Pattern Recognition Project 2

least squares regression and nearest neighbor classification

Task 2.1:
Least Squares Method for Polynomial Regression

Problem Statement:

- Use method of least squares to fit polynomial model to the data
- Plot the results for $d \in \{1, 5, 10\}$
- Use the resulting model to predict missing weight values

Method:

Estimation of weights using: $w = \left(XX^{\top}\right)^{-1} X y$

Problem faced:

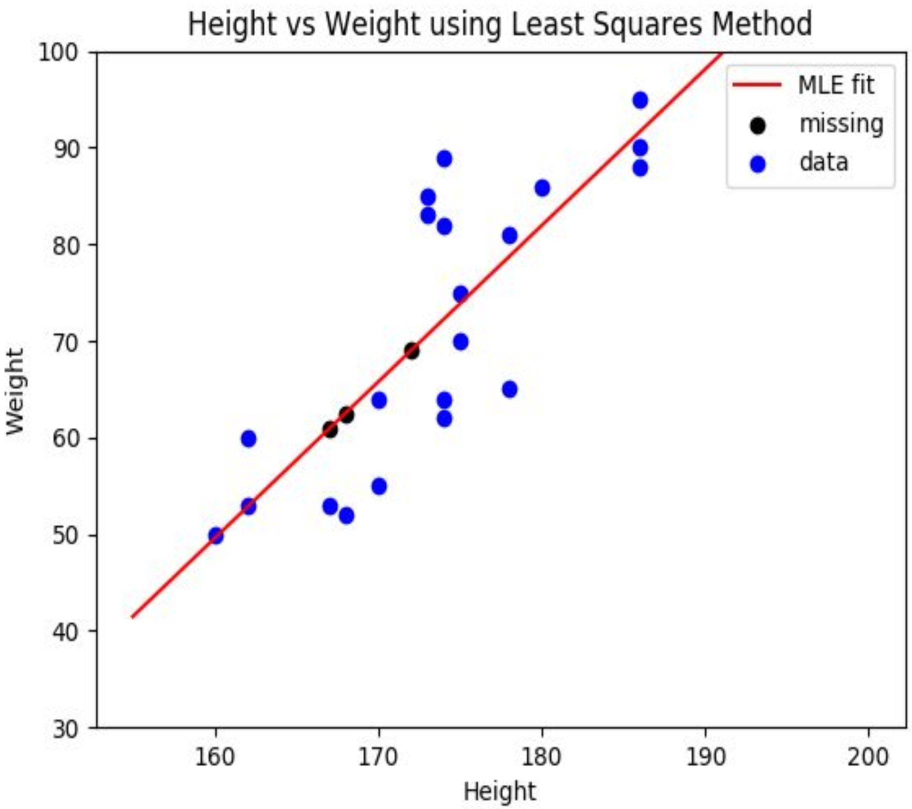
Difficulty in finding inverse of matrices with large values for d=5 and 10.

Solution:

To resolve we used standardization of matrix values for numerical stability

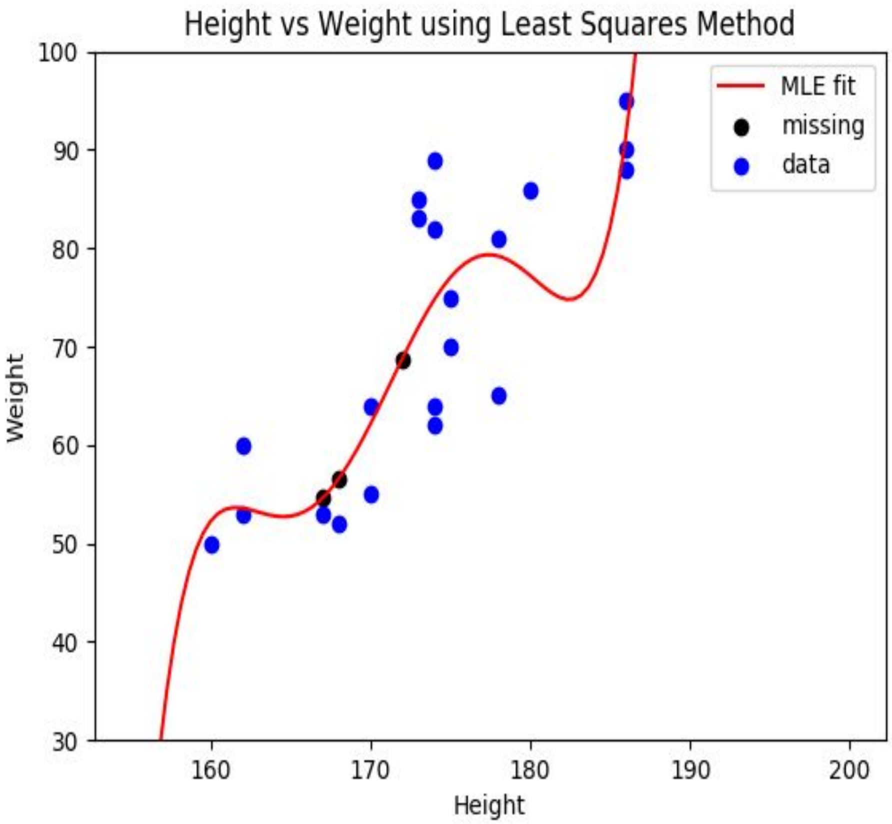
Results for order 1:

Height	168.0	172.0	167.0
Weight	62.51	68.98	60.89



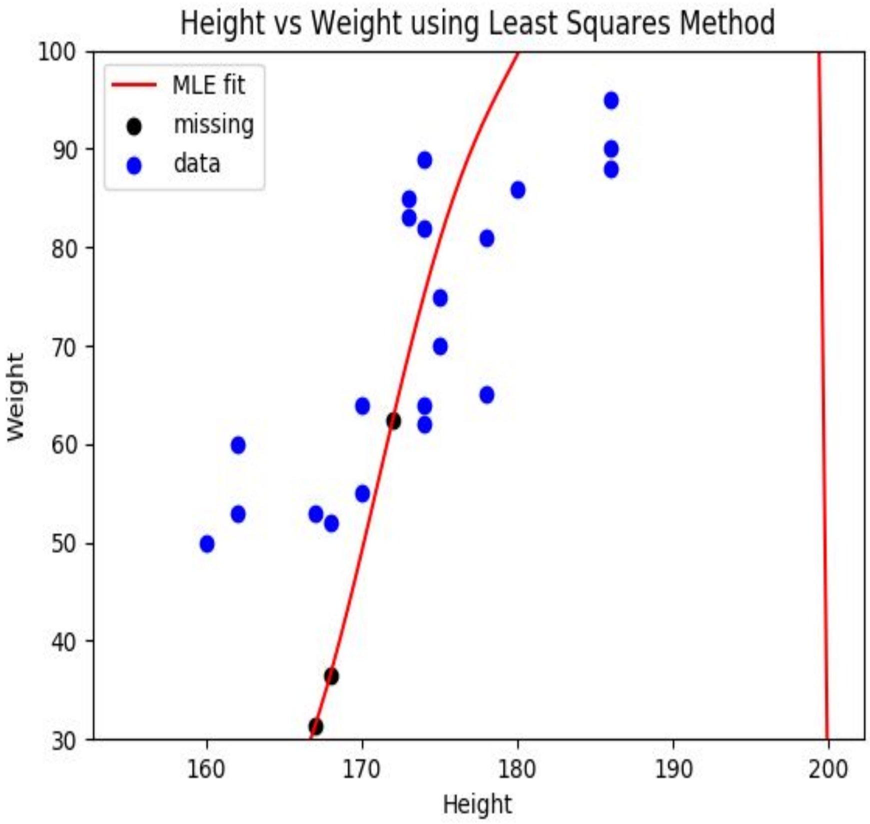
Results for order 5:

Height	168.0	172.0	167.0
Weight	56.53	68.76	54.58



Results for order 10:

Height	168.0	172.0	167.0
Weight	36.57	62.46	31.30



Task 2.2:
Conditional Expectation for Missing Value Prediction

Problem Statement:

- To fit a Bivariate Gaussian to model the joint density of x(height) and y(weight).
- Use the resulting model to predict missing weight values

Approach

PDF: $$P(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[-\frac{z}{2(1-\rho^2)}\right],$$

Where :

$$z \equiv \frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2},$$

$$\rho \equiv \operatorname{cor}(x_1, x_2) = \frac{V_{12}}{\sigma_1\sigma_2}$$

The conditional expectation of X given Y satisfies : $$\mathbf{E}[X \mid Y] = \mathbf{E}[X] + \rho\frac{\sigma_X}{\sigma_Y}(Y - \mathbf{E}[Y]).$$

The conditional distribution of X given Y is normal with mean E[X | Y] and variance σ2x^

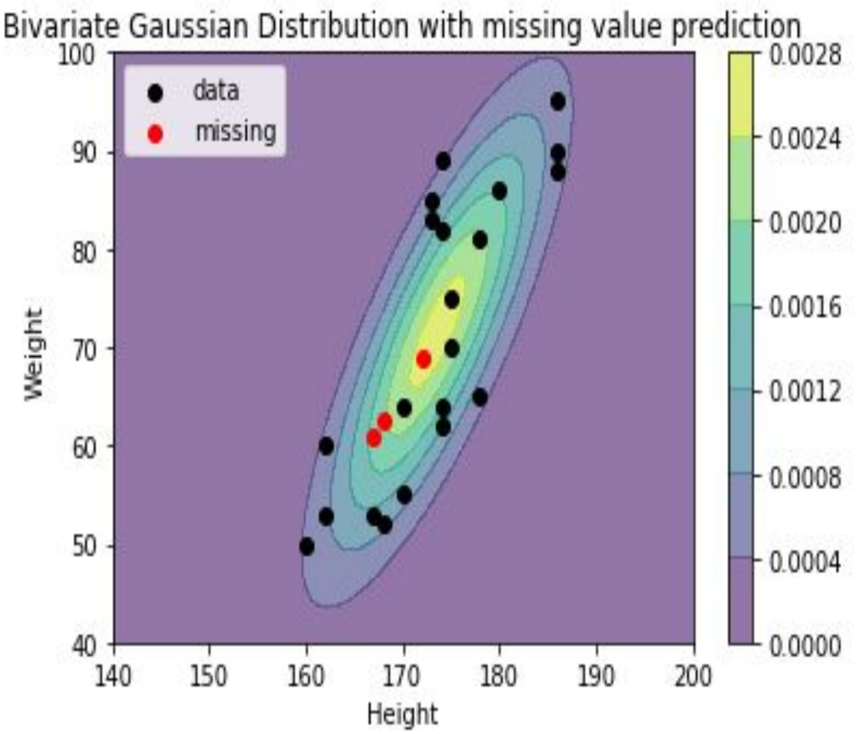
Thus the Conditional PDF:

$$f_{X,Y}(x, y) = ce^{-q(x,y)},$$

Where: $$c = \frac{1}{2\pi\sqrt{1-\rho^2}\,\sigma_X\sigma_Y}.$$ $$q(x, y) = \frac{\frac{x^2}{\sigma_X^2} - 2\rho\frac{xy}{\sigma_X\sigma_Y} + \frac{y^2}{\sigma_Y^2}}{2(1-\rho^2)}.$$

Missing Values

Weights	Heights
62.50	168.0
68.98	172.0
60.89	167.0



Exercise 3:
Bayesian Regression

Preprocessing Data:

- **Problem:** Fit a 5th order Polynomial using Bayesian Regression.
- Given Data:

$$\boxed{H = [h_1, h_2, ..., h_n]}$$

- Preprocessing for a 5th order Polynomial:

$$\boxed{X = [h'_1, h'_2, ..., h'_n], \text{ where } h'_i = [1, h_i, h_i^2, ..., h_i^5]^T}$$

- Standardization for numerical stability (used only for MLE):

$$\boxed{x = \frac{x - \mu_X}{\sigma_X}}$$

Fitting Weights:

- Preprocessed Input:

$$X = [h'_1, h'_2, ..., h'_n], \text{ where } h'_i = [1, h_i, h_i^2, ..., h_i^5]^T$$

- Given Weights: $W = [w_0, w_1, ..., w_5]$

- **MAP Update**:

$$W_{MAP} = (XX^T + \frac{\sigma^2}{\sigma_0^2} I)^{-1} Xy$$

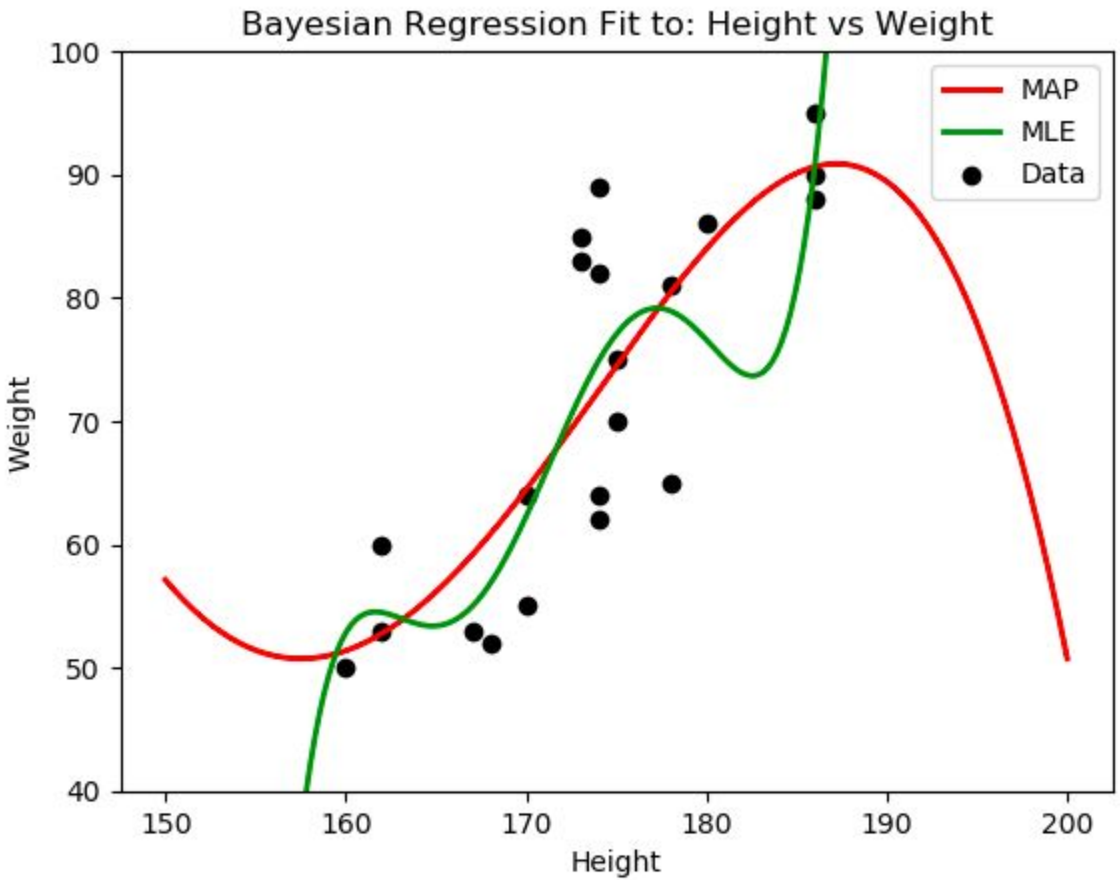
MLE Update:

$$W_{MLE} = (XX^T)^{-1} Xy$$

MAP vs MLE

Mean Squared Error Loss:

- MLE : 62.4830
- MAP: 69.8029



Exercise 4:
Boolean functions and the Boolean Fourier Transform

Problem Statement:

- Use least squares to approximate rules of a cellular automata by the multiplication of an vector with an matrix
- Two variants of matrices:
  - Usage of the matrix X specifying the rule (with entries in {-1, 1})
  - Usage of a feature matrix based on X

Simple Model fitting

Problem: find w* that minimizes :

$$\boxed{E[w] = ||Xw - y||^2}$$

Solution from lecture:

$$\boxed{w^* = (X^T X)^{-1} X^T y}$$

Results:

$$\boxed{\begin{aligned}
w^*_{110} &= (0.25, -0.25, -0.25)^T \\
\hat{y}_{110} &= (0.25, -0.25, -0.25, -0.75, 0.75, 0.25, 0.25, -0.25)^T \\
w^*_{126} &= (0.00, 0.00, 0.00)^T \\
\hat{y}_{126} &= (0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00)^T
\end{aligned}}$$

Model fitting using the feature Matrix

To increase the quality of our results we replace the matrix by the feature matrix, as seen on the left

where **φ** is defined as:

$$\boxed{\varphi_S(x_1, x_2, x_3) = \prod_{i \in S} x_i}$$

$$\boxed{\Phi = \begin{bmatrix} — & \varphi(X_1) & — \\ — & \varphi(X_2) & — \\ — & \varphi(X_3) & — \\ — & \varphi(X_4) & — \\ — & \varphi(X_5) & — \\ — & \varphi(X_6) & — \\ — & \varphi(X_7) & — \\ — & \varphi(X_8) & — \end{bmatrix}}$$

Final Results

$$\boxed{\begin{aligned} w^*_{110} &= (0.25, 0.25, -0.25, -0.25, -0.25, -0.25, -0.75, 0.25)^T \\ \hat{y}_{110} &= (-1, +1, +1, -1, +1, +1, +1, -1)^T \\ w^*_{126} &= (0.50, 0.00, 0.00, -0.50, 0.00, -0.50, -0.50, 0.00)^T \\ \hat{y}_{126} &= (-1, +1, +1, +1, +1, +1, +1, -1)^T \end{aligned}}$$

Exercise 5:

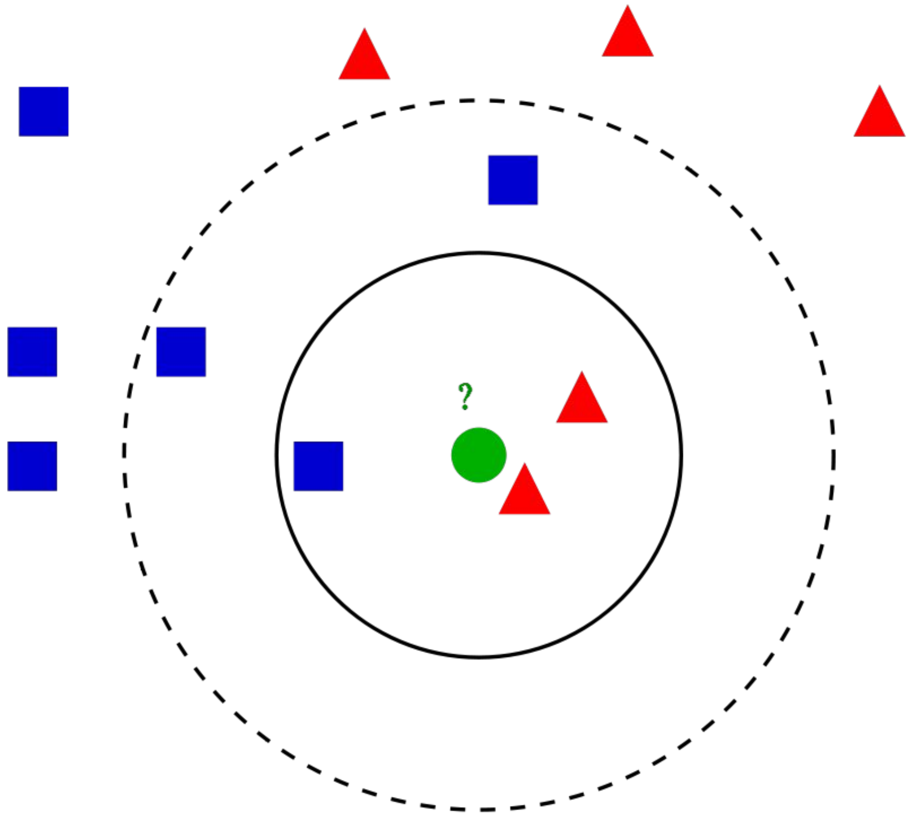
Nearest Neighbor Classifier

Problem Statement:

- Implement a function that realizes an n-nearest neighbor for $n \in \{1, 3, 5\}$.
- Show the percentage accuracy.
- Show the overall runtime for $n = 1$ for test data.

What is a nearest neighbor classifier?

It’s a method in which n closest points w.r.t a given test sample are searched in the training data and based on the majority, a class is assigned to it.



Method:

- Calculate Euclidean of the test data with all of the training data
- Find the n training points with smallest distances
- Take the majority of the labels from the neighbors

Broadcasting

- Calculation of distance involves subtracting the points and then taking the p-norm of the difference.
- How to calculate the difference between single test point and all of the training data ?
  1. Iterate over training data using for loop and subtract the test point from each training point
  2. Vectorise approach using Broadcasting

Broadcasting

It is process of making two arrays of dissimilar dimensions compatible with each other for performing arithmetic operations.

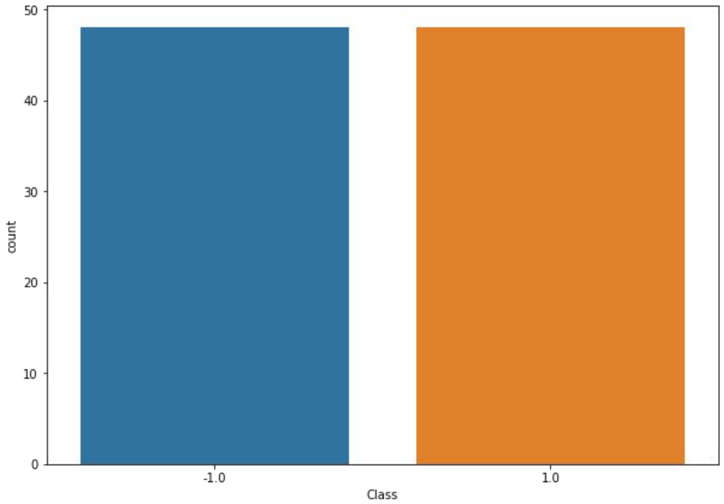
Example

|1 2 3| + |7 8 9| |4 5 6|	≡	|1 2 3| + |7 8 9| = |8 10 12| |4 5 6| |7 8 9| |11 13 15|

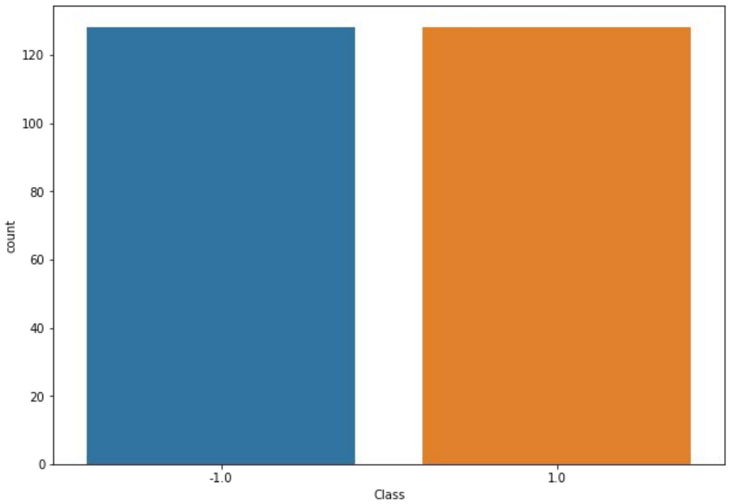
Now for subtraction of test data from training samples, a single line of code can be used with the in-built broadcasting of numpy

```
diff = X_train - data_point
```

Visualisations

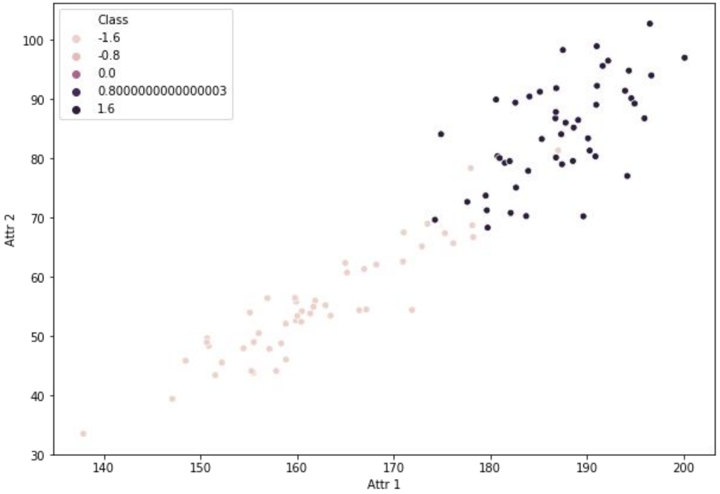


Test Data

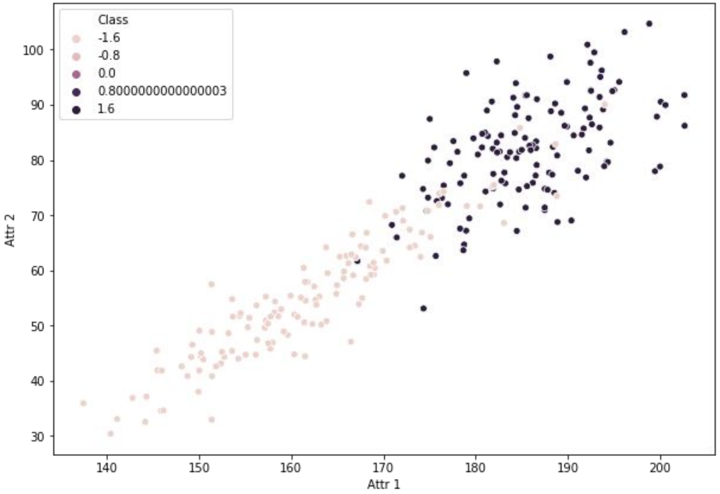


Training Data

Visualisations



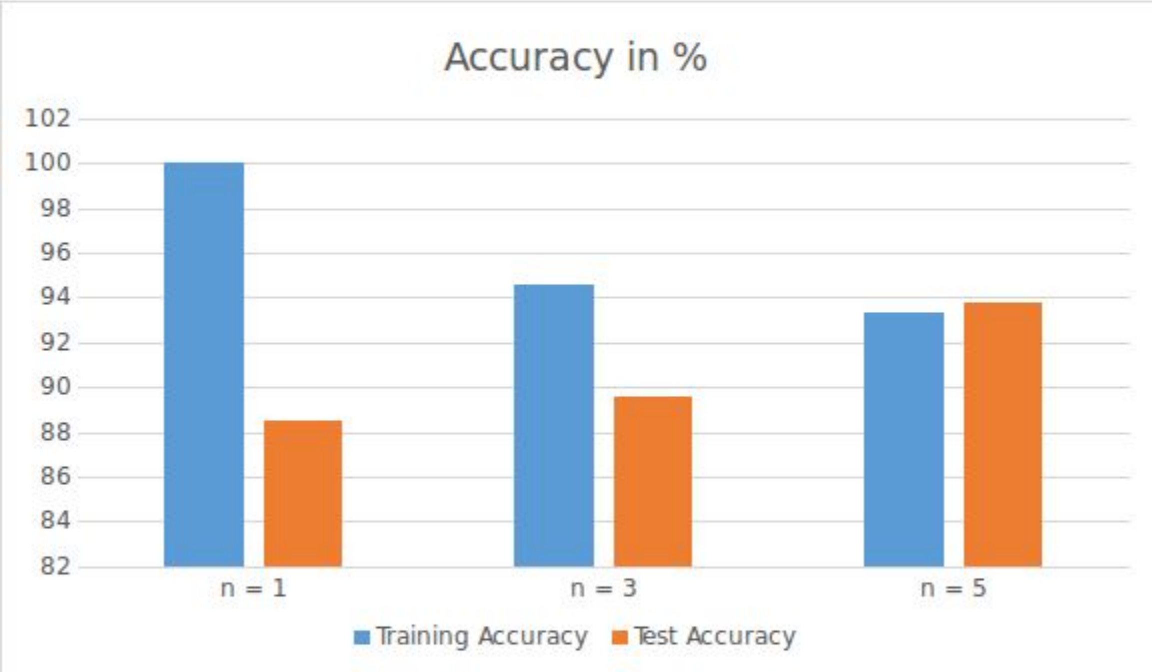
Test Data



Training Data

Results

Accuracy

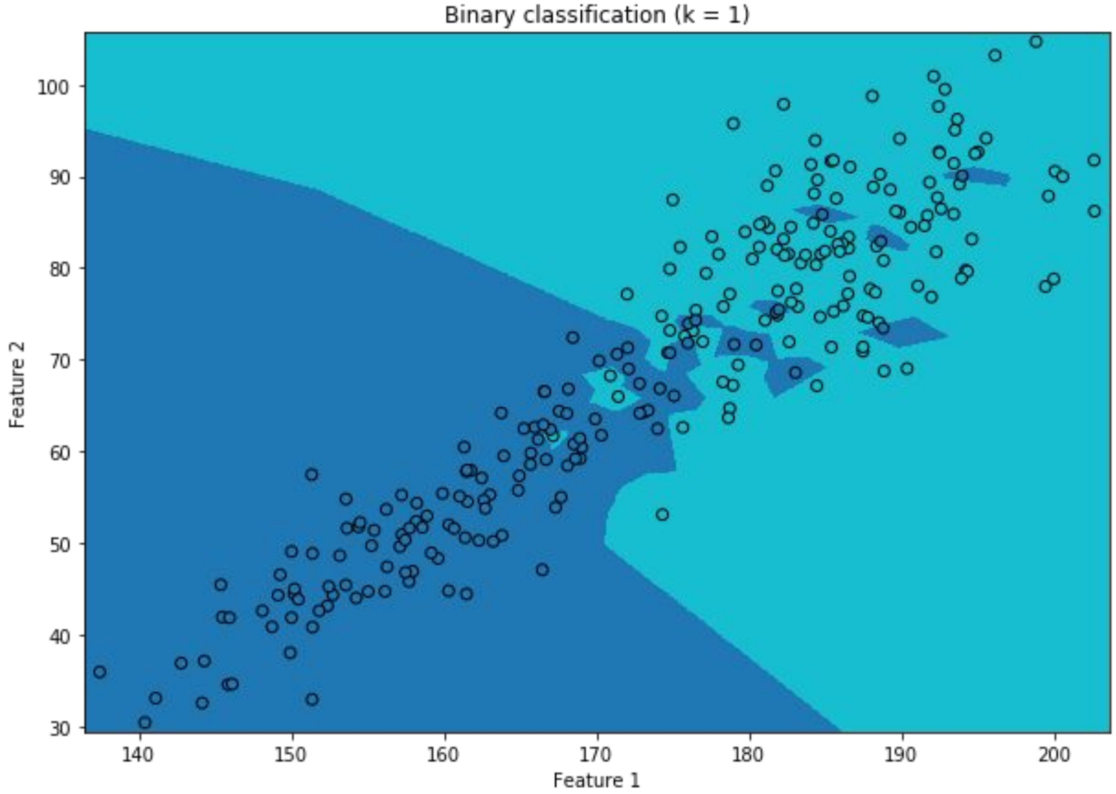


Results

Execution time

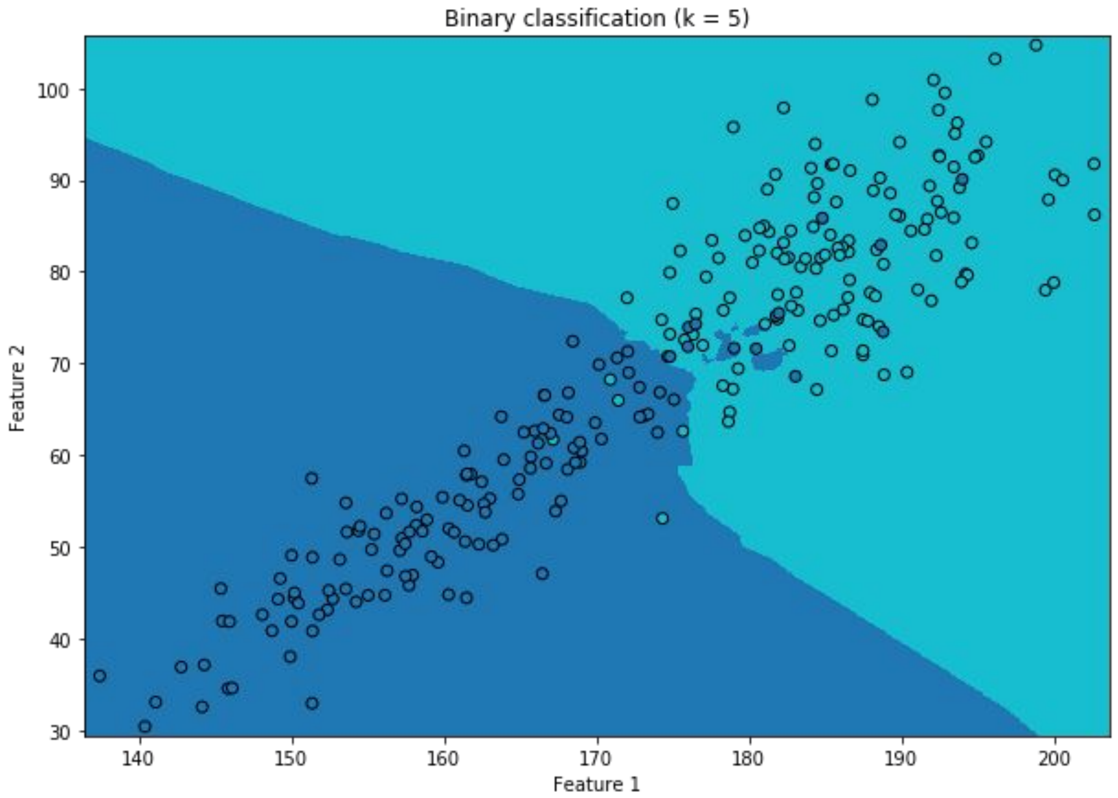
Time taken for computing the 1-nearest neighbor for the test data was 0.1654 seconds

Decision Boundary



Decision Boundary

Decision Boundary



Exercise 6:
k-Dimensional Tree
(k-D Tree)

k-D Trees

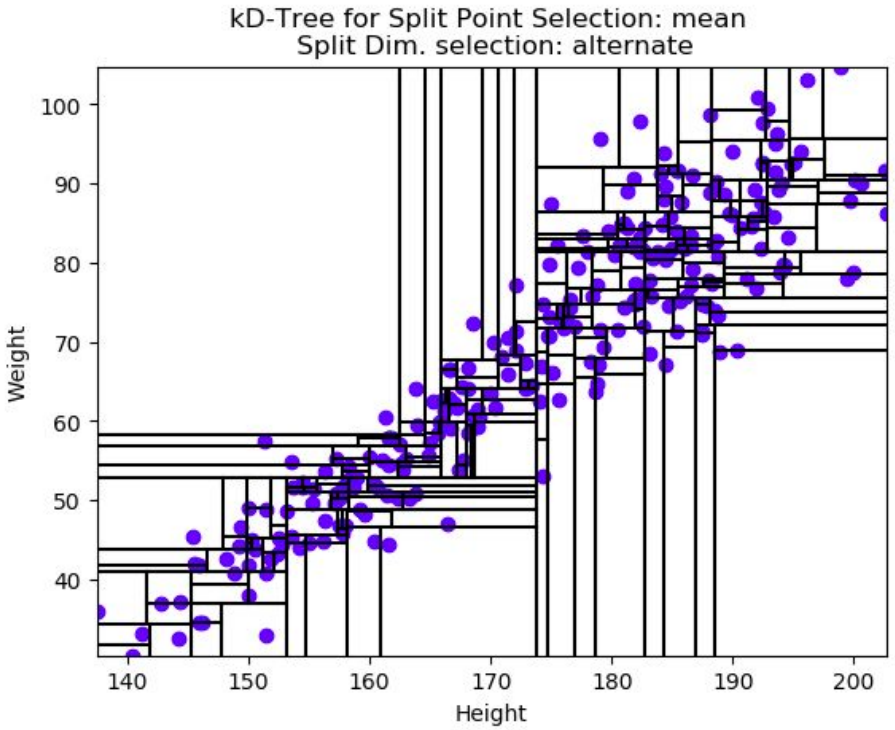
- **Def:** Space partitioning data-structure for organizing points in a k-dimensional space.
  - Data is split along a dimension at a predefined point (axis parallel lines)
- **Problem Statement:** 1-NN search on a 2-dimensional space.
  - Training : data2-train.dat
  - Testing : data2-test.dat

k-D Trees

- **Splitting Dimension**:
  - Alternate through dimensions
  - Choose the dimension of highest variance
- **Point of Split (Position of splitting plane)**:
  - The mean of the dimension values
  - The median of the dimension values

Combination of the above results in 4 different trees.

k-D Tree I (Mean: Alternate)

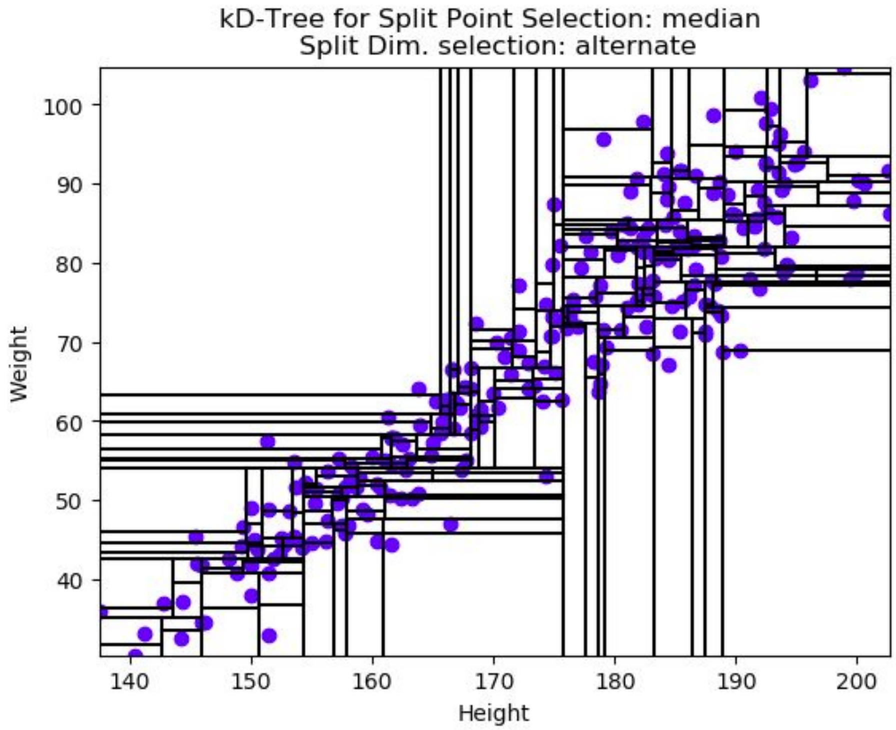


Statistics:

- Tree Depth = 10
- Accuracy on Test Set: 84.38%

- Creation Time : 0.005 secs
- Testing Time : 0.007 secs

k-D Tree II (Median: Alternate)

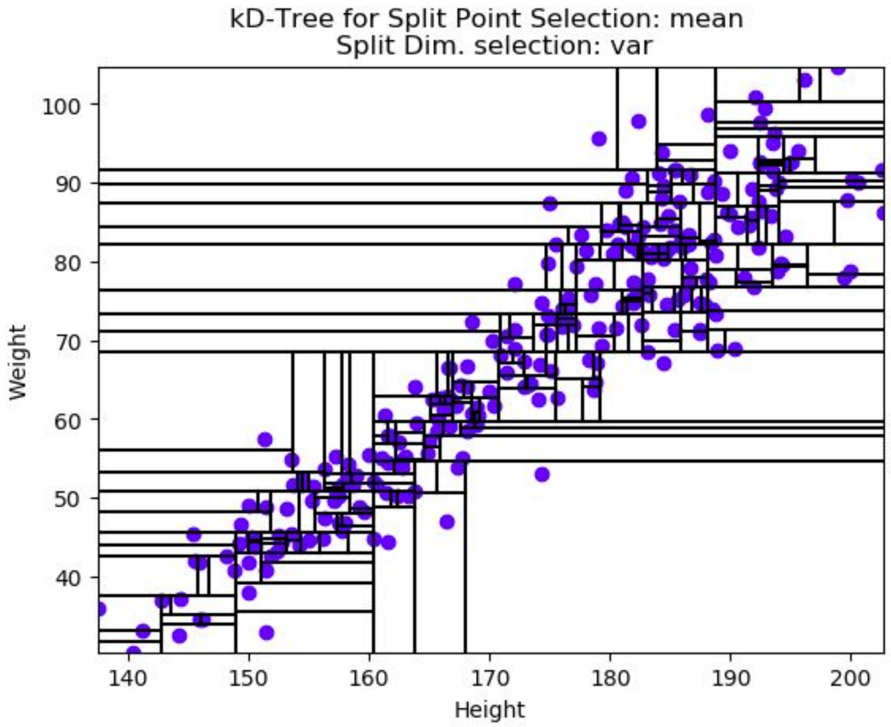


Statistics:

- Tree Depth = 8
- Accuracy on Test Set: 87.5%

- Creation Time : 0.014 secs
- Testing Time : 0.005 secs

k-D Tree III (Mean: Highest Variance)

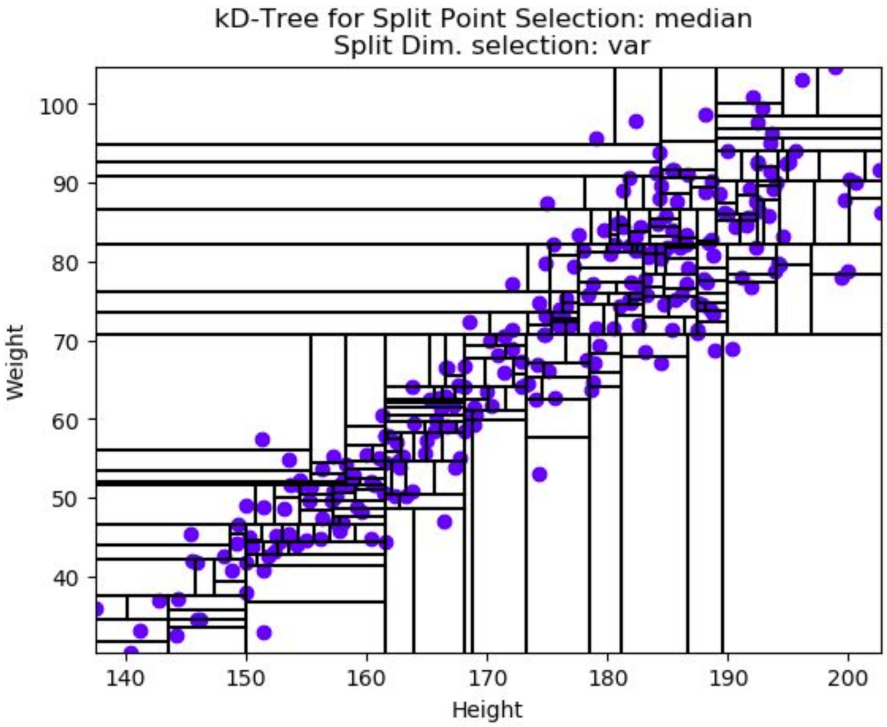


Statistics:

- Tree Depth = 10
- Accuracy on Test Set: 87.5%

- Creation Time : 0.012 secs
- Testing Time : 0.006 secs

k-D Tree IV (Median: Highest Variance)



Statistics:

- Tree Depth = 8
- Accuracy on Test Set: 88.54%

- Creation Time : 0.023 secs
- Testing Time : 0.005 secs

Performance of different k-D Trees

Sl.	k-D Tree Type	Test Acc.	Train Time	Search Time
1	Split = mean, Dim = alternate	84.37	0.005	0.007
2	Split = median, Dim = alternate	87.50	0.014	0.005
3	Split = mean, Dim = variance	87.50	0.014	0.007
4	**Split = median, Dim = variance**	**88.54**	0.025	0.005
5	**Sklearn kD Tree, leaf_size = 1**	**88.54**	**0.0002**	**0.0002**

- Highest Accuracy on Test Set: 88.54%
  - median based splitting with dimension of highest variance
  - Significantly faster than naive implementation (0.16s) search time
- Scikit-learn benchmarks:
  - Sklearn-significantly faster. But with equal classification accuracy

Thank You.