to differentiate between the blue and green classes, but it should do a good job in separating the red class. The following plot shows the same. Note that there however is an ambiguous region (shaded both blue and green) who's class can be decided by comparing the distance between it and the class means.

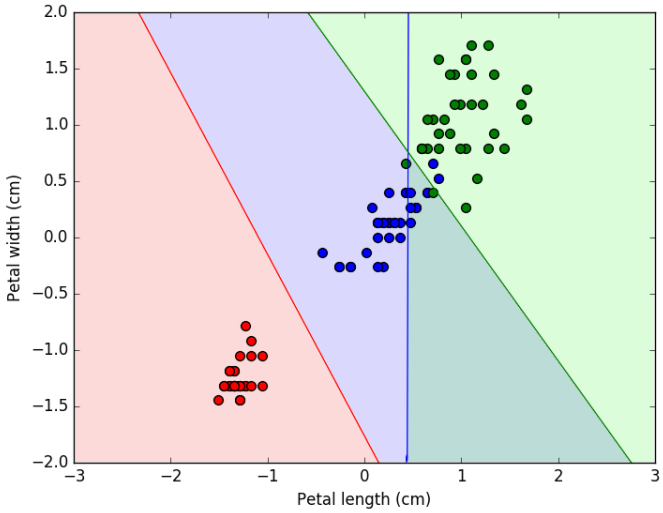


Figure 11: LDA Decision Boundaries

- The learnt LDA is tested on the 50 data samples. The following are the results. It can clearly be seen that the red class has very high precision and recall and the blue and green class where we expected some confusion because of overlap have slightly lesser precision and recall.

Class	Precision	Recall	F-Score
Class 1	1.000	1.000	1.000
Class 2	0.81	1.000	0.9
Class 3	1	0.733	0.846

- When 3 fold - cross validation was performed on the data, following were the F1 Scores that were obtained.

Class	F-Score
Class 1	0.980
Class 2	0.921
Class 3	0.979

3.2 QDA

- QDA does not make the assumption of equal covariance of all classes. The classifier boundary that it learns will thus have quadratic terms. The discriminant of the class is given by the following formula.

$$\delta_k(x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \log\pi_k$$

- QDA is expected to do better than LDA given that it can model more complex classifier boundaries. The same method followed in LDA's case is followed with QDA.

- As can be seen in the classifier boundaries in the plot, the curves look quadratic. But it looks like the train data has been overfit. This is because, for points near the blue quadratic, even though they are very close to the class mean, it is possible to classify them as green. The model is evaluated on test data to check its accuracy.

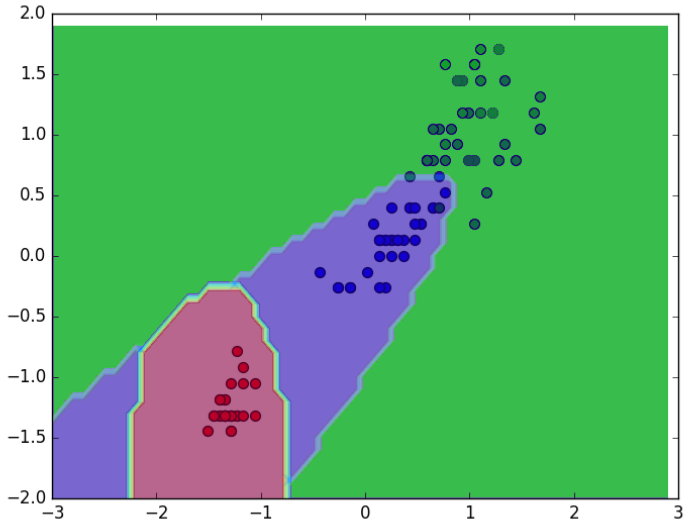


Figure 12: QDA Decision Boundaries

- The following are the evaluated metrics when a test train split was used

Class	Precision	Recall	F-Score
Class 1	1.000	1.000	1.000
Class 2	0.857	1.000	0.923
Class 3	1	0.800	0.888

- When 3 fold - cross validation was performed on the data, following were the F1 Scores that were obtained. As can be seen, QDA performs better than LDA as expected, given that the classes are not linearly separable. If the classes had turned out to be linearly separable, LDA would have been an optimal classifier.

Class	QDA (F1)	LDA (F1)
Class 1	1.000	0.980
Class 2	0.921	0.921
Class 3	1.000	0.979

3.3 RDA

- RDA is used to strike a compromise between LDA and QDA. The covariance matrices used for each class is given by the following formula.

$$\hat{\sum_k}(\alpha) = \alpha \times \hat{\sum_k} + (1-\alpha) \times \hat{\sum}$$

- As can be seen, RDA will limit to QDA if $\alpha = 1$ and will limit to LDA if $\alpha = 0$. α is a hyperparameter than can be learnt using a validation set. The discriminant function that RDA uses is the same as that of QDA.

- There is one more tweak that can be done to the model. λ can be used as a regularization parameter. It can be used as follows.

$$\hat{\sum}(\gamma) = \gamma \times \hat{\sum} + (1-\gamma)\hat{\sigma}^2 \times I$$

- There are thus two parameters α and γ as per the textbook. There is no direct RDA function in *sklearn*. I have thus borrowed code from QDA implemented in sklearn, tweaked the code and obtained results. The code gives an interface very similar to QDA and uses the discriminant function to decide which class it belongs to. I am citing scikit-learn for the code I have written (which we have been using throughout anyway). The function takes in two parameters α and γ which implement the above functionality.

- It is important to understand what the parameters are doing. As explained before, α is trying to strike a trade-off between the LDA and the QDA models. RDA can tend to both LDA and QDA at the extreme ends of the values of α it can take. The parameter γ can shrink the covariance matrix to the identity matrix. It basically controls how close the LDA matrix is, to the identity matrix. Note that α is the more important parameter and we are going to mainly focus on that.

- The following is a plot for $\alpha = 1.0$ and $\gamma = 1.0$. This should give a very similar answer to QDA, and the plot below shows the same.

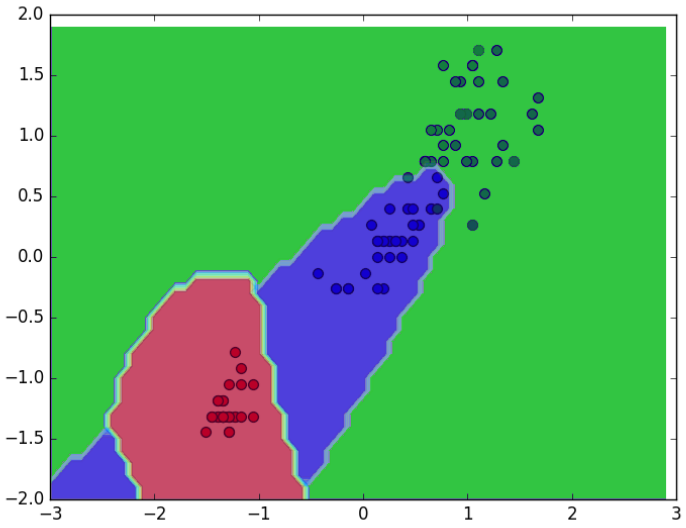


Figure 13: RDA simulating QDA

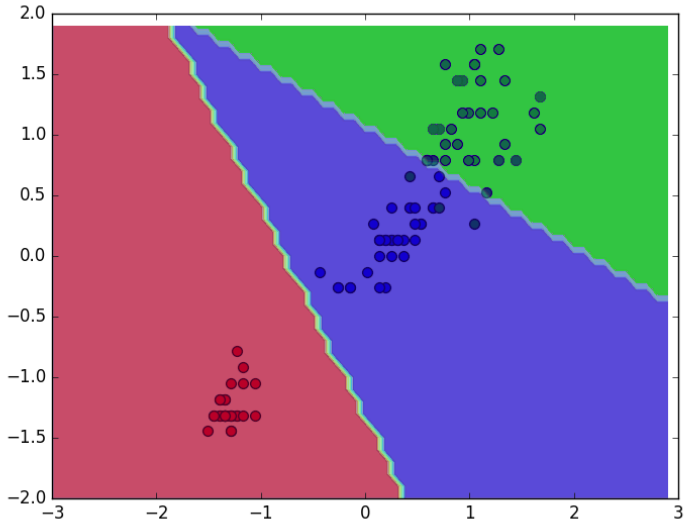


Figure 14: RDA simulating LDA

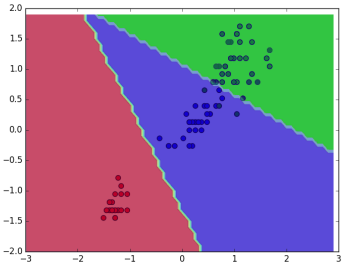
- Now that we know that RDA is a pretty flexible model, we vary both α and γ to find out the best parameters. Both γ and α are varied over their whole ranges. Note that there is unsufficient data for validation. However, the right procedure would be to perform a cross validation to get the best hyperparameters as is done in the next question. The best hyperparameters obtained are $\alpha = 1.0$ and $\gamma = 1.0$. This is actually the same answer as that of QDA. But from the plots we can see that, $\alpha = 0.7$ actually seems like the more generalized model. Note that when we change γ, it is always the case that F-score increases as γ increases. We thus stick to $\alpha = 0.6$.

- The following are the evaluated metrics.

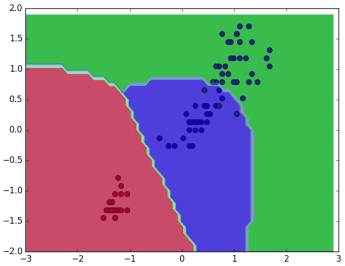
Class	Precision	Recall	F-Score
Class 1	1.000	1.000	1.000
Class 2	0.857	1.000	0.923
Class 3	1	0.800	0.888

- Comparing this with QDA we can see that the accuracy is almost the same and according to me, RDA will be able to generalize better given the fact that it does not take very shar turns as in the case of QDA. We could have probably tested this had we had more data. But this seems intuitively correct as well.

- The plots for 4 different values of α are shown side by side to see how RDA's decision boundary slowly changes from quadratic to linear when the variation is made.

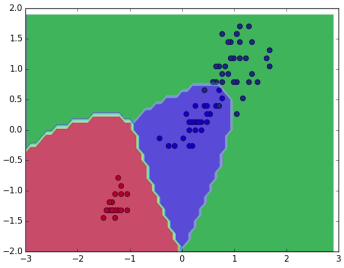


(a) alpha=0.0

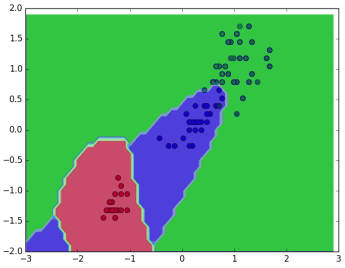


(b) alpha=0.3

Figure 15: RDA



(a) alpha=0.7



(b) alpha=1.0

Figure 16: RDA

4 Question 4 - Support Vector Machines

- Stratified Cross validation is used to create the folds. A function called GridSearchCV is used from sklearn. It iterates over a parameter grid and gives the parameters that do the best on cross-validation.
- We choose to do 5 fold cross validation as the training set size is only 1000.
- coef0 is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial. This is used in the polynomial kernel.

4.1 Linear Kernel

- A Linear Kernel will give a linear classification boundary. SVMs can be extended to multi-class by either using $one - vs - one$ scheme or $one - vs - rest$ scheme. By default the sklearn Support Vector Machine functions use $one - vs - rest$ as the scheme. We stick with this as the complexity of this is lesser and we also have lesser number of decision boundaries to learn (4 in this case and 6 in $one - vs - one$ case).

- The linear kernel gives a very simple model. The only hyperparameter to tune is thus C. Higher values of C force the classifier to be a hard margin classifier. It is also observed that for C values of 10 and above, the model takes considerably more amount of time to train. We thus follow a different method to find the best C parameter. A graph is plotted to check what values of C allows the model to perform well. The range of values used here is large. After finding the approximate range to work with, $k - fold$ cross validation is used to tune the hyperparameters. The training set size is about 1000. We thus use 5 fold cross validation rather than 10 fold, as a crucial parameter such as C should be tuned on dataset size of 200 rather than 100. The following is the graph used to narrow the range. Using the information from the graph, the range of values of C to look at is narrowed down to $0.1 - 2$, as there is a peak which lies in that region.

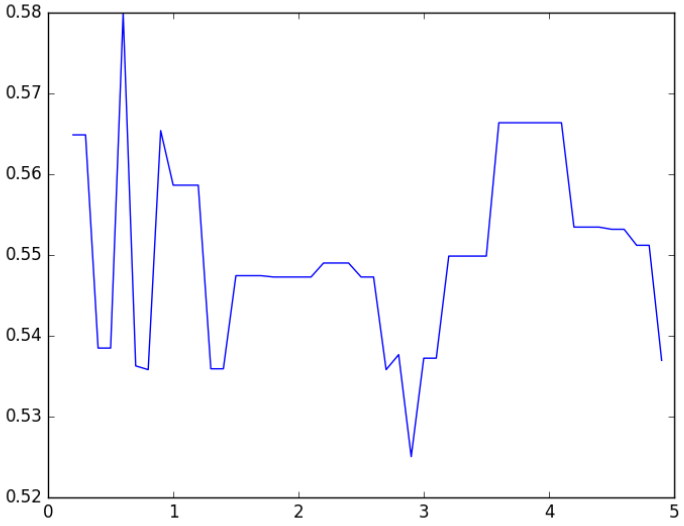


Figure 17: Average F1 Score v/s C

- We now use $5 - fold$ cross validation. Only the training data is used for the cross validation as we stick to the rule that none of the test data should be seen before testing time. This gives us around 200 for validation. The hyperparameters found out are as following.

C
0.4

- The evaluation metrics are tabulated below.

Class	Precision	Recall	F-Score
Class 1	0.44	0.6	0.51
Class 2	0.8	0.6	0.685
Class 3	0.48	0.6	0.53
Class 4	0.538	0.35	0.424

4.2 Polynomial Kernel

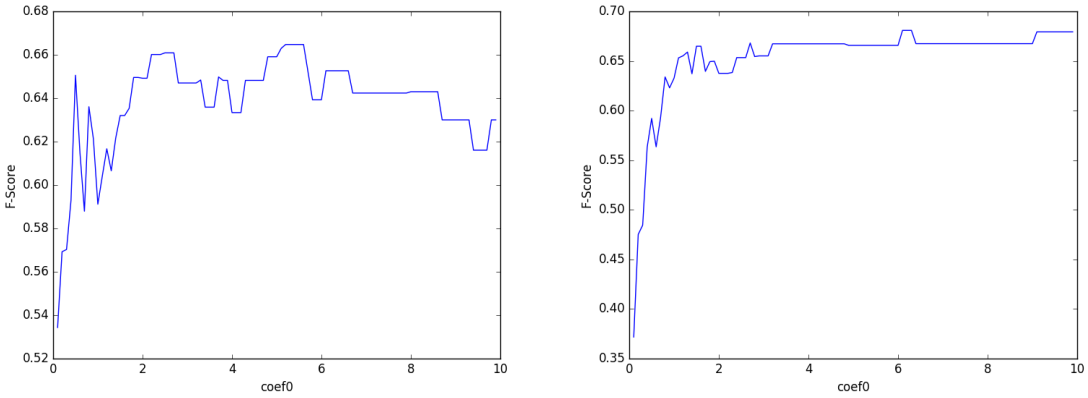
- Polynomial Kernel gives more complex models and thus has more hyperparameters to tune. The hyperparameters are *gamma*, C, *degree* and $coef0$. A polynomial kernel is given by $(gamma * u' * v + coef0)^{degree}$. This definition is borrowed from LIBSVM's official page. gamma and coef0 control the trade off between lower-order polynomial terms and higher order polynomial terms, which is apparent from the expansion. After expansion, it can also be seen that in polynomial kernel, *gamma* and C are serving the same purposes, trade-off between higher order and lower order terms. We thus vary one of the parameter widely and the other parameter comparatively lesser. The degree of the polynomial has to be chosen carefully so as to not overfit. A degree of 2 or 3 is usually considered okay to not overfit. Degrees $1, 2, \ldots, 5$ are considered. Care is taken to see if there is overfitting.

- Considering all the parameters together increases the parameter search space exponentially. For example, if there are 10 parameters and all of them can take just 10 different values, the parameter search space becomes, 10^{10}. To avoid this exponential blow up in our case, like in previous methods, the range of values that can be taken by different parameters are reduced by varying it individually on default parameters, and then a Grid Search is done.

- Tuning $coef0$. Two separate experiments are run on degrees 3 and 5. It is concluded that coef0 values between 2 to 6 can be used as a narrower range. An interesting observation is that, in in case of degree 5, F-score is improving as the coef0 value increases. Higher the coef0 value, lower the weightage given to higher order terms. Thus, the model basically favors lower order terms.

- *gamma* is by default set to $\frac{1}{num\_of\_features}$. We vary *gamma* between 0 and 1 even though coef0 is supposed to take care of the parameter.

- Higher degree kernels will tend to overfit. Also, an RBF kernel can give the same performance, if not better then higher degree polynomial kernels as it use infinite basis expansion. The following is the graph plotted for degrees between 1 to 20. As can be seen, increasing the degree is actually decreasing the accuracy. We thus restrict ourselves to degrees between 1 to 7, where the highest peak is. The simpler the model, the better.



(a) For Degree 3 (b) For Degree 5

Figure 18

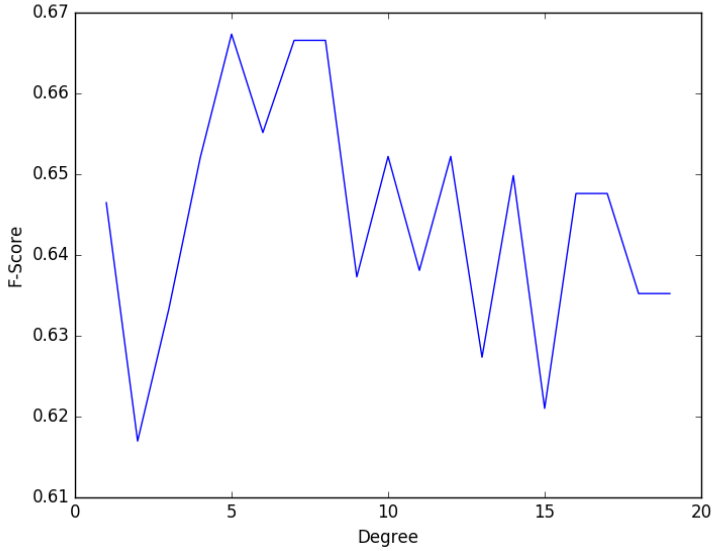


Figure 19: Average F1 Score v/s Degree

- Now that we have found smaller ranges to work with, we do a GridSearchCV to find the best parameters. The following are the values of the parameters found. Degrees 2 and 3 give very similar accuracy scores. Even if degree 3 model was performing slightly better, it would be a good idea to choose the simpler model as it does not overfit. When evaluating on the test dataset, the degree 3 model actually does slightly better. But choosing a model based on the test set would be the same as leaking the test data. So we stick with the degree 2 model as the best bet. For information, the $F1-macro$ scores for degree 2 model and degree 3 model are 0.6225 and 0.648. A good way to actually do cross validation is to use **nested cross-validation**. But since the dataset given has explicit train

and test splits, we do not use the same.

C	degree	coef0	gamma
0.2	2	3.0	0.1

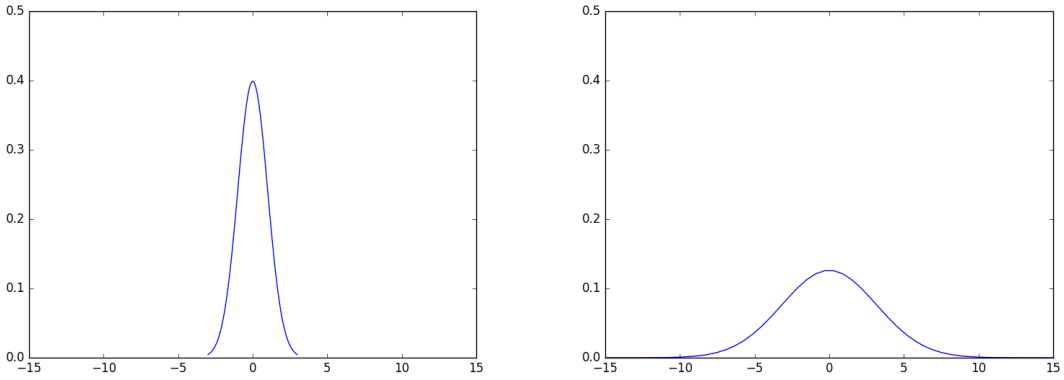
- The evaluated metrics are as follows.

Class	Precision	Recall	F-Score
Class 1	0.5	0.45	0.47
Class 2	0.78	0.9	0.83
Class 3	0.739	0.85	0.79
Class 4	0.5	0.4	0.44

4.3 Gaussian Kernel

- The Gaussian or the RBF kernel is a widely accepted default kernel for SVMs. The kernel can project the input features into an infinite feature space. This gives very high flexibilty to the model. The model also gives very smooth boundaries. As always, the flexibility comes at the cost of overfitting.

- The definition of a Gaussian kernel is $e^{-\gamma \times |u-v|^2}$. There are thus 2 parameters to tune, C and γ. C represents the same thing being used so far. γ controls the variance of the gaussian that is used. A small γ means a gaussian with large variance. Thus, say v is a support vector, it will have effects even on points u which are far away from it. Larger γ will imply the vice versa. Below are plots for large value of γ and small value of γ respectively (Figure 16).



(a) Large Gamma (b) Small Gamma

Figure 20

- In Linear and Polynomial Kernel cases, increasing the value of C used to increase the training time considerably as it is hard to find a hard margin classifier when

the data is not linearly separable. But given the flexibility of the model, RBF kernel can deal with hard margin cases. The range of values of C considered should thus be much larger. Below is a graph which plots $F-Score$ vs C values. We thus consider C values between 15 and 40.

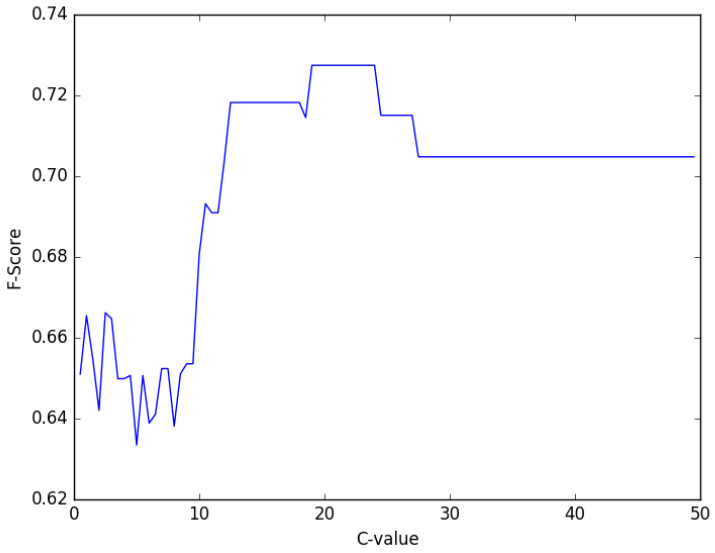


Figure 21: Average F1 Score v/s C

- A similar curve is plotted for γ. C value of 20 is chosen. From the plot, we use range of 0.01 to 0.1 for GridSearchCV. Smaller values seem to be doing better.

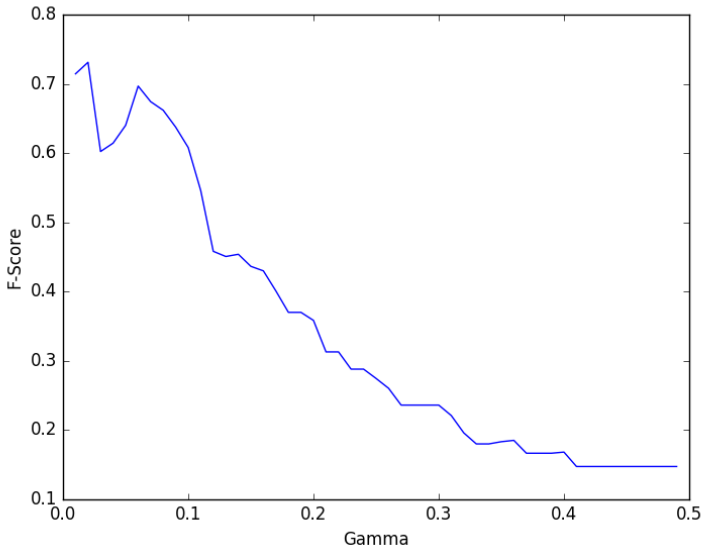


Figure 22: Average F1 Score v/s gamma

- After performing GridSearchCV, the following are the parameters obtained.

C	gamma
20	0.02

- The evaluated metrics are the following.

Class	Precision	Recall	F-Score
Class 1	0.611	0.550	0.570
Class 2	0.869	1.000	0.930
Class 3	0.800	0.800	0.800
Class 4	0.631	0.600	0.615

4.4 Sigmoid Kernel

- The Sigmoid Kernel (as defined on the official libsvm page) is given by $tanh(\gamma \times u' \times v + coef0)$. There are thus 4 parameters to tune, C, γ and $coef0$. Unlike in the case of Polynomial Kernel, γ and $coef0$ have different effects on the kernel, they will thus have to be separately tuned. The same methodology used in previous parts is used here to narrow down the ranges of the parameters.

- As can be seen from the graph below, the model seems to favor smaller values of γ. A gamma value of 0.01 does the best.

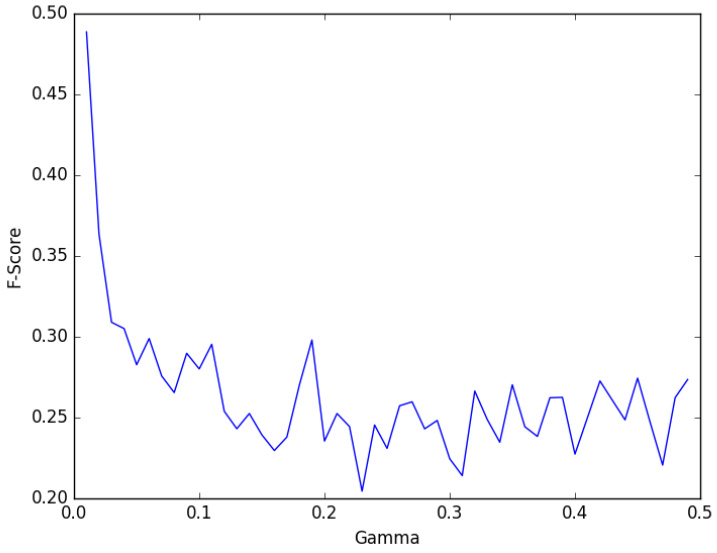


Figure 23: Average F1 Score v/s gamma

- The $coef0$ parameter is tuned similarly. Values between 0.1 to 100 are analyzed. The dependency with respect to coef0 is not very easy to see. But the values

γ and $coef0$ are correlated to an extent. This is because, small values of γ will need small values of $coef0$, or else the constant term will start dominating. Given that we will be using a small value of γ, we settle for $coef0$ values between 0 to 5.

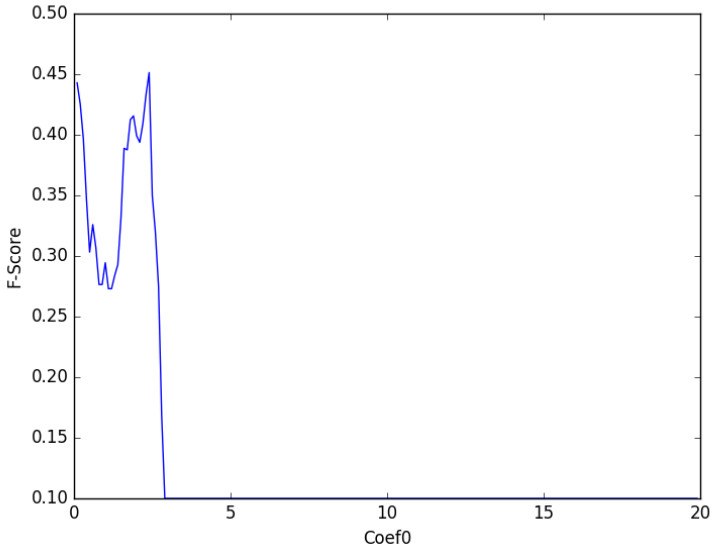


Figure 24: Average F1 Score v/s Coef0

- C generally is the most important parameter. We use default γ and $coef0$ values to tune the same. A softer margin classifier seems to do better than a hard margin classifier. We perform a GridSearchCV using these values.

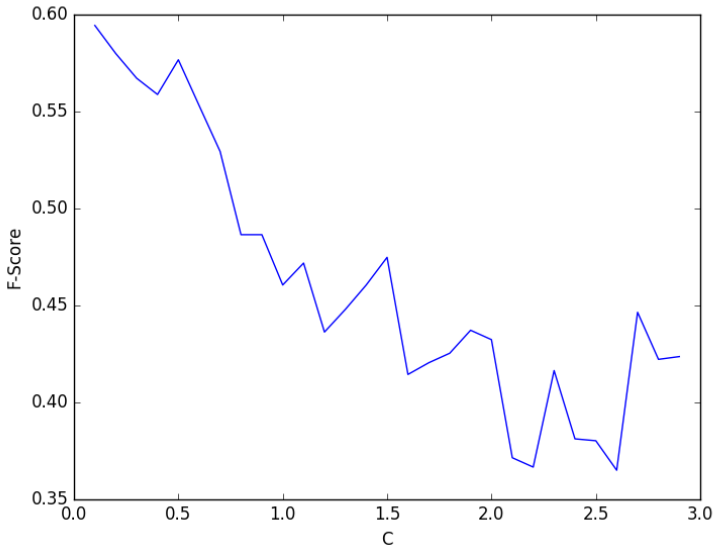


Figure 25: Average F1 Score v/s C

- The best parameters are tabulated.

C	gamma	coef0
0.4	0.01	0.0

- The evaluated metrics on the test data are tabulated below.

Class	Precision	Recall	F-Score
Class 1	0.727	0.400	0.516
Class 2	0.667	0.800	0.727
Class 3	0.529	0.450	0.486
Class 4	0.428	0.600	0.500

4.5 Results

- From the above experiments, the following macro F1 scores are obtained for all the models.

Kernel	Macro-F1
Linear	0.53
Polynomial	0.63
Gaussian	0.72
Sigmoid	0.55

- As can be seen, the Gaussian Kernel performs the best out of all the kernels followed by polynomial kernel. The Sigmoid Kernel does only slightly better than the Linear Kernel and is thus unsatisfactory in this case. It also takes a much longer time to train than the linear kernel or the Gaussian Kernel. This is as expected. The Gaussian Kernel does an infinite Basis expansion and is this able to model complex decision boundaries. Linear Classifier suffers here as the Classes clearly do not have linear boundaries as inferred in PA1b, as neural networks would have gotten high accuracy in that case. Polynomial Kernel does almost as well as neural networks thank to its flexibility of fitting higher degree terms.

5 Question 5 - Decision Trees

- The data from the website is downloaded as a csv file. The last 1124 instances are used as test data. That leaves 7000 data samples for training. This is a two class classification problem. Weka's $CSVLoader$ function is used from the commandline to convert the $.csv$ files to $.arff$ files which is the format expected by weka classifiers.

- A few points about the dataset are that, in the problem we are told to consider the last 1124 examples as test instances. This is not a very good way to choose the test data as most of the missing values are now concentrated in test data. But nevertheless, we stick with the same as the question necessitates us to do so. Another point is that, a few nominal values are missing in the test dataset but are present in the train dataset. There are some attributes for which all the possible values are not taken in the test dataset. But this shouldn't cause too many problems now, but to start using such a model in production, it is probably better to perform a Cross Validation.

- As is inferred from the below trees, *Stalk-root* does not appear in the tree anywhere. The missing values thus do not have a large effect.

5.1 Running the Decision Tree with default parameters

- The decision tree is first run with default parameters offered by $J48$ so that it can be used as a baseline model. The unpruned tree is used.

Pruning	MinNumObj	Reduced Error Pruning
No	2	No

- The default parameters themselves build a very good model. Decision trees are usually prone to overfitting. The learnt tree however does very well on the test data as well. The following are metrics for the baseline model on test data.

Class	Precision	Recall	F-Score
Edible	1	1	1
Poisonous	1	1	1

- The tree has 25 leaves and its total size is 31. It is thus impossible to beat the baseline model now as it has achieved the maximum possible accuracy. However, it is important always to get the best possible with the least possible, meaning, we aim for the best possible accuracy, precision and recall values, with the least complex model. $GraphViz$ is used to generate the graph for easy visualization. The $-g$ flag can be used to generate the $GraphViz$ code. The numbers in bracket represent the total number of data samples from training set which belong to that leaf. If there is any misclassification, it is indicated next to it.

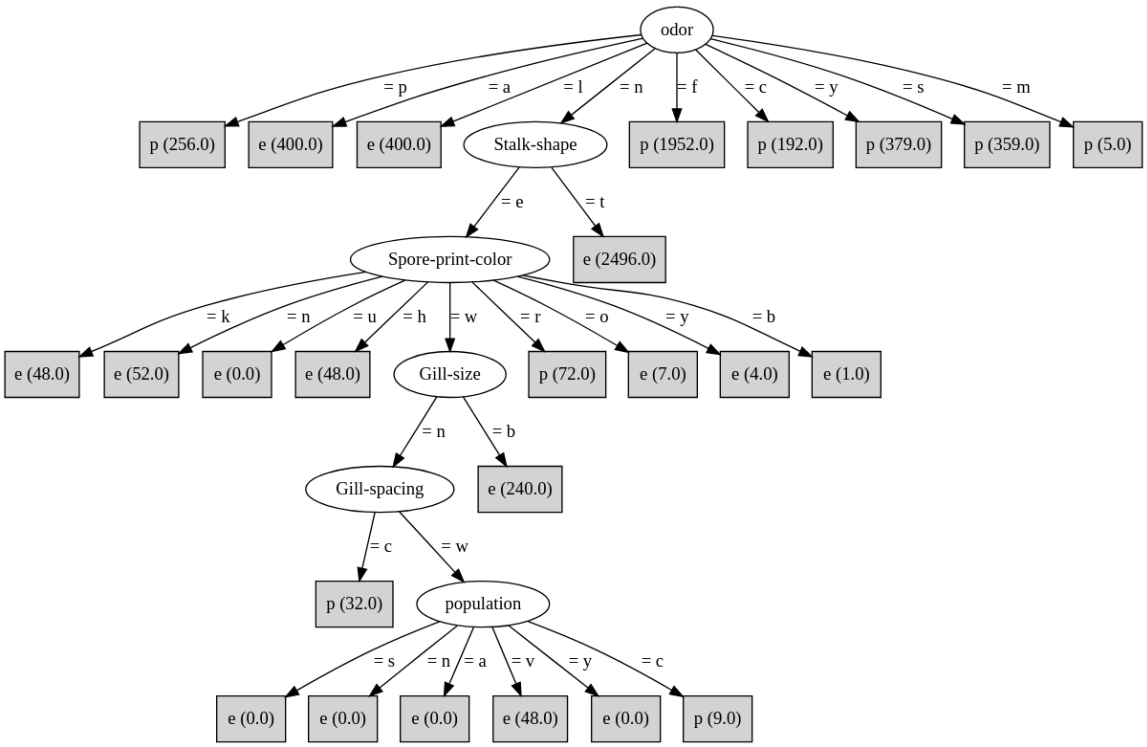


Figure 26: Baseline Model

5.2 Tinkering with the MinNumObj parameter

- This parameter can be changed using the $-M$ flag. This represents the minimum number of instances that need to be present in a leaf of the decision tree. MinNumObj can be considered as a parameter which can reduce overfitting. Having a different leaf for each instance of the training data will definitely give 100% accuracy on training data, but it will give very bad accuracy on test data. As can be seen in the figure above, there are leaves present with 1 or 4 leaves. Weka does something different from what we think it does. Weka ensure that atleast 2 of the branches going out of a node have more than $MinNumObj$ instances. This is done because, say there are 10 splits at a node. If $MinNumObj = 5$, this will have to ensure that there are 50 points at the node. This might be too harsh. Thus weka weakens this a little as can be seen from the leaf which has just one node.

- The following plot shows the dependency of number of leaves of the tree with respect to $MinNumObj$. The number of leaves are naturally expected to decrease as this parameter is increased. But sometimes the leaves may also increase as, when Weka splits on a node, it creates branches for all possibilities of the nominal variable for that feature. Thus, lots of branches are created when such a feature is encountered. But this discrepency is uncommon (but occurs with -M 10). The plot was generated after getting logfiles after changing the $MinNumObj$ parameter. A script was written to extract the values from

Weka's output. The overall trend is that the number of leaves reduce as we increase the parameter.

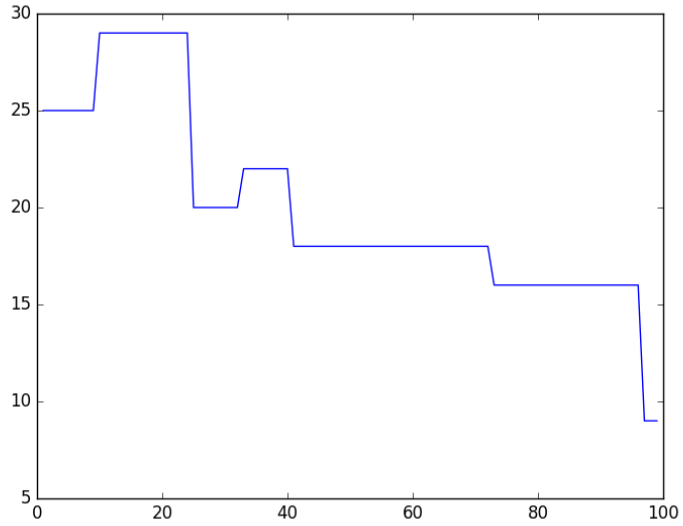


Figure 27: Number of leaves vs MinNumObj

- We now aim to see the dependency of the accuracy measure with respect to the parameter. We obviously expect the accuracy to go down as we have already achieved the best possible. But the reason we are doing it is to achieve a model of lesser complexity. The following plot shows the dependency. There is an interesting sudden drop in F score when $MinNumObj = 25$. There is probably a leaf which is of size 25 which is being wiped out because of the constraints which we have imposed.

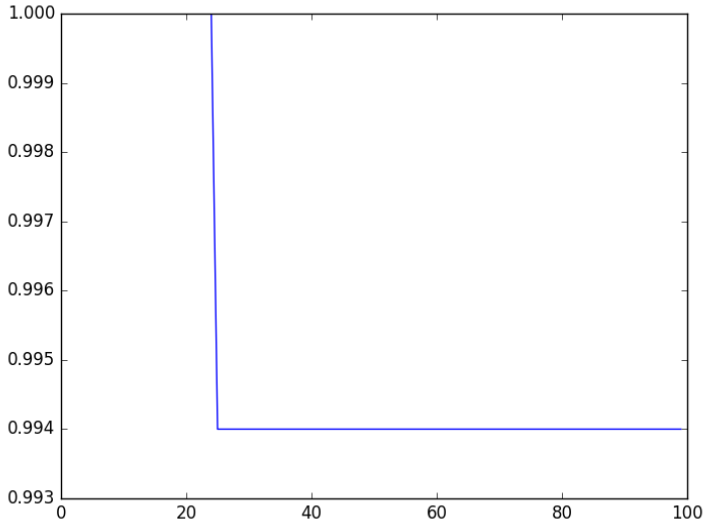
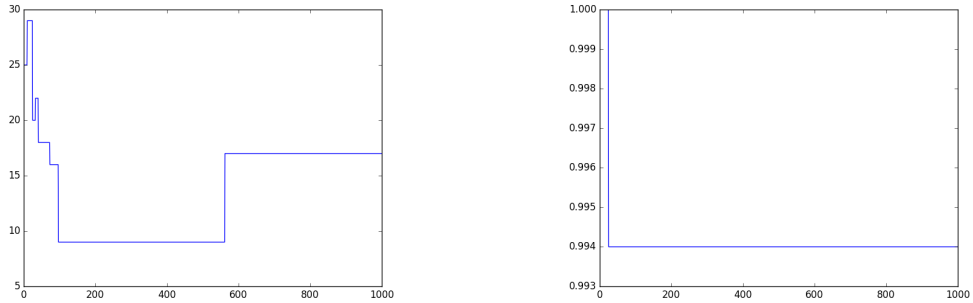


Figure 28: F-Score vs MinNumObj



(a) Number of leaves vs MinNumObj (b) F-Score vs MinNumObj

Figure 29: Extended Plots

- Important point to infer from the above plots is that the model is still doing very well even with less number of leaves. Using **Minimum Descriptor length** as the error metric will surely give a low complexity model as the better model because of the fact that it gives an accuracy of 0.994, which is very close to 1. The above are two more experiments in which the range over which parameter was run is much larger (Figure 29). The following is the tree obtained with $MinNumObj$ as 400. This value is chosen after seeing the graph. It gives only 9 leaves, but an F-Score of 0.994. As can be seen, *odor* is the most deciding feature when it comes to classification. More comments on this in a later subsection.

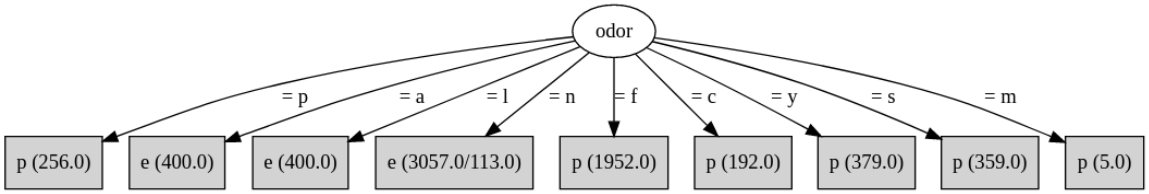


Figure 30: MinNumObj=400

5.3 Reduced Error Pruning

- Reduced error pruning is a very simple and fast way of pruning the tree. Though it is considered a naive way of doing the same, its efficiency and simplicity make it a very popular method.
- Reduced error pruning uses a validation set to make sure there are no coincidences or errors. A node is only removes if the tree which is obtained after the procedure does no worse or better on the validation set as compared to the initial tree. Use of all, training set, test set and validation set is recommened if there is enough data. We have about 7000 data samples in training set, which gives us the freedom to use a validation set of sufficient size.
- There are criticisms of REC that it is a very aggressive strategy and thus ends up over pruning the tree.
- The $-R$ flag can be passed in commandline to do Reduced error pruning on the tree. Weka implements this be taking another number ($-N\#$) which represents the number of folds. It uses one of the folds as validation set. Since 10 folds are suggested for cross-validation and we have enough (7000) data samples, using the same reasoning, we choose the number of folds to be 10. Like in the previous case, there are two parts of the tree that we want to keep a tab of, the number of leaves, and the F-Score.
- The following are the evaluated metrics after using reduced error pruning.

Class	Precision	Recall	F-Score
Edible	0.985	1.000	0.993
Poisonous	1.000	0.989	0.995

- The tree is drawn below for easier visualization. It is to be noted that this tree has only 21 leaves and its size is 27, as compared to the 25 leaves and 31 size baseline model. Reduced error Pruning has thus reduced the size of the tree at the cost of a small error. The error it is making is almost negligible.

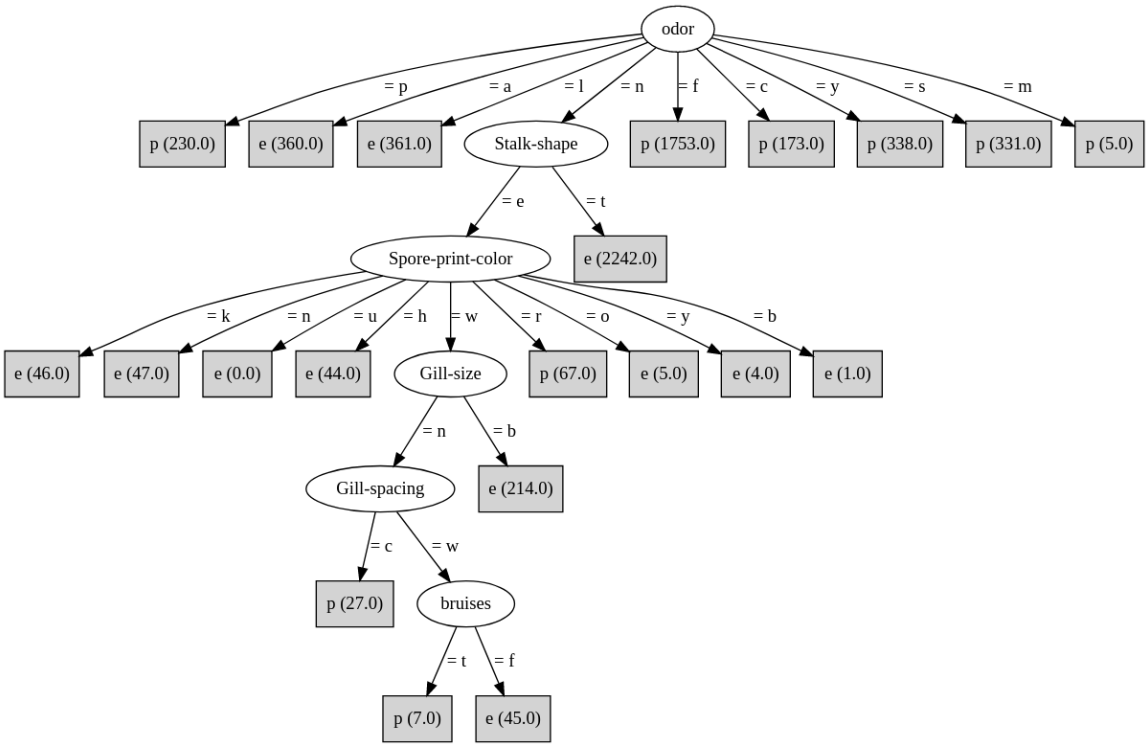


Figure 31: Reduced Error Pruning

5.4 Important features

- The important features can be decided in two ways. The first way is to see what features are being used to split the tree a lower heights. Consider Figure 26 which shows the baseline model. *odor* seems like the most important feature given that it is being used to split first. Only 5 features are being used in the tree. This means that the rest of the 17 features are unimportant. Considering the order in which the features are being used to split the nodes of the trees, the important features are *odor, Stalk-Shape, Spore-Print-Color, Gill-Size, Gill-spacing and population*. Another reason why it is easy to see that odor is the most important features is that even after using several techniques to reduce the size of the tree, the feature stays. Consider Figure 30 for example, which gives a very small tree. The tree has an F-Score of 0.994. All it does is predict the class based on the feature odor, by disregarding all the other 21 features. This very clearly shows how the important features will be at the lower levels of the tree. The same important features appear again in Figure 31 above.

5.5 Best Decision Tree

- The best decision tree depends on the application for which it is being used. We have already got a decision tree which gives an F-Score of 1. That will in some sense be the best decision tree. But at the same time, we might also want to reduce the complexity of the model. This gives a hint of using Minimum

Description length as the error metric. This will ensure that we have the least complex model with the best possible error.

- Another important point in this specific data-set is the weight we want to give to precision and recall of different classes. Note that this classifier might be used in a restaurant where the mushroom will be used if it not poisonous. Since poisonous mushrooms may end up causing very harmful affects, it is important to have the recall as 1, considering the poisonous mushroom as positive class. This will ensure that every poisonous mushroom is identified. In this case since all values are almost 1, there is no problem.

- The model is saved using the $-d$ flag and stored in a file called $J48.model$. It can be loaded using the $-l$ flag.

- Note that the F-measure of all models is between 1.000 and 0.994. 0.994 implies that there are $7000 - 0.994 \times 7000$ wrongly classified examples, which amounts to about 42 wrongly classified samples. Note that the simplest tree that we found had only 8 leaves, and the most complex tree which gave F-score as 1 had 25 leaves. We thus have two options now.

Model	F-score	Leaves	Wrongly Classified	Approx MDL
Complex	1.000	25	0	13.28
Simple	0.994	9	42	18.31

- We will have to transmit values of all the 22 features of the wrongly classified data samples which will result in $42 \times 6 = 252$ values. The tree models can be described very easily, with $log_2 22$ bits for telling which feature is being used to split, and $2 \times log_2 size\_of\_tree$ for describing the tree. We just need to describe the edges. This coupled with the fact that we do not want to wrongly classify poisonous mushrooms as edible, we choose the higher complexity model as the best model.