Programming Assignment 2

EE15B025 : Ganga Meghanath
October 10, 2017

FEATURE EXTRACTION

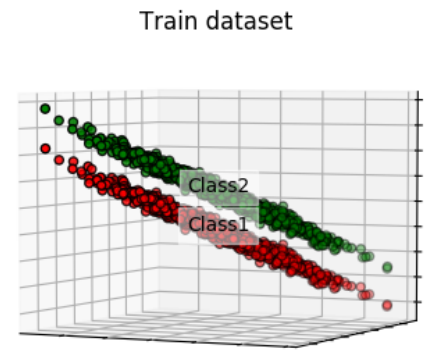
0.1 PRINCIPAL COMPONENT ANALYSIS : (PCA)

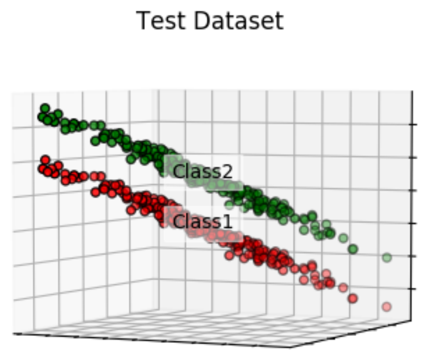
PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system.It can be thought of as fitting an n-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. If some axis of the ellipsoid is small, then the variance along that axis is also small, and by omitting that axis and its corresponding principal component from our representation of the dataset, we lose only a small amount of information.

A 3-dimensional dataset (DS3) has been provided. It contains two classes. Principal Component Analysis (PCA) is performed on this dataset and a single feature is extracted. The data in this projected space is used to train Linear Regression with indicator random variables. The learnt model has been used for classifying the test instances.

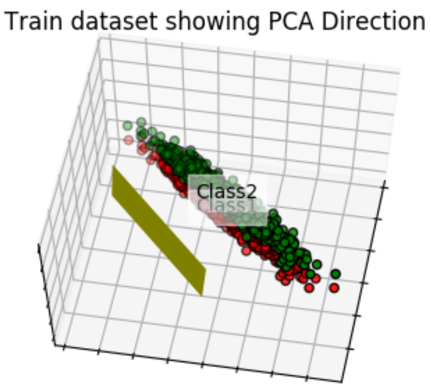
The results obtained are shown below :

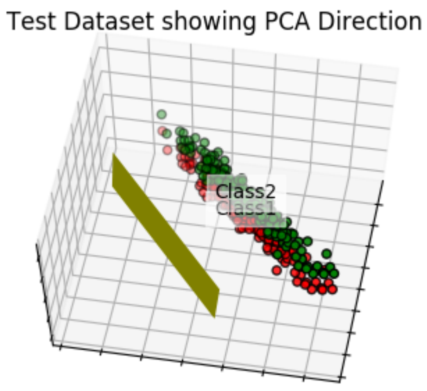
Labels	**Per Class Estimates**		
	Precision	**Recall**	**F-measure**
Class 1	0.611940298507	0.615	0.613466334165
Class 2	0.613065326633	0.61	0.611528822055
Overall Accuracy	61.25 %		



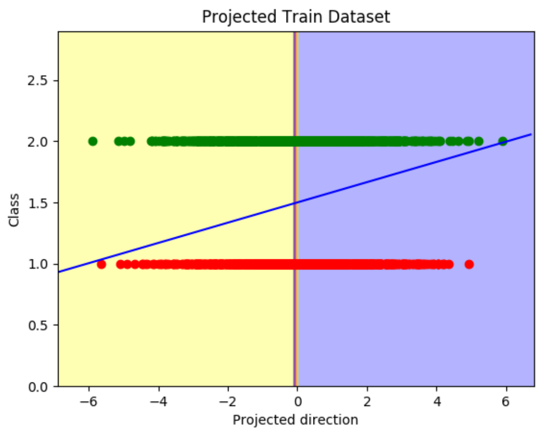


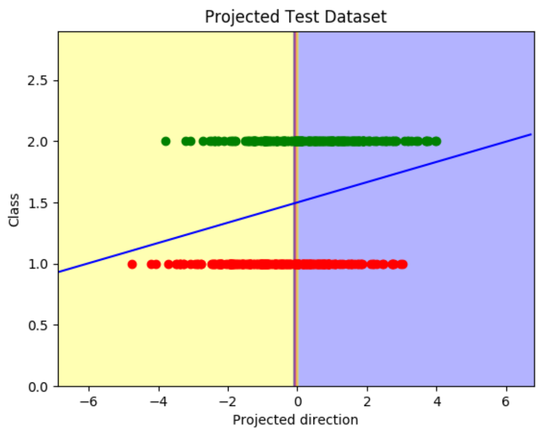
The 3D plot of the Dataset





The 3D plot of the Dataset with PCA Direction





The dataset depicted on the projected space along with the classifier boundary (y-axis:class)

The brown line indicates the decision boundary and the blue line indicates the predictions obtained from linear regression along the x-axis.

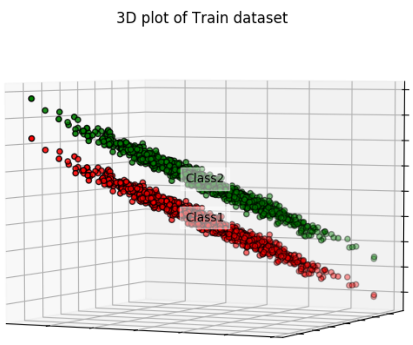
0.2 Linear Discriminant Analysis : (LDA)

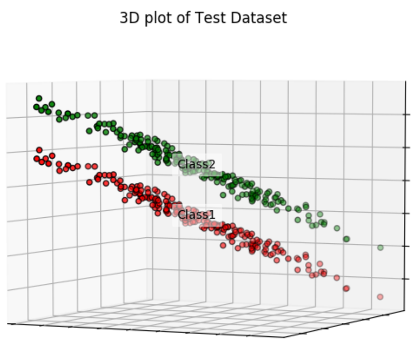
Linear Discriminant Analysis is most commonly used as dimensionality reduction technique in the pre-processing step for machine learning applications. The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid overfitting and also reduce computational costs.

A 3-dimensional dataset (DS3) has been provided. It contains two classes. Linear Discriminant Analysis (LDA) is performed on this dataset and a single feature is extracted. The data in this projected space is used to train Linear Regression with indicator random variables. The learnt model has been used for classifying the test instances.

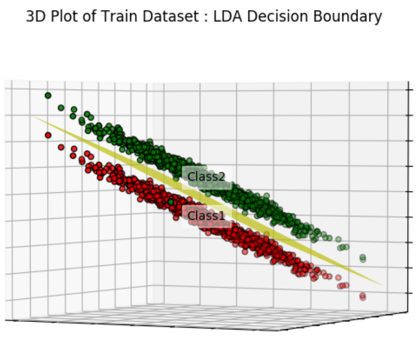
The results obtained are shown below :

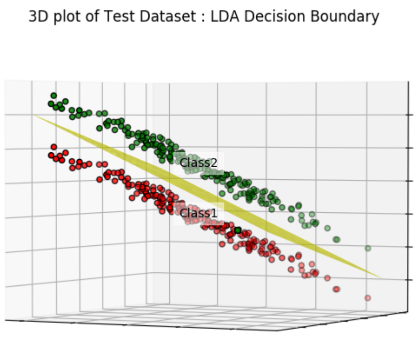
Labels	**Per Class Estimates**		
	Precision	**Recall**	**F-measure**
Class 1	1.0	1.0	1.0
Class 2	1.0	1.0	1.0
Overall Accuracy	100 %		



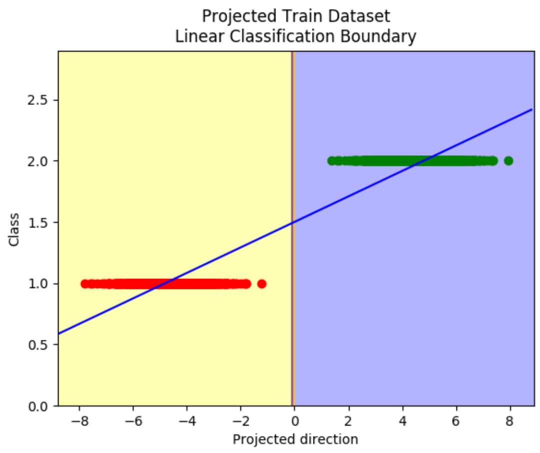


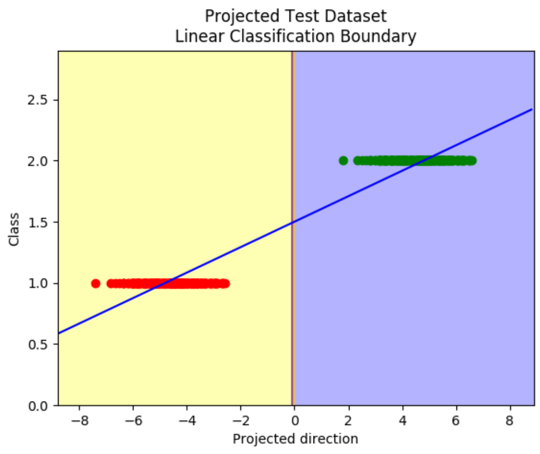
The 3D plot of the Dataset





The 3D plot of the Dataset with PCA Direction





The dataset depicted on the projected space along with the classifier boundary (y-axis:class)

The brown line indicates the decision boundary and the blue line indicates the predictions obtained from linear regression along the x-axis.

Comparison between PCA and LDA

As we can observe, LDA gives better results on the same dataset as compared to PCA when it comes to classification. This is because, in the case of a single derived feature, PCA attempts to find the direction in which overall variance of the dataset is maximum and the dataset is projected onto this direction, whereas in the case of LDA, it tries to find a direction that maximises the ratio of 'between class variance' and 'within class variance'. Hence, LDA tries to maximise the variance (or distance) between the two classes, while minimising the variance within a class, which is exactly what we want for classification purposes. Essentially, LDA explicitly attempts to model the difference between the classes of data, whereas PCA on the other hand does not take into account any difference in class. As a result, LDA gives better features for classification as compared to PCA.

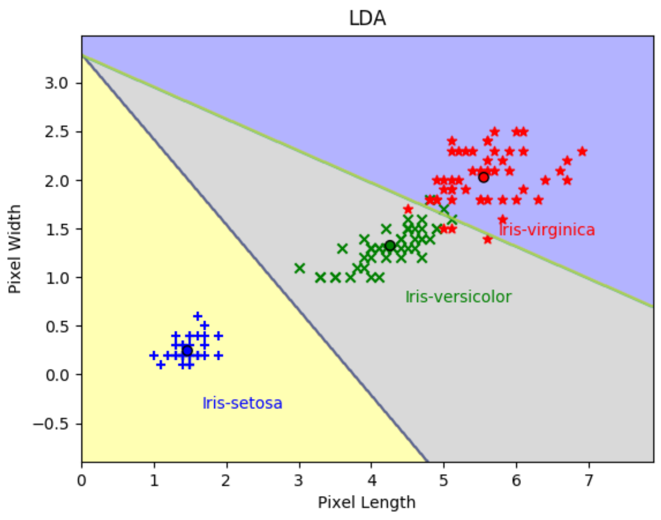
0.3 LDA and its variants : QDA and RDA

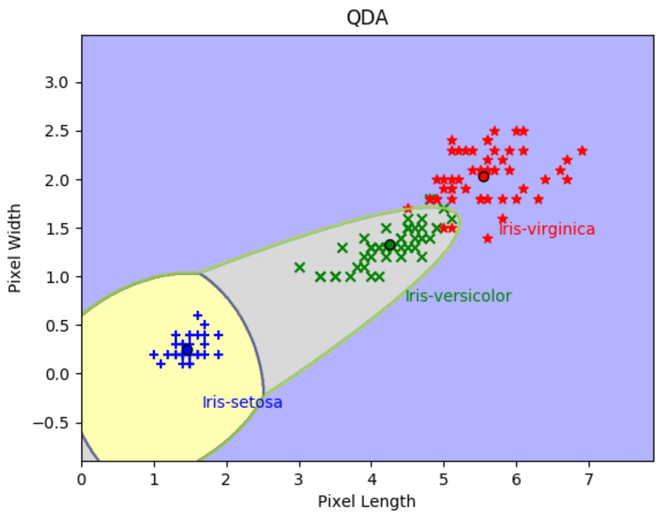
Quadratic discriminant analysis (QDA) is closely related to linear discriminant analysis (LDA), where it is assumed that the measurements from each class are normally distributed. Unlike LDA however, in QDA there is no assumption that the covariance of each of the classes is identical. Hence, the discriminant function is a quadratic function and will contain second order terms.
Regularized discriminant analysis is sort of a trade-off between LDA and QDA. RDA shrinks the separate covariances of QDA toward a common covariance as in LDA. The parameter α (reg_param in the QDA function in sklearn) controls the complexity of the model.In an extreme case, if α = 1, then the second term is zero and you get what you would with QDA. If α = 0, then you get what you would with LDA.

$$\hat{\Sigma}_k(\alpha) = \alpha\hat{\Sigma}_k + (1-\alpha)\Sigma_k$$

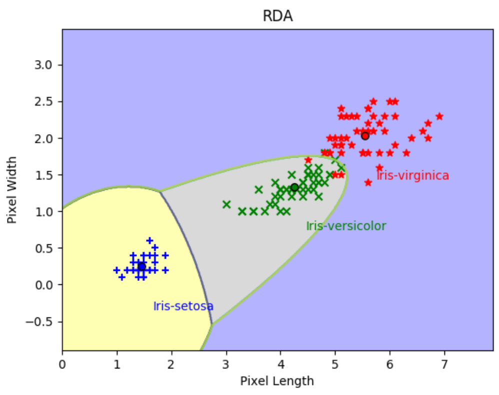
$\hat{\Sigma}_k(\alpha)$ is a convex combination of the common covariance matrix in LDA and a separate covariance matrix estimated as in QDA. (Note : sklearn regularizes the covariance estimate as (1-reg_param)*Sigma + reg_param*np.eye(n_features))
The decision boundaries learned by LDA, QDA and RDA have been visualised for Iris Dataset from http://archive.ics.uci.edu/ml/datasets/Iris/. (Only petal width and petal length features have been utilised as attributes)

The boundaries learnt after performing LDA and QDA are as follows :

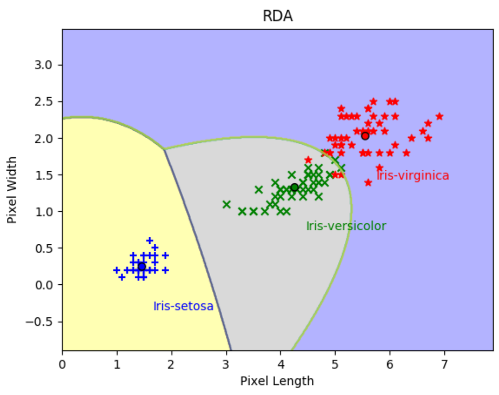




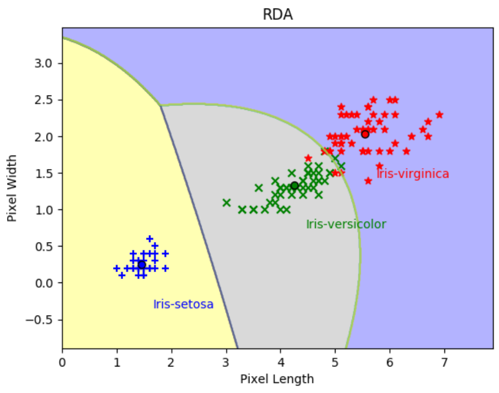
The boundaries learnt after performing RDA is as follows :



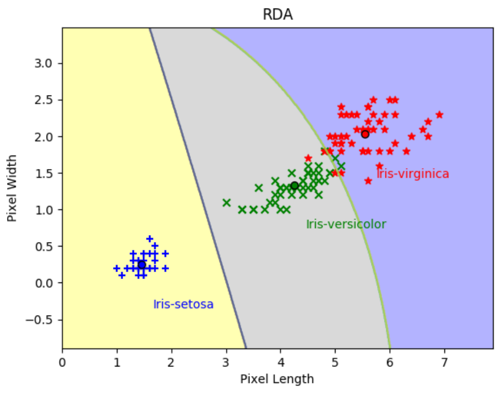
(i) $\alpha = 0.01$



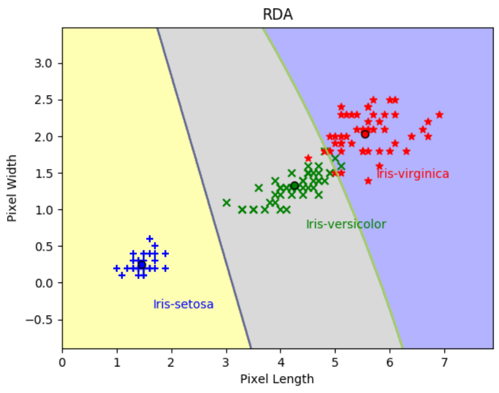
(j) $\alpha = 0.05$



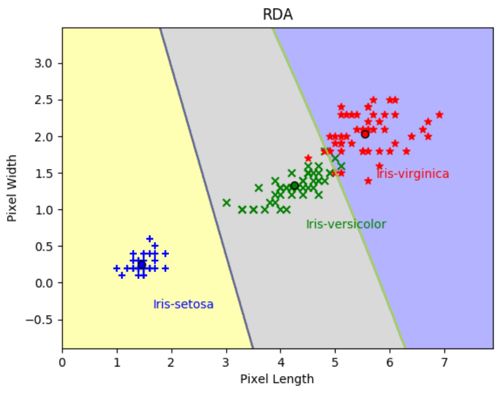
(k) $\alpha = 0.1$



(l) $\alpha = 0.25$



(m) $\alpha = 0.5$



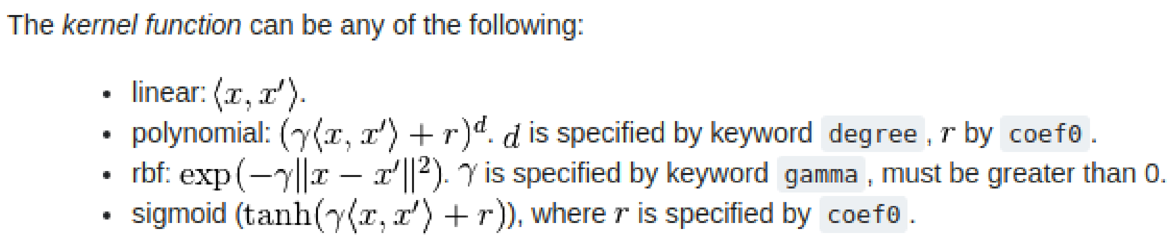
(n) $\alpha = 0.75$

SVM

Support Vector Machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.(Classification has been demonstrated below).
The dataset DS2 has been utilised for the classification. The objective is to train an SVM to classify the test images into either of the following four categories:

coast
forest
inside-city
mountain

The python sklearn package has functions that help implement libsvm with the concerned kernels. "data_creation.py" converts the datasets to libsvm format as train and test.



The *kernel function* can be any of the following:

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma\langle x, x' \rangle + r)^d$. d is specified by keyword `degree`, r by `coef0`.
- rbf: $\exp(-\gamma\|x - x'\|^2)$. γ is specified by keyword `gamma`, must be greater than 0.
- sigmoid $(\tanh(\gamma\langle x, x' \rangle + r))$, where r is specified by `coef0`.

Linear Kernel

The model has been selected after analysing the average accuracy obtained on doing a cross-validation using 10 different batches. A Stratified K-Folds cross-validator has been used for the purpose of creating the batches. Different models were tested by varying the value of the penality 'C'. The best model obtained has been saved in 'svm_model1.model' in the 'Models' folder. The best 3 results obtained are listed below :

C	**Accuracy**
0.1	0.65
4.71423423423	0.6
5.27263157895	0.6

The results for the best model is as follows :

Labels	Per Class Estimates		
	Precision	**Recall**	**F-measure**
coast	0.59	0.50	0.54
forest	0.68	0.85	0.76
inside-city	0.60	0.60	0.60
mountain	0.72	0.65	0.68

POLYNOMIAL KERNEL

The model has been selected after analysing the average accuracy obtained on doing a cross-validation using 10 different batches. A Stratified K-Folds cross-validator has been used for the purpose of creating the batches. Different models were tested by varying the value of the penality 'C', degree of the polynomial 'd', Kernel coefficient 'γ' and the independent term 'coef0'. The best model obtained has been saved in 'svm_model2.model' in the 'Models' folder. The best 3 results obtained are listed below :

Degree	C	γ	Coef0	Accuracy
2	0.0164285714286	8.0	2.14285714286	0.65306122449
2	3.8125	0.5125	0.0	0.591836734694
2	0.1	0.1	3.0	0.510204081633

The results for the best model is as follows :

Labels	Per Class Estimates		
	Precision	**Recall**	**F-measure**
coast	0.64	0.35	0.45
forest	0.80	0.80	0.80
inside-city	0.55	0.80	0.65
mountain	0.45	0.45	0.45

GAUSSEAN KERNEL

The model has been selected after analysing the average accuracy obtained on doing a cross-validation using 10 different batches. A Stratified K-Folds cross-validator has been used for the purpose of creating the batches. Different models were tested by varying the value of the penality 'C' and the Kernel coefficient 'γ'. The best model obtained has been saved in 'svm_model3.model' in the 'Models' folder. The best result obtained using the chosen model is listed below :

C	γ	Accuracy
94.7373684211	0.01	0.6625
3.31081081081	0.501351351351	0.6375
10.0	0.21387755102	0.6375

The results for the best model is as follows :

Labels	Per Class Estimates		
	Precision	Recall	F-measure
coast	0.56	0.45	0.50
forest	0.84	0.80	0.82
inside-city	0.64	0.90	0.75
mountain	0.59	0.50	0.54

Sigmoid Kernel

The model has been selected after analysing the average accuracy obtained on doing a cross-validation using 10 different batches. A Stratified K-Folds cross-validator has been used for the purpose of creating the batches. Different models were tested by varying the value of the penality 'C', Kernel coefficient 'γ' and the independent term 'coef0'. The best model obtained has been saved in 'svm_model4.model' in the 'Models' folder. The best result obtained using the chosen model is listed below :

C	γ	Coef0	Accuracy
500.0	0.0111020408163	0.0	0.755102040816
212.040816327	0.021387755102	0.0	0.734693877551
73.6868421053	0.01	0.01	0.612244897959

The results for the best model is as follows :

Labels	Per Class Estimates		
	Precision	Recall	F-measure
coast	0.59	0.50	0.54
forest	0.71	0.60	0.65
inside-city	0.59	0.80	0.68
mountain	0.58	0.55	0.56

Hence, the best accuracy was achieved using a sigmoidal kernel in the various trials conducted using the given dataset and different c-SVM parameters.

Decision Trees

The entire problem has been done using Weka. Mushroom dataset (2-class problem with 8124 instances) from UCI machine learning repository https://archive.ics.uci.edu/ml/datasets/Mushroom has been used. The last 1124 instances are used as test data and the rest as training data.

The python file 'Dataset_creation.py' reads the dataset, segregates it as training and testing data with 7000 and 1124 instances each and converts them into the format ARFF and creates the training and testing files 'mushroom_train.arff' and 'mushroom_test.arff'.

The models obtained by the decision trees have been saved in the folder 'Models'. The format used is 'cross_valid_<MinNumObj>_rp.model' and 'cross_valid_<MinNumObj>.model'. 'rp' signifies that Reduced Error Pruning has been done on the tree.

The trees obtained using the models have been saved in the folder 'Trees'. The format used is 'cross_valid_<MinNumObj>_rp.png' and 'cross_valid_<MinNumObj>.png'. 'rp' signifies that Reduced Error Pruning has been done on the tree.

The results obtained using the models have been saved in the folder 'Results'. The format used is 'cross_valid_<MinNumObj>_rp' and 'cross_valid_<MinNumObj>'. 'rp' signifies that Reduced Error Pruning has been done on the tree.

MinNumObj refers to the minimum number of datapoints in each leaf. We can observe from the table that, as the MinNumObj value decreases, the tree becomes bigger and more accurate in prediction and performs better both on training and test sets, and as the MinNumObj is increased, the tree becomes smaller and the accuracy of prediction or overall performance decreases. This is because, in case of a small value of MinNumObj, the tree will grow until each leaf has only a small number of datapoints and hence, the classifier performs better

Reduced Error Pruning refers to removing a subtree at some internal node and making it a leaf and assigning to it, the most common class at that node, as long as the performance on the validation set remains the same or decreases. We can see that the performance on the test data more or less remains the same even after pruning the tree, though the performance on the training data decreases a little in comparison to the situation when there is no pruning.

The effect of MinNumObj and Reduced Error Pruning has been depicted below :

MinNumObj	**Reduced Error Pruning**	**Accuracy on train_set**	**Accuracy on test_set**
1	No	99.9857 %	100 %
1	Yes	99.9571 %	100 %
2	No	99.9857 %	100 %
2	Yes	99.9571 %	100 %
5	No	99.9857 %	100 %
5	Yes	99.9 %	100 %
10	No	99.9857 %	100 %
10	Yes	99.9 %	100 %
25	No	99.8714 %	99.3772 %
25	Yes	99.4286 %	99.3772 %
50	No	99.4143 %	99.3772 %
50	Yes	98.8714 %	99.3772 %
100	No	98.3857 %	99.3772 %
100	Yes	98.3857 %	99.3772 %

As we can observe from the above table, the best model would have MinNumObj=2 and we don't prune the tree. The confusion matrix for the test set is as follows :

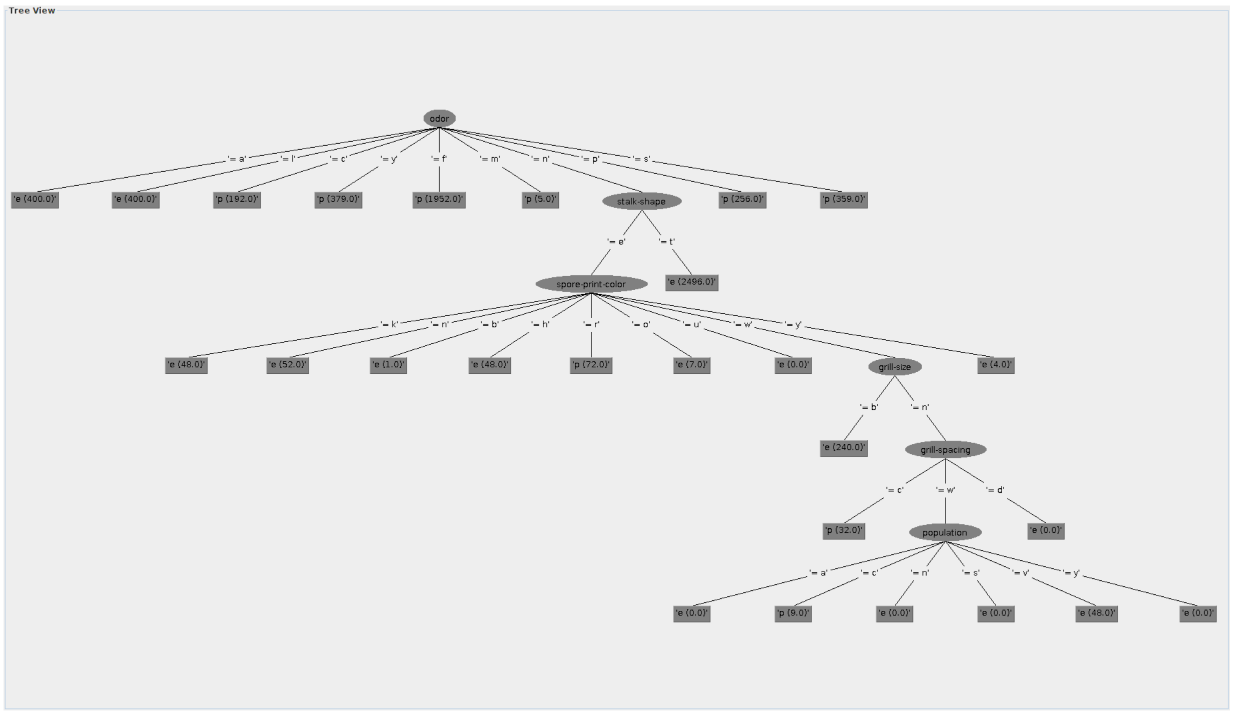
Classified as	**Edible**	**Poisonous**
Edible	464	0
Poisonous	0	660

The results obtained are as follows :

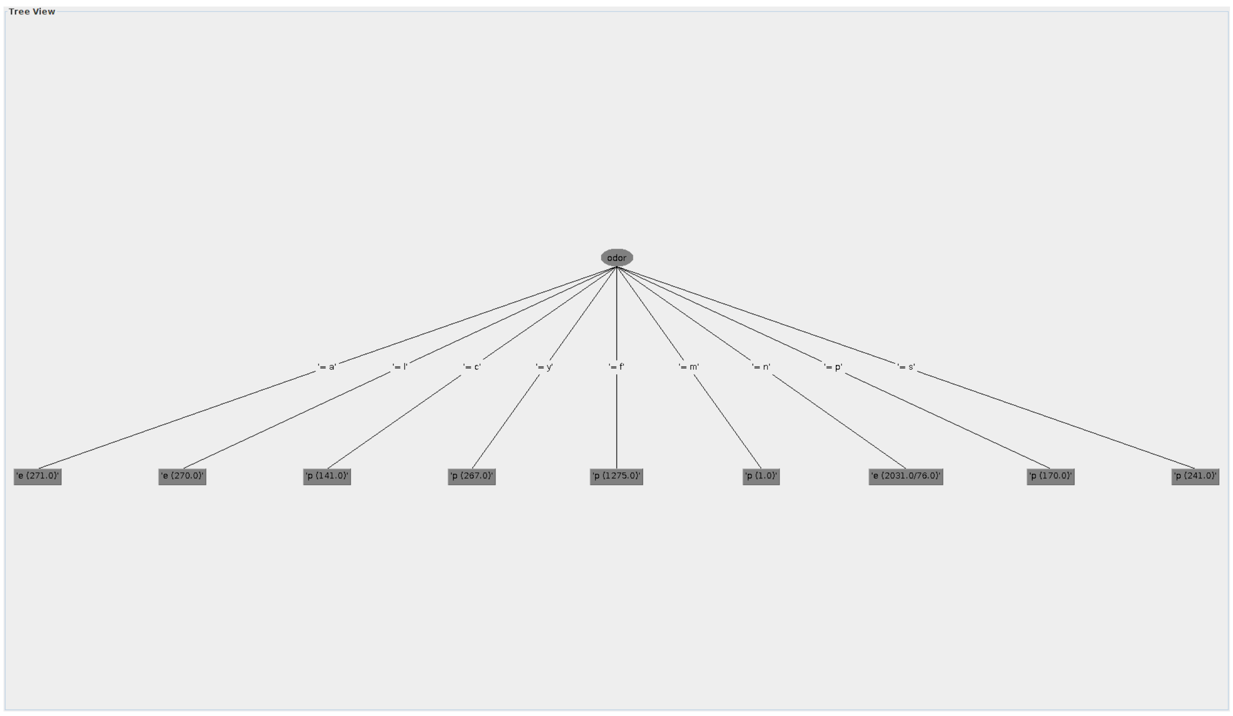
Class	**Recall**	**Precision**	**F-Measure**
Edible	1	1	1
Poisonous	1	1	1

As we can infer from the tree given above, the important features in deciding whether a mushroom is edible or not are Odor, Stalk-shape, Spore-print-color, Grill-size, Grill-spacing and Population.

The tree generated :



With MinNumObj=100 and Reduced Error Pruning, we get 99.3772% accuracy on the test set and hence, the most important feature to decide whether a mushroom is edible or not is Odor as we can see from the tree given below.



The confusion matrix for the test set is as follows :

Classified as	**Edible**	**Poisonous**
Edible	464	0
Poisonous	7	653

The results obtained are as follows :

Class	**Recall**	**Precision**	**F-Measure**
Edible	1	0.985	0.993
Poisonous	0.989	1	0.995