CS5691: Pattern Recognition and Machine Learning

Programming Assignment 1

(Team 7)

Nithya S - CH16B113 Ashwin S Nair- CH16B033

QUESTION 1

Using Dataset 1:

Visual analysis of the dataset:

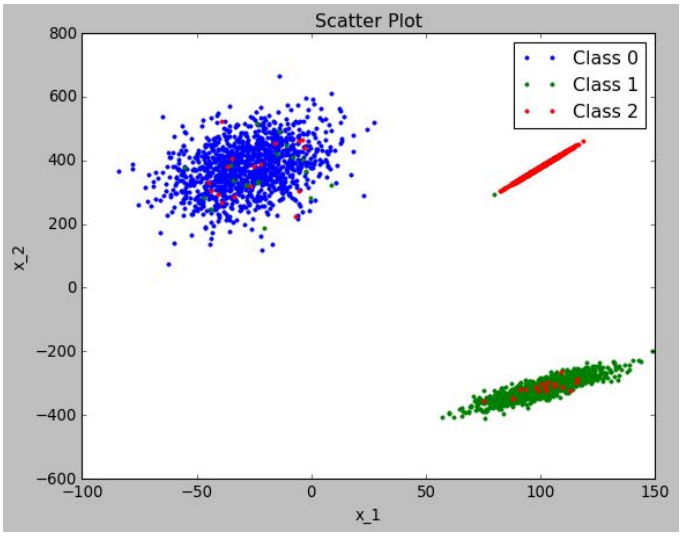


Fig: Bivariate data with 3 classes

In order to make a reasonable assumption for the **class conditional distribution**, histograms of each feature vector were plotted for all three classes. As can be seen from the following figures, the **histograms are unimodal**. Hence it would be reasonable to assume the class conditional distributions to be **Gaussian**.

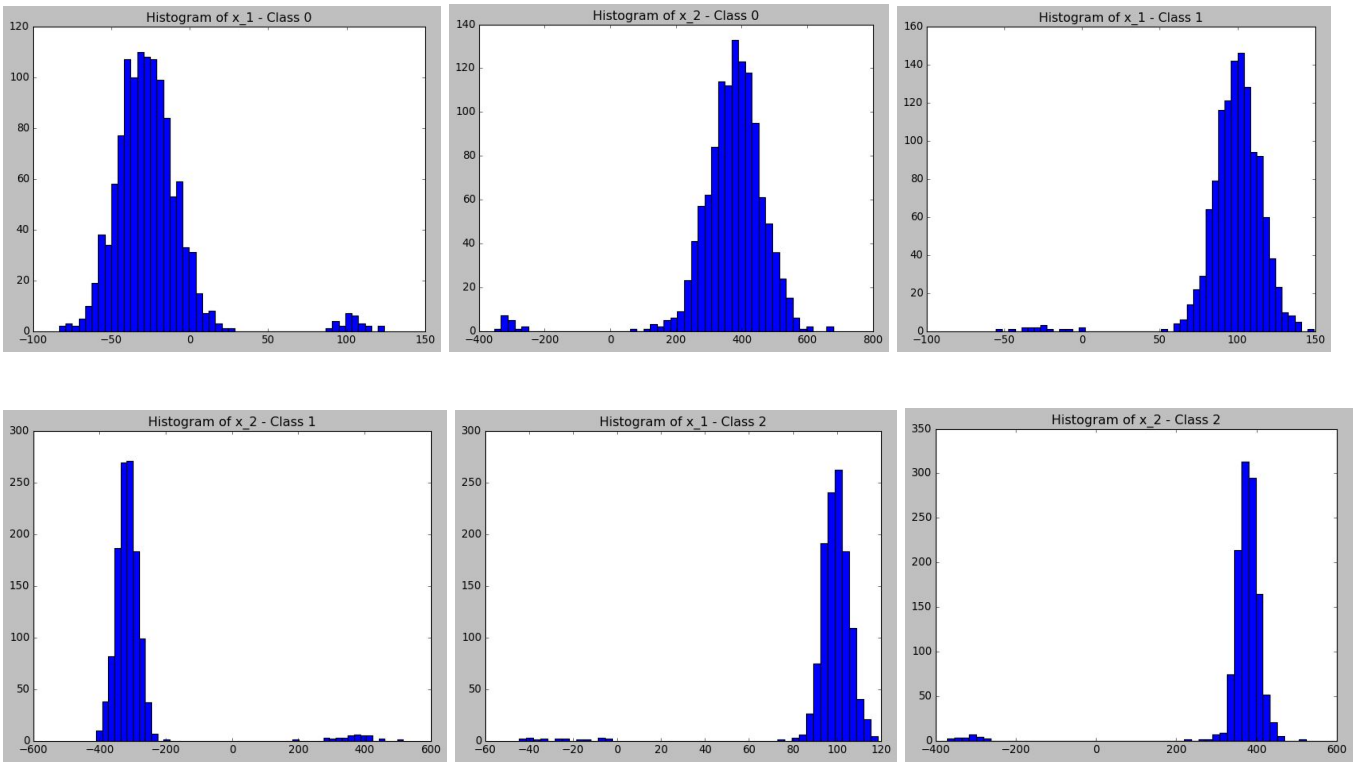


Fig: Histograms of feature values (class-wise)

Assumption: For models 2 and 4 (classifiers using the same covariance matrix for all classes), the covariance matrix that is used is a weighted (based on prior) average of the MLE estimates of covariance matrices for the three classes.

(i) Table of classification accuracies for all the models on training data and test data.

Models 2 to 5 perform equally well. Hence all 4 models are equally good (based on test and train accuracy). Hence, models 2, 3, 4, 5 have been selected for the consecutive parts.

Model number	Train Accuracy	Test Accuracy
1	0.794444	0.79
2	0.970833	0.974444
3	0.970833	0.974444
4	0.970833	0.974444
5	0.970833	0.974444

(ii) Confusion matrix on test set for the best model among those in parts (a)-(e).

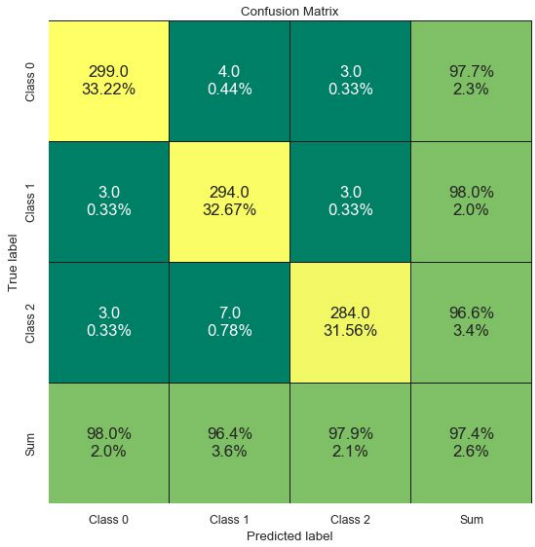


Fig: Models 2, 3, 4, 5 generate the same confusion matrix (test data).

(iii) Decision boundary and decision surface of the best model. Superpose the training data on this plot.

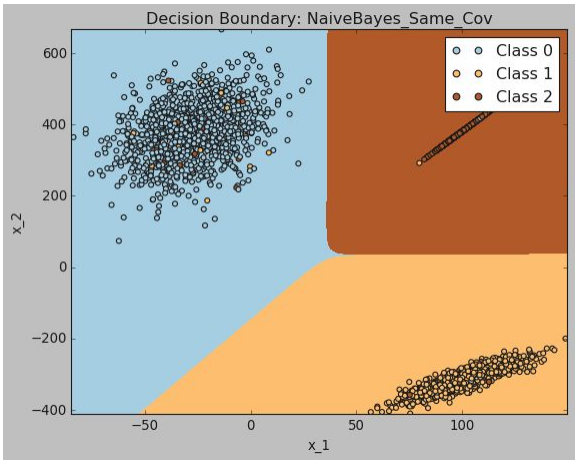


Fig: Model 3

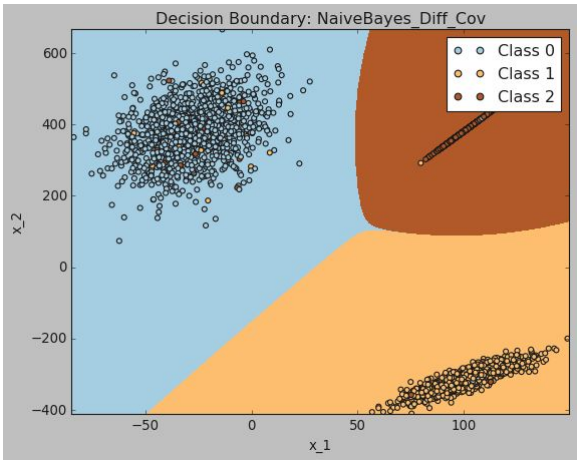


Fig: Model 4

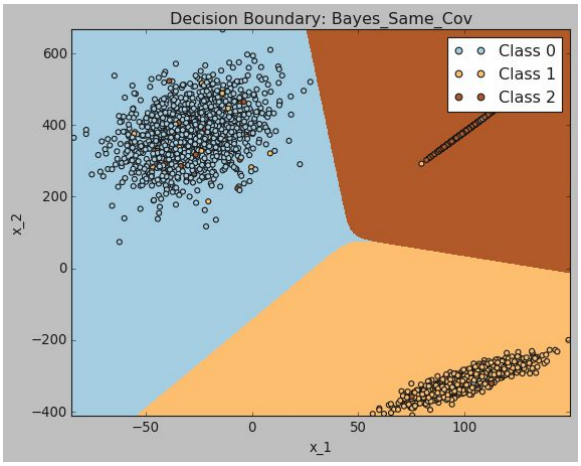


Fig: Model 5

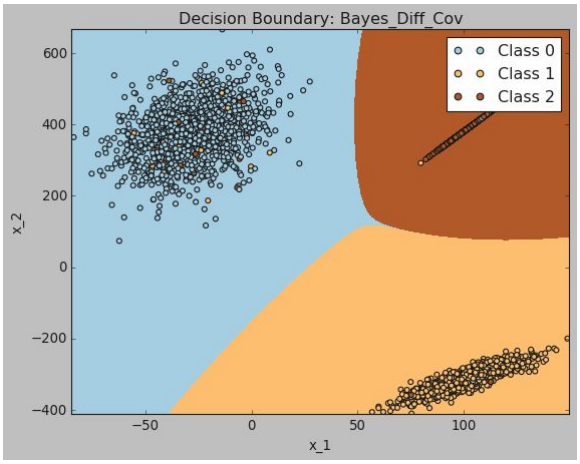


Fig: Model 6

(iv) Contour curves and eigenvectors of the covariance matrix for the best model.

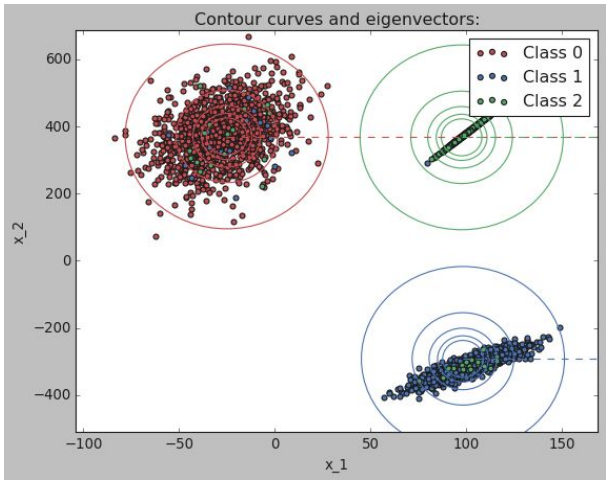


Fig: Model 3

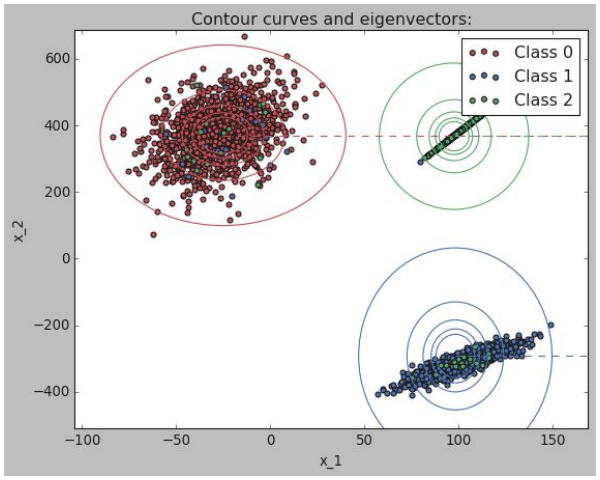


Fig: Model 4

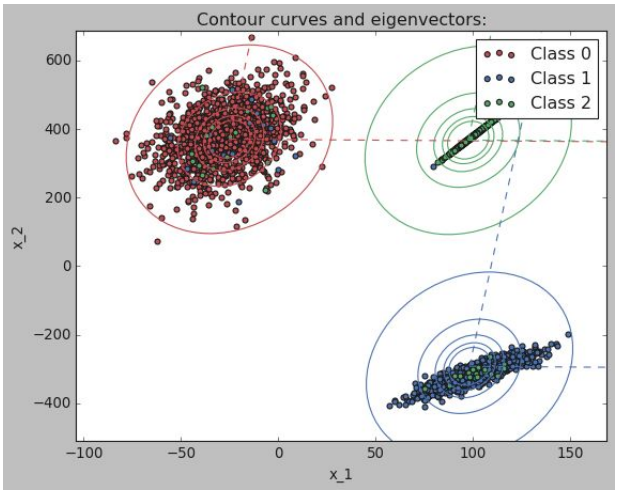


Fig: Model 5

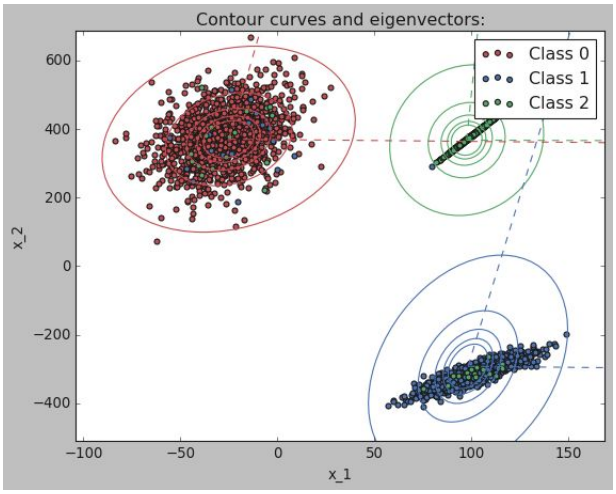
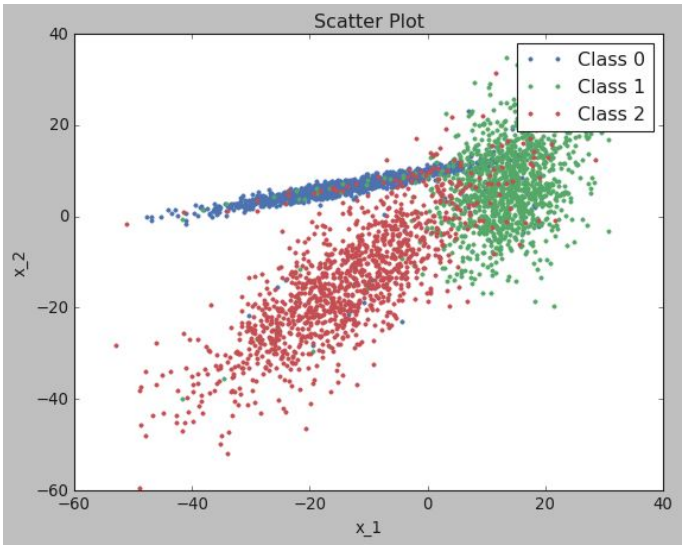


Fig: Model 6

Using Dataset 2: (Proceeding with the same approach as in case of Dataset 1)

Visual analysis of the dataset:



In order to make a reasonable assumption for the **class conditional distribution**, histograms of each feature vector were plotted for all three classes. As can be seen from the following figures, the **histograms are unimodal**. Hence it would be reasonable to assume the class conditional distributions to be **Gaussian**.

Fig: Bivariate data with 3 classes

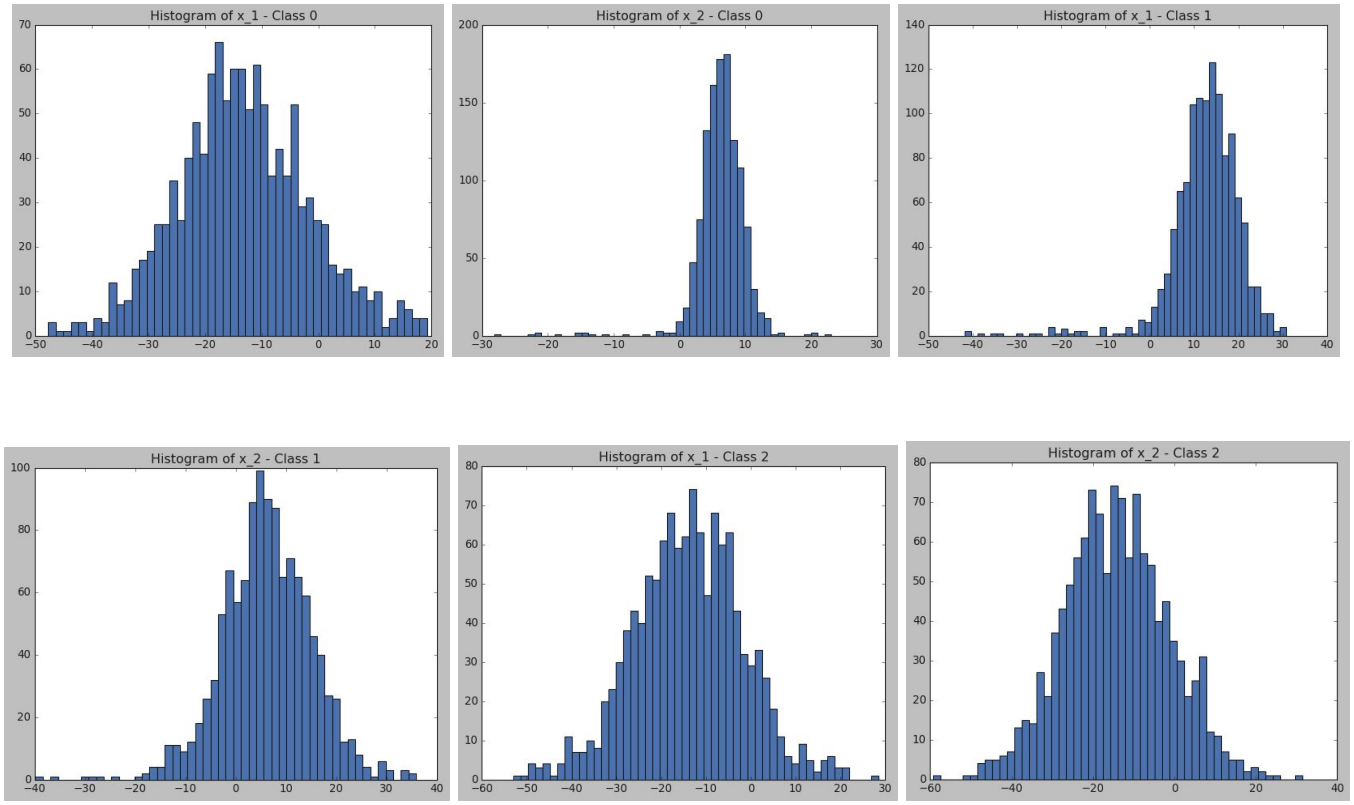


Fig: Histograms of feature values (class-wise)

(i) Table of classification accuracies for all the models on training data and validation data.

Model 5 is the best in terms of test and train accuracy. Hence it is chosen for the subsequent parts.

Model number	Train Accuracy	Test Accuracy
1	0.858333	0.867778
2	0.853333	0.862222
3	0.882222	0.891111
4	0.874722	0.882222
5	0.903611	0.908889

(ii) Confusion matrix on test set for the best model among those in parts (a)-(e).

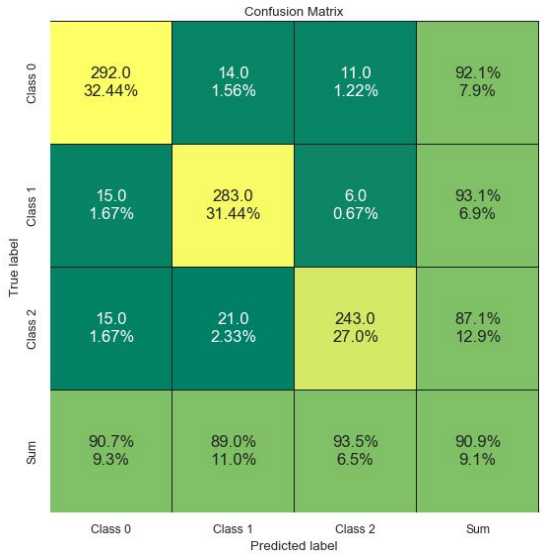


Fig: Model 5 - Confusion matrix (test data)

(iii) Decision boundary and decision surface of the best model. Superpose the training data on this plot.

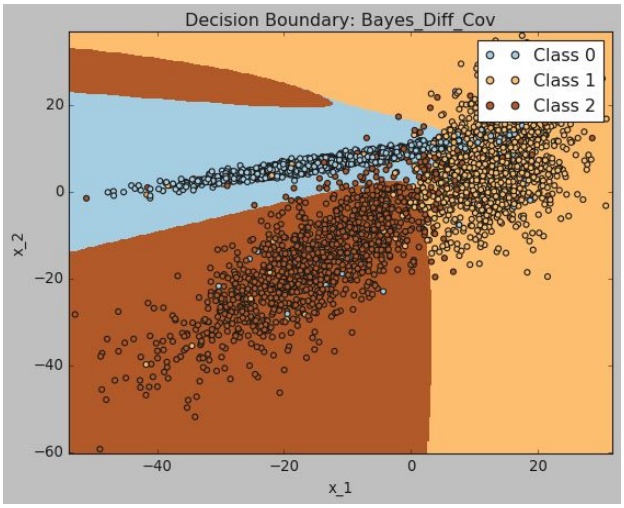


Fig: Model 5 - Decision region

(iv) Contour curves and eigenvectors of the covariance matrix for the best model.

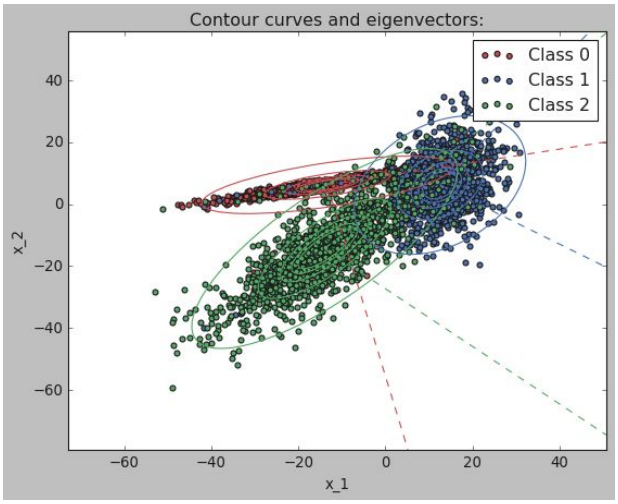


Fig: Model 5 - Contour curves & Eigenvectors

Important Observations and Inferences:

- It can be seen that **Models 2, 3, 4, 5 performed equally well** (in terms of test and train accuracy) in case of **Dataset 1**. This does not mean that all these models are equivalent. It can be clearly seen that the **decision regions are different** for these models.
- When Dataset 2 was used, **Model 5 outperformed** the other models.
- The above mentioned observations can be explained by looking at the nature of data points in each Dataset. The **data points** of different classes **are very well-separated** in **Dataset 1**. As a result, even the poor models are able to classify the data points with a high accuracy. The same cannot be said for Dataset 2.

QUESTION 2

Consider Bayesian estimation of mean on one-dimensional Gaussian dataset 3. Suppose prior of the mean is $P(\mu) = N(\mu_0; \sigma_0)$.

(a) Estimate σ (use the sample variance).

Given:

- Prior: $P(\mu) \sim N(\mu_0, \sigma_0)$
- Dataset: data3.csv file which is a one dimensional matrix with 1500 elements.
- Dataset is Gaussian with unknown mean $\Rightarrow P(x|\mu) \sim N(\mu, \sigma^2) \rightarrow$ Likelihood term

Posterior: $P(\mu|D) \sim N(\mu_n, \sigma_n^2)$, where μ_n and σ_n^2 are given by:

$$\frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2} \quad \text{and} \quad \frac{\mu_n}{\sigma_n^2} = \frac{n}{\sigma^2}\overline{x}_n + \frac{\mu_0}{\sigma_0^2}$$

σ^2 is the variance in the dataset given that the mean = μ
It can be estimated by the sample variance of the given dataset.

$$\sigma^2_{Estimate} = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})^2$$

$$\sigma^2_{Estimate} = 1.9657$$

(b) Assuming that $\mu_0 = -1$ plot your estimated densities P(x|D) for n = 10; 100; 1000 with the following rates for r $\sigma^2 / \sigma_0^2 = 0.1, 1, 10, 100$.

$$P(x|D) = \int P(x|\mu)\, P(\mu|D)\, d\mu$$

Where $P(\mu|D) \sim N(\mu_n, \sigma_n^2)$ and $P(x|\mu) \sim N(\mu, \sigma^2)$. Upon simplification we get:

$$P(x|D) \sim N(\mu_n, \sigma^2 + \sigma_n^2)$$

Let alpha (α) = σ^2 / σ_0^2. For each value of $\alpha = \sigma^2 / \sigma_0^2$ and n, we calculate the values of μ_n and σ_n^2. From this P(μ|D) is obtained.

σn values for different n and alpha

σ^2 /σ0^2	0.1	1	10	100
n = 10	0.618545	0.592701	0.439559	0.187429
n = 100	0.196479	0.195601	0.187429	0.139001
n = 1000	0.0621599	0.062132	0.0618545	0.0592701

Figure: Values of estimates of σ_n for different values of α and n

μn values for different n and alpha

σ^2 /σ0^2	0.1	1	10	100
n = 10	0.773202	1.44506	-0.316244	-0.831362
n = 100	0.858108	0.767256	0.716555	0.0987981
n = 1000	1.01295	0.948521	1.01367	0.853376

Figure: Values of estimates of μ_n for different values of α and n

For every combination of n and σ^2 / σ_0^2, P(x|D) ~ N (μ_n, $\sigma^2 + \sigma_n^2$) is plotted.

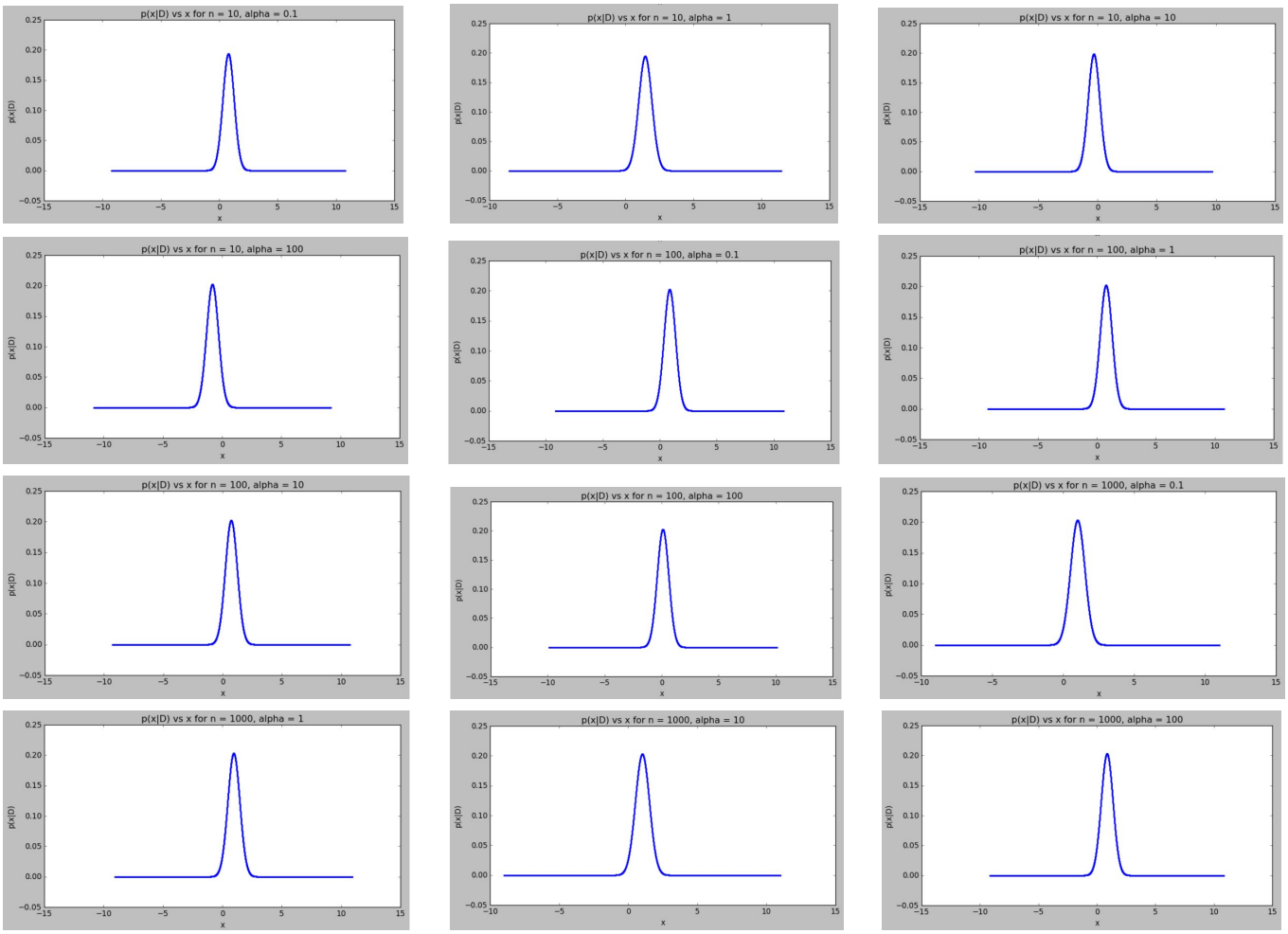


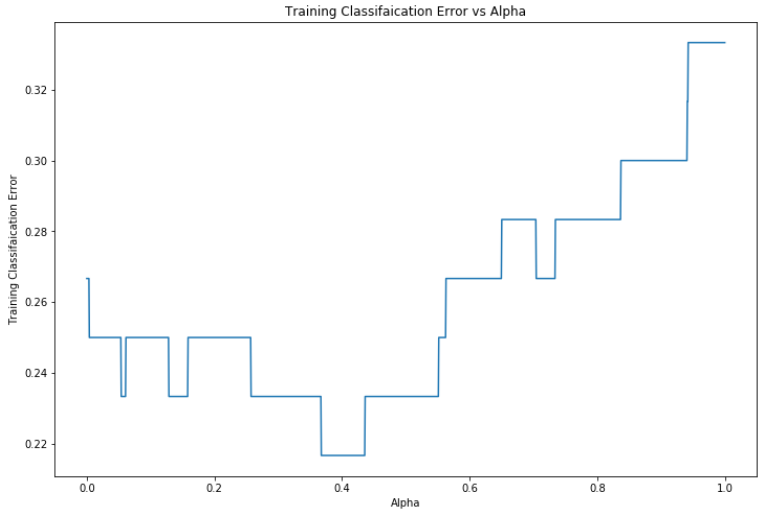
Figure: P(x|D) plots for different n and alpha

QUESTION 3

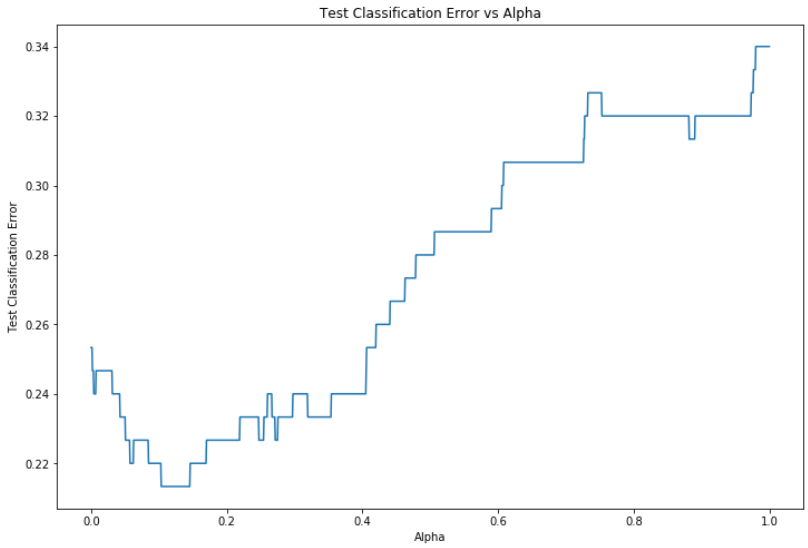
a) Report the estimated means and covariances of your data

Class	Estimated mean	Actual Mean	Estimated Covariance	Actual Covarinace
1	[0.3392595 0.61656112 -0.34167993]	[0. 0. 0.]	[[3.24175107 0.15192638 0.38377377] [0.15192638 6.84813478 0.18501716] [0.38377377 0.18501716 1.98197715]]	[[3 0 0] [0 5 0] [0 0 2]]
2	[1.03912963 5.82596232 -3.00765116]	[1 5 -3]	[[1.1529935 0.44403436 1.14503538] [0.44403436 4.31991909 1.64468026] [1.14503538 1.64468026 6.74591011]]	[[1 0 0] [0 4 1] [0 1 6]]
3	[1.16745981 -0.13912728 -0.88972324]	[0. 0. 0.]	[[7.78280415 -1.53913241 -1.65767207] [-1.53913241 8.94858583 0.7392485] [-1.65767207 0.7392485 8.15284836]]	[[10. 0. 0.] [0. 10. 0.] [0. 0. 10.]]

b) Plot the classification error on training data as a function of α, where $0 < \alpha < 1$.



(c) Use your code from part (a) to generate 50 test points from each category. Plot the test error as a function of α.



QUESTION 4

Create a 100 × 100 matrix of random numbers between 0 and 1 such that each entry is highly correlated with the adjacent entries. Perform SVD on it.

To create a 100 x 100 correlated matrix, a one dimensional feature of shape (100,1) was created using numpy's 'random.random_sample' function. Using this column (feature), subsequent features, with a **correlation coefficient of 0.999 with the initial feature**, were created. This was repeated till 100 features were obtained. After this, the elements of the newly generated features were scaled appropriately so that all the elements fall in the **interval (0,1)**. These **100 features are highly correlated**.

We perform SVD on the generated matrix A by decomposing it into U, s, and VT where
U = Eigenvectors of AA^T
V = Eigenvectors of A^TA
s = Diagonal Matrix with the elements as eigenvalues of $A.A^T$
and $A = U\ s\ V^T$

(a) Compute the Frobenius norm of the matrix A.

- For the highly correlated matrix, Frobenius norm = **3062.538**
- For the highly uncorrelated matrix, Frobenius norm = **3258.759**

(b) What fraction of the Frobenius norm of A is captured by the top 10 singular vectors?

For any matrix,

Frobenius Norm = Sum of squares of all elements = Sum of squares of singular values = Sum of Eigenvalues

Hence, to see what percentage of the frobenius norm is captured by the top ten singular vectors, the percentage of the squares of the singular values captured by the top ten singular values is computed.
Fraction of Frobenius norm covered = (Sum of top ten eigenvalues) / (Sum of all eigenvalues)

- For the highly correlated matrix, the fraction is **0.9958**
- For the uncorrelated matrix, the fraction is **0.8198**

(c) What fraction of the Frobenius norm of A is captured by random 10 singular vectors?

- For the highly correlated matrix, the fraction of singular values captured is **0.0007224**
- For the uncorrelated matrix, the fraction of singular values captured is **0.03079**

(d) Plot a graph showing the number of singular vectors required to capture 50%, 75% & 95% of the data respectively?

To get the singular vectors to capture the above percentages of Frobenius norm, the number of eigenvalues, that have to be summed up until the (Sum of eigenvalues selected) / (Sum of all eigenvalues) is greater than 50%, 75% and 95% respectively, is counted.

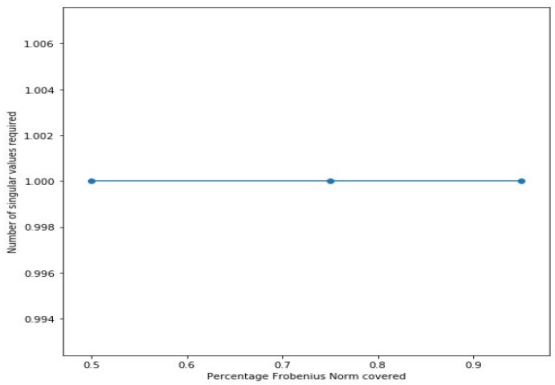


Fig: For highly correlated matrix

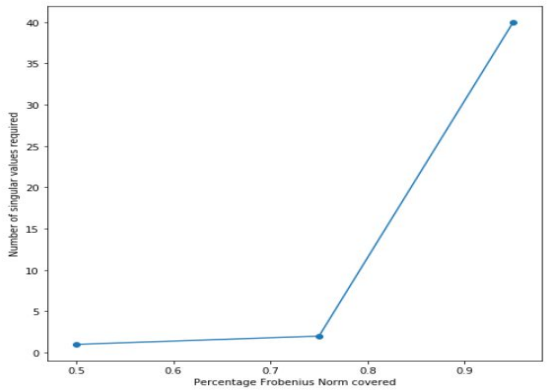


Fig: For uncorrelated matrix

(e) Write down your observations.

We observe that when the columns of the matrix are highly correlated, very few singular vectors are required to capture even high variances, whereas when the columns of the matrix are uncorrelated, we require large number of singular vectors to capture high variances.

In our case, to capture 95% variance, we needed only one singular vector in the case of the highly correlated matrix whereas we needed around 40 singular vectors in the case of the uncorrelated matrix.

Conversely, in the case of highly correlated matrix, top ten singular vectors are able to capture more than 99.5% variance whereas in the uncorrelated matrix, the top ten singular vectors are able to capture only around 82% variance.

(f) Repeat all the above five parts for 100 × 100 matrix of statistically independent numbers between 0 and 1.

For generating a 100 x 100 matrix of statistically independent numbers, numpy's 'random.random_sample' function was used. Since, the generated numbers in each column are completely random, the columns will be statistically independent.

NOTE: Answers to parts (a) through (e) are provided along with the answers for highly correlated matrix

QUESTION 5

(a) First convert it to grayscale.

Please refer to the code.

(b) Reconstruct the image using top N (N = 10%, 25%, 50%) principal components. Plot the reconstructed images along with their corresponding error image.

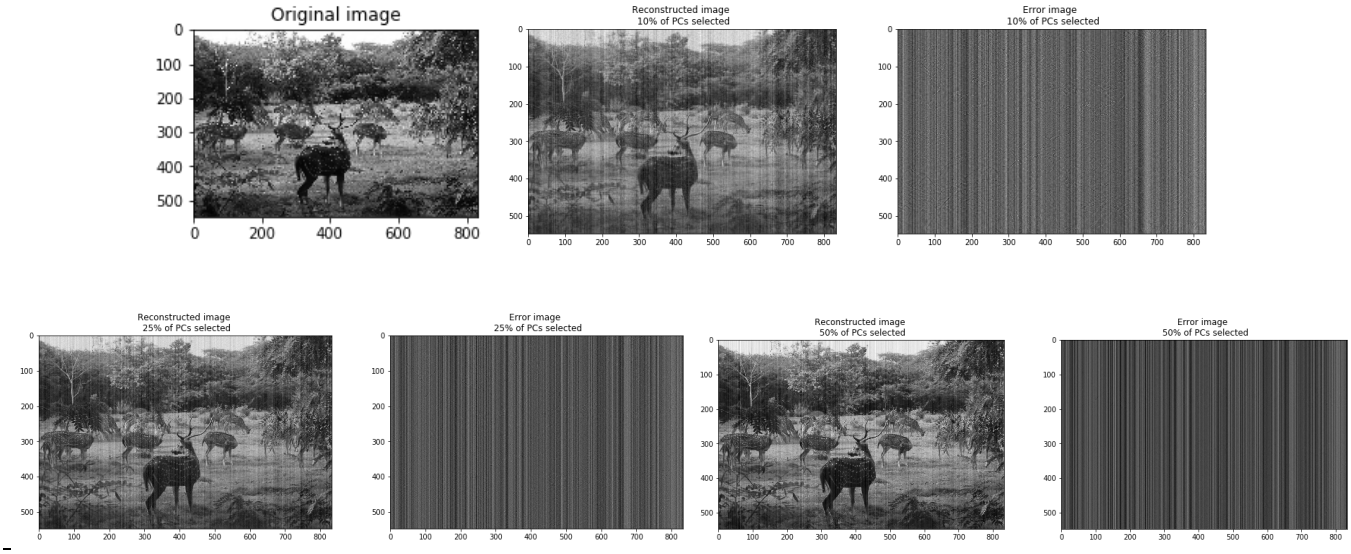
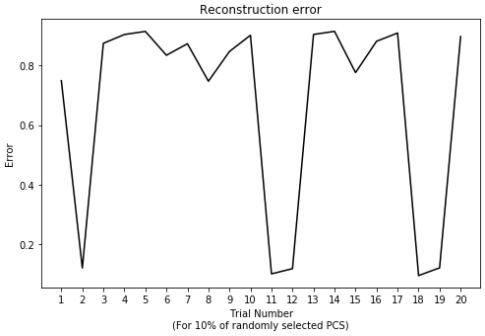


Fig: Reconstructed image and the corresponding error image

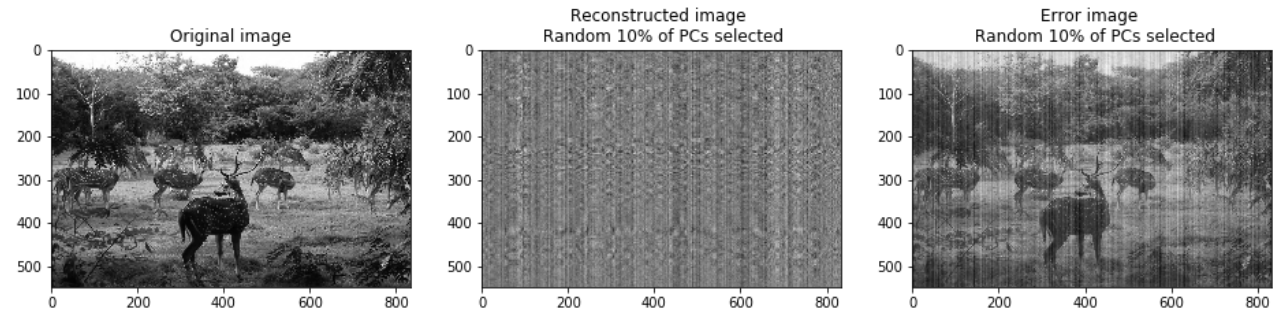
Image number	Frobenius norm (10% of PCs selected)	Frobenius norm (25% of PCs selected)	Frobenius norm (50% of PCs selected)
7	0.982027	0.99229	0.996635

(c) Try random N (N = 10%) principal components instead of top N. Plot the reconstructed image along with its error image. Explain the observed trend in the result.



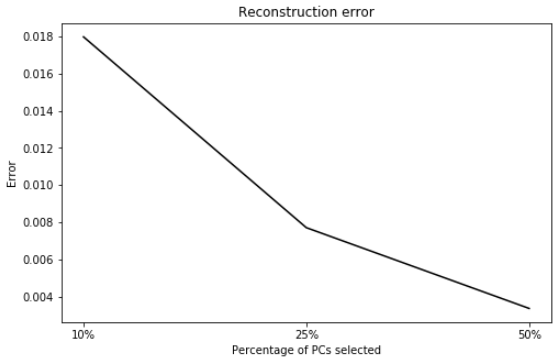
Random 10% of the PCs (principal components) were selected, after which the error in the reconstructed image was obtained. This was repeated 20 times and the error has been plotted against the trail number. It can be seen that there is no trend in the error obtained. It changes drastically based on the choice of PCs and is always higher than what was obtained while choosing the first 10% PCs (which is 0.017972).

Result for one trial:



It can be seen that the error image is closer to the original image. This is because, **choosing random 10% of PCs does not capture the characteristics of the original image** and whatever was not captured, appears in the error image.

(d) Plot a comparative graph of the reconstruction error vs N.



(e) Write your inference in one or two lines.

It can clearly be seen that choosing more number of PCs reduces the error. This is because the variance captured increases with the number of PCs chosen. From the previous part, it can be inferred that the order of PCs (descending in terms of eigenvalues) also matters

QUESTION 6

Generate 100 sample points in the domain (0,1). For each sample point, generate a target value using the function assigned to your team, with additive Gaussian Noise having mean zero and variance 0.2. Divide the data set as 80:20 (train:test). Perform polynomial regression. In particular, do the following: (Plot the data, the target function and the regression output. Also tabulate the coefficients obtained for each model).

- numpy's random.random_sample function (which generates data points between 0 and 1) was used to generate data of the shape (100,1).
- Gaussian noise was added using numpy's random.randn function, which generates data from a Gaussian distribution with mean = 0 and variance = 1. Multiplying this with $0.2^{0.5}$ gives us a Gaussian noise with mean = 0 and variance = 2 .

Using the function assigned to us,
$y = \exp(\sin(2\pi x) + \log_e(x)$
the target variable of the shape (100,1) was generated

The dataset was split into train and test data, in the ratio 80:20 by shuffling the data and choosing the first 80 data points to be in train and remaining 20 data points to be in the test dataset.

(a) Choose 10 points from the training data-set and perform regression for degrees: {1,3,6,9}

Since the training dataset was generated by shuffling the initial dataset, the first 10 data points from the train set is used for modelling.

Formula for linear regression parameters,

$B = (XX^T)^{-1} (X^T y)$, ------- (1)

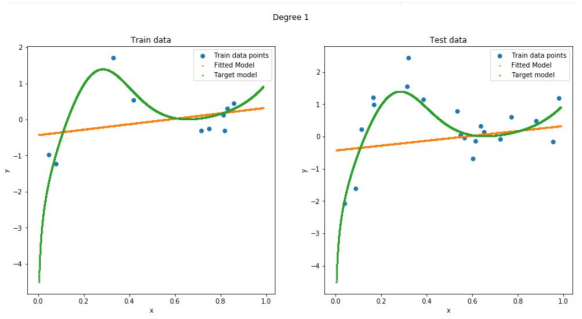
For the purpose of performing polynomial regression, polynomial features of the form x, x^2, x^3,...., x^n were generated, where n = {1,3,6,9} in this case.

Using the new transformed dataset, X_new = [x, x^2, x^3,...., x^n] of the shape (100,n), linear regression was performed using formula (1).

(b) For each of the models above, analyze over-fitting by varying the data-set size

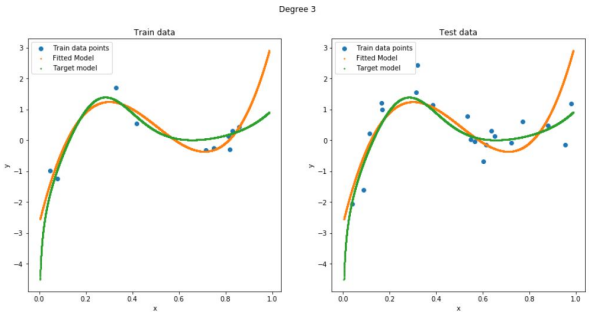
For the purpose of studying overfitting with varying data size, linear regression was performed on datasets of sizes 10, 40 and 80.

Observations:



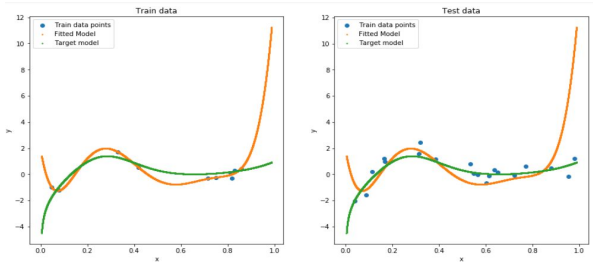
Coefficients for n = 1

1	x^1
-0.425068	0.757785



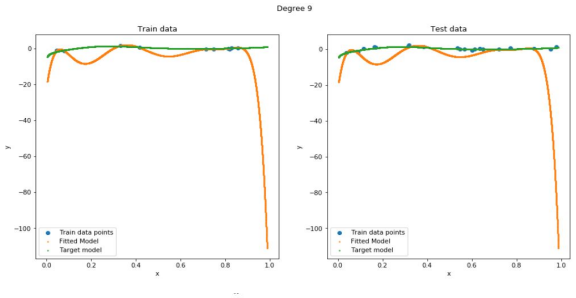
Coefficients for n = 3

1	x^1	x^2	x^3
-2.66521	30.315	-71.7423	47.3525



Coefficients for n = 6

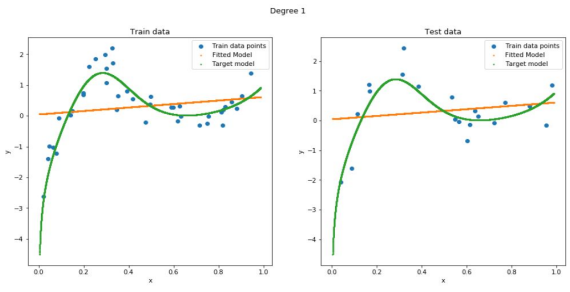
1	x^1	x^2	x^3	x^4	x^5	x^6
1.777	-102.239	1118.89	-4504.34	8369.98	-7348.56	2478.39

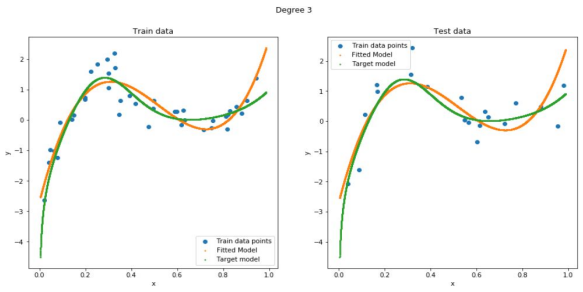


Coefficients for n = 9

1	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9
-21.7452	917.98	-13521.5	82938	-241867	312990	-17562	-409206	421092	-135917

Figure: Data points, target function, model and mean squared error for train and test data for polynomial degree 1,3,6,9 (for ten data points)

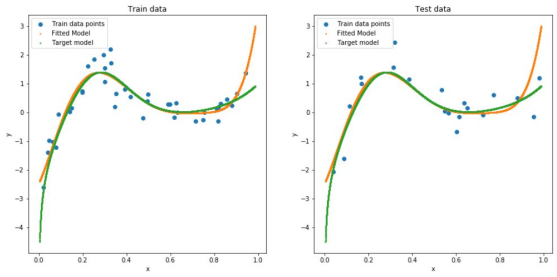




Coefficients for n = 1

1	x^1
0.0496817	0.567401

Degree 6

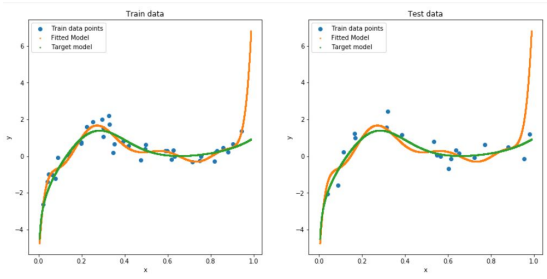


Coefficients for n = 6

1	x^1	x^2	x^3	x^4	x^5	x^6
-2.46879	16.8974	99.2154	-742.386	1630.25	-1534.87	537.012

Coefficients for n = 3

1	x^1	x^2	x^3
-2.65599	29.4946	-67.8658	43.6979

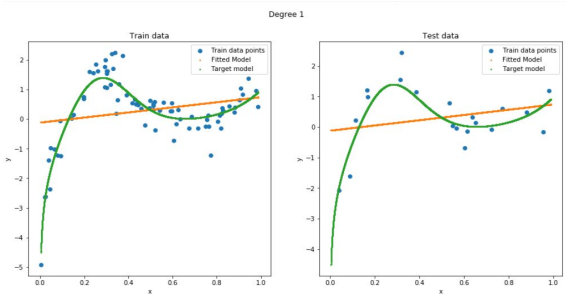


Coefficients for n = 9

1	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9
-5.47181	195.354	-3290	28673.1	-136077	375264	-619954	605188	-321956	71970.8

Figure: Data points, target function, model and MSE for train and test data for degree 1,3,6,9 (for thirty data points)

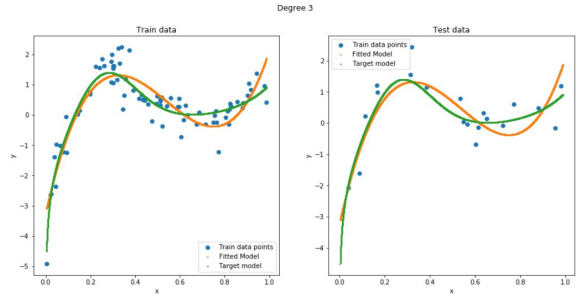
Degree 1



Coefficients for n = 1

1	x^1
-0.110518	0.859736

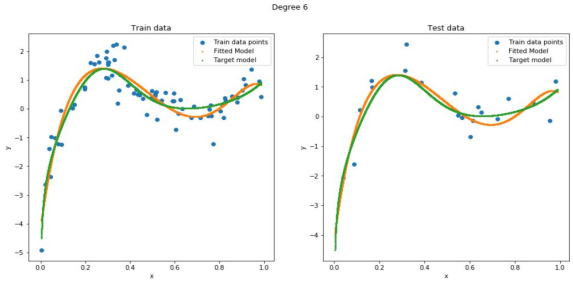
Degree 3



Coefficients for n = 3

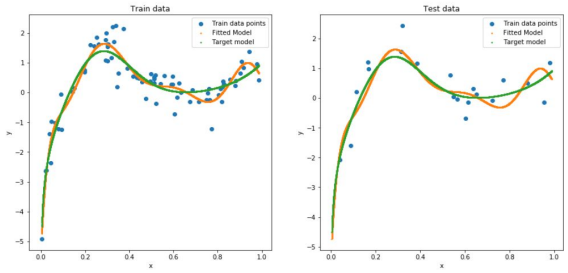
1	x^1	x^2	x^3
-3.22598	32.719	-72.2071	44.8581

Degree 6



Coefficients for n = 6

1	x^1	x^2	x^3	x^4	x^5	x^6
-4.1254	56.9604	-225.664	478.748	-661.934	541.388	-184.603



Coefficients for n = 9

1	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9
-5.32163	156.282	-2303.57	18557.4	-82485	213415	-330521	301978	-150035	31244.3

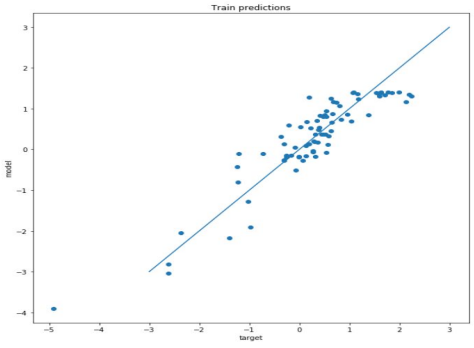
Figure: Data points, target function, model and mean squared error for train and test data for polynomial degree 1,3,6,9 (for eighty data points)

- We observe that when the size of the data is very small, like 10, models of lower orders work better (in our case n=3 yields best results) since there are very few data points, higher order polynomial regression will lead to a lot of variance.
- When all of the train data is used (size = 80), n = 6 yields the best results since a lot more data is available to us. n = 9 leads to overfitting and hence increase in test mean squared error.

<u>Conclusion:</u> Lower order polynomials lead to high bias and higher order polynomials lead to high variance, hence a bias variance tradeoff exists and the **lowest mean squared error is achieved for n=6 when the entire train data is used**.

(c) Show the scatter plot with target output t_n on x-axis and model output y (x_n, w) on y-axis for the best performing model, for training data and test data

We observed that the best performing model is for n = 6, i.e. 6th order polynomial. On performing regression with n = 6 and predicting the y values for training and test data, we plotted the target t_n.



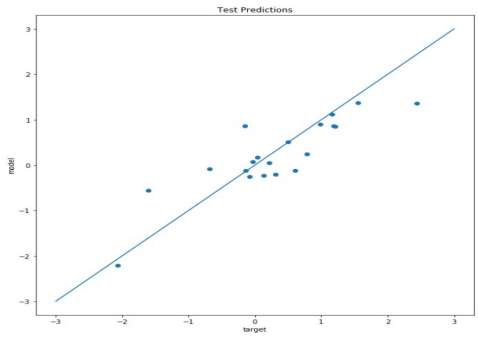


Figure: Model prediction vs target for train and test dataset

(d) Plot the root-mean-square (RMS) error.

Using the formula for RMSE,

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

The RMS error for n = 1,3,6 and 9 for both train and test dataset was calculated and plotted against their respective n.

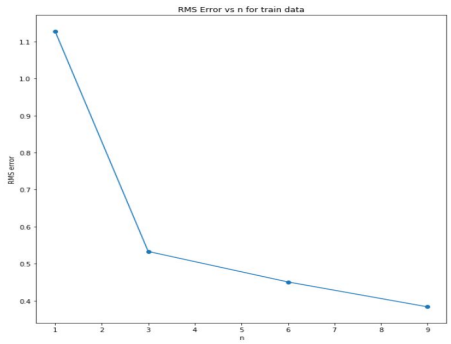


Figure: RMS error for train dataset

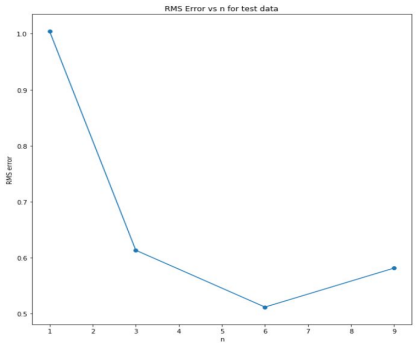


Figure: RMS error for test dataset

The RMSE is the minimum for n = 6 for the test dataset as seen before.