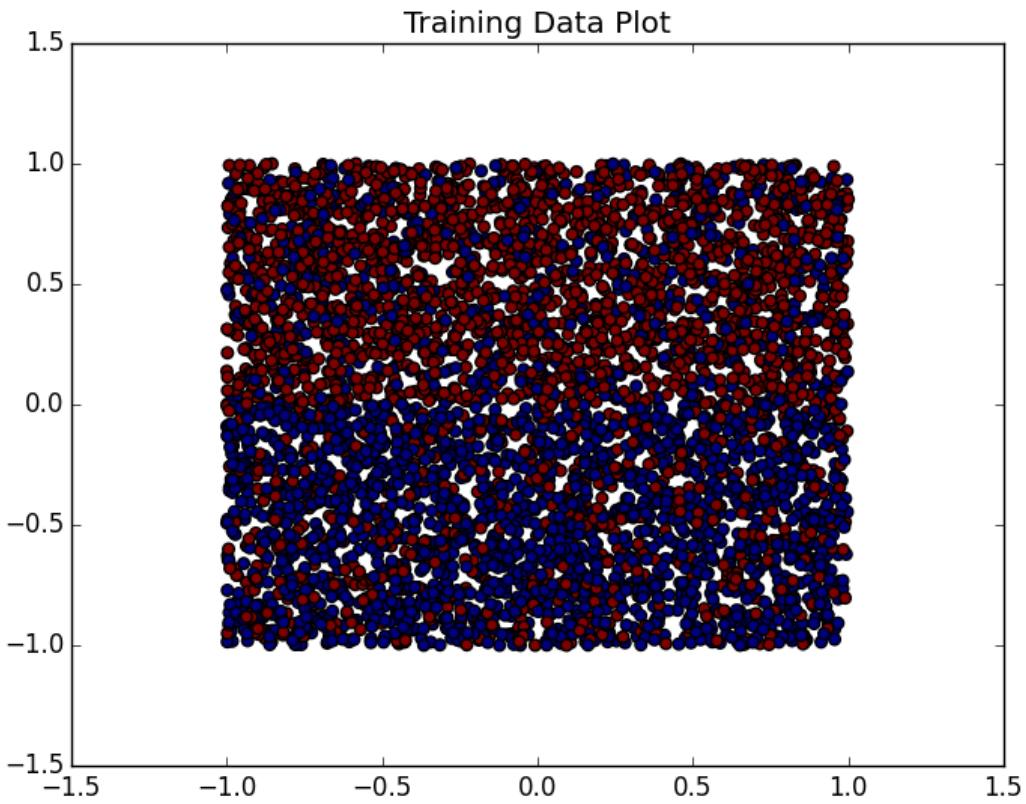
Yuan Bowei 1001916
Machine Learning
September 22, 2015

Machine Learning Assignment01

3. K-nearest Neighbours

b) Plot of the training data



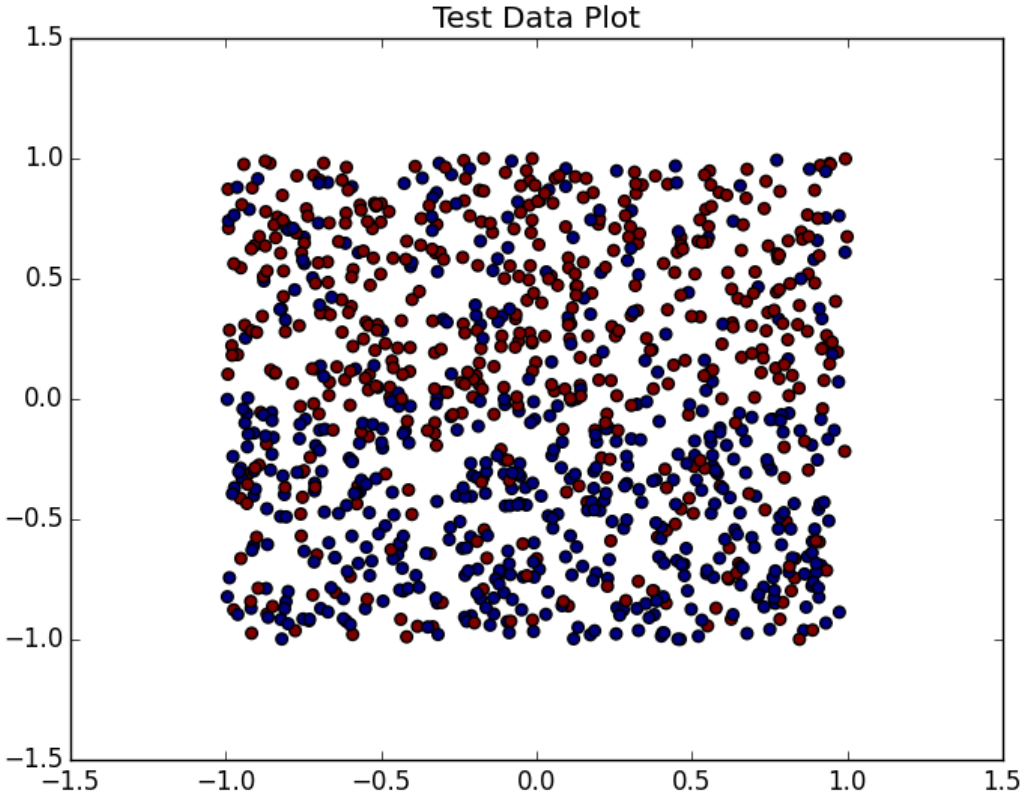
c) See code, kNN.py, the function is implemented with KDTree data structure in scipy

d) As mentioned in 3.c), the kNN algorithm is implemented with KDTree, which internally calculated the square root for the query method. But in this case the computation of square root is not necessary because of its two main usages in this problem:

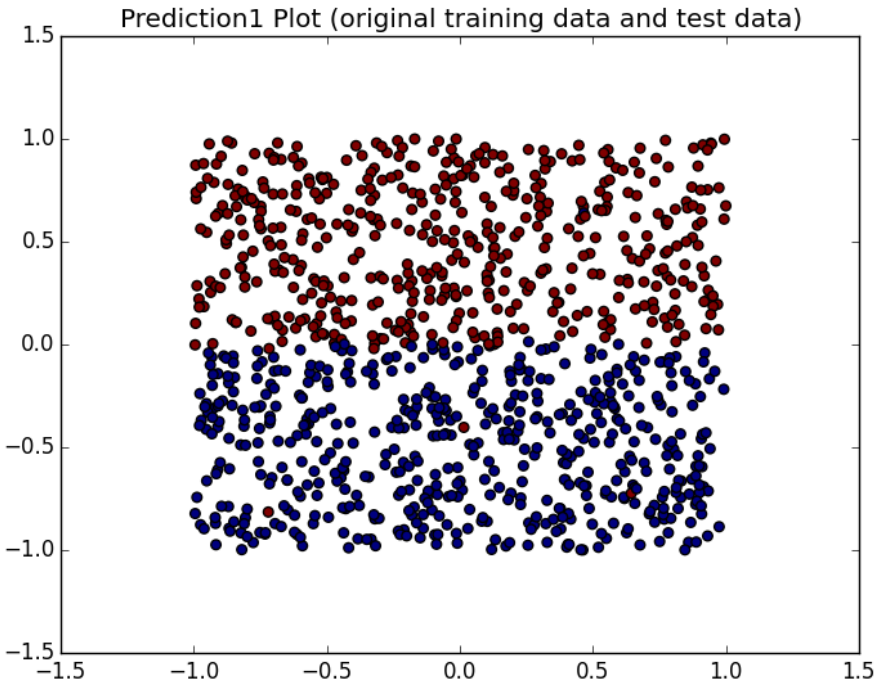
1) The distances are used to compare so that the k nearest neighbours can be selected. The bigger Euclidean metric is still bigger even after taking square root.

2) the inverse of the distance is used as a weight when predicting the label of the point with the labels of N nearest neighbours. This step won't be affected whether we take the square root or not.

e) Plot of the testing data:



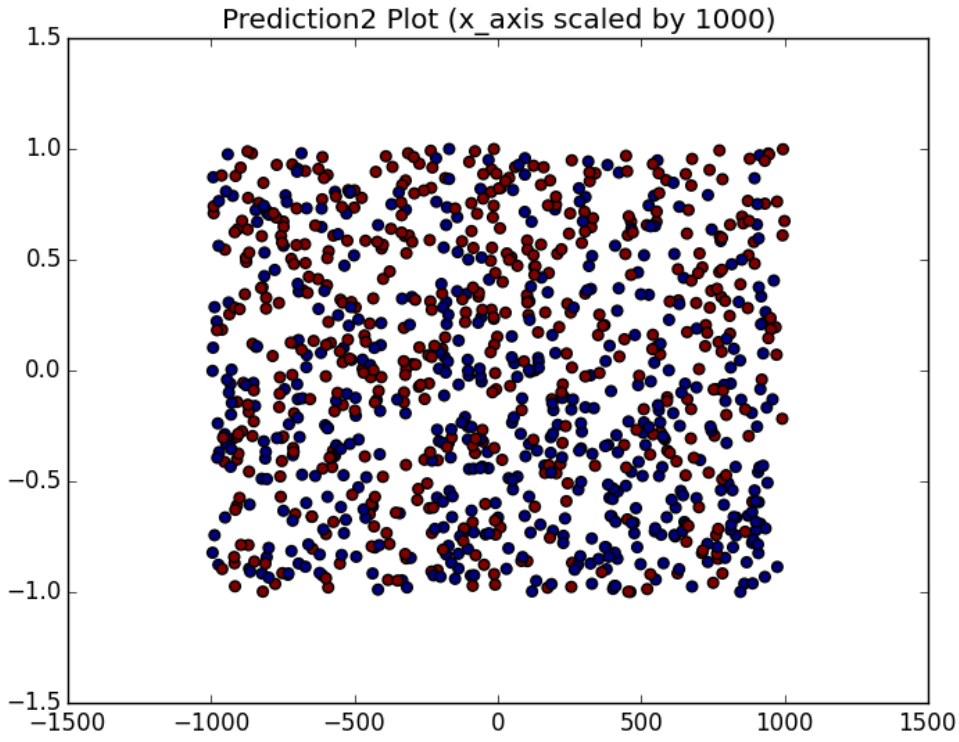
Plot of this Prediction:



correction rate: 0.74

=> 0-1 Error score: 0.26

f) Plot of prediction after scaling x-axis by 1000



correction rate: 0.565

=> 0-1 Error Score: 0.435

g) The error rate grew after scaling the x-axis by 1000

h)

The standard deviation of the unscaled training data:

(0.58143566246545297, 0.58234323276506317)

i) Standard deviation after the less general normalisation on original training data:

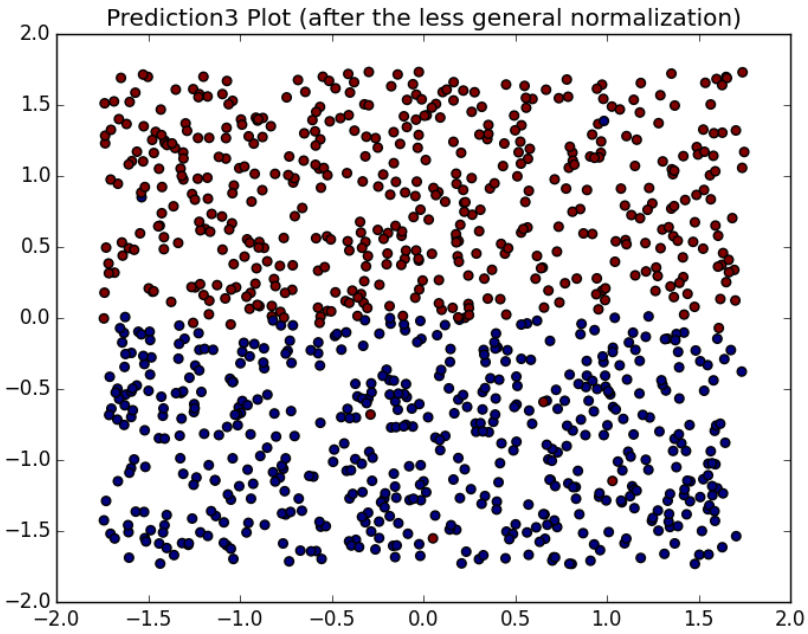
(1.0000000000000002, 0.99999999999999989)

Standard deviation after the less general normalisation on original testing data:

(1.0, 1.0)

Correction Rate now: 0.741

NOTE: applying the more general normalisation to x-scaled training data and testing data, the correction Rate bounces from 0.565 to 0.74, so we can see the effect of normalisation on dataset with large standard deviations.



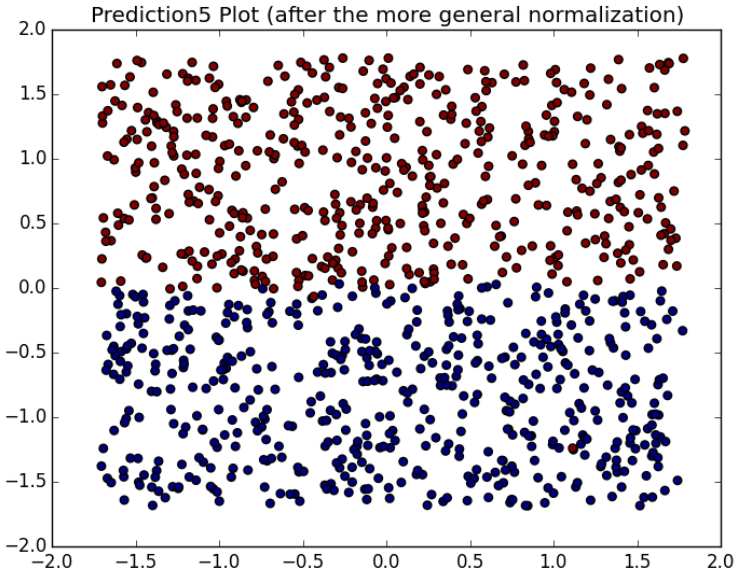
j) applying the more general normalisation to original training data and testing data:

Standard Deviation after this step:

Training Data: (1.0000000000000002, 0.9999999999999989)

Testing Data: (1.0, 1.0)

Plot of the prediction after applying more general normalisation to original data:

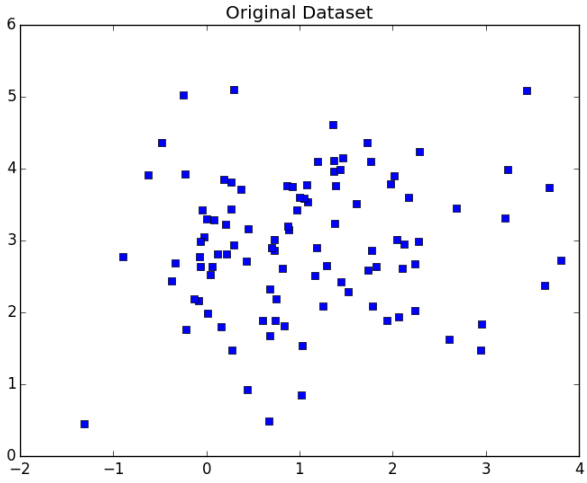


Correction Rate now: 0.741

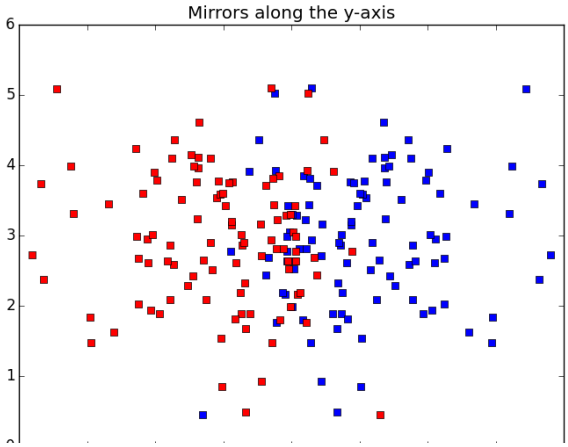
h) the revised euclidean metric
see code kNN.py

4. Linear Transformation

1) original dataset:

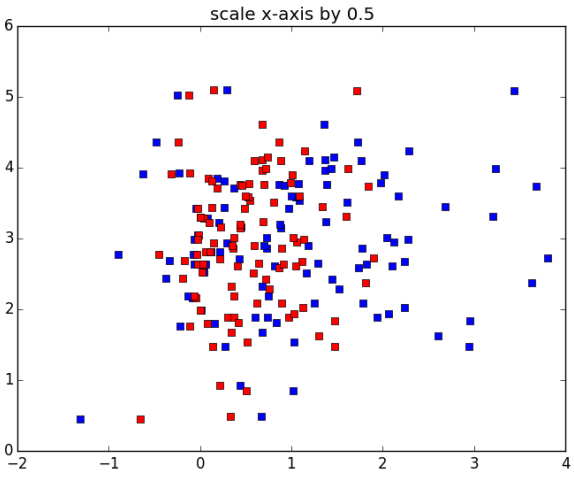


2) mirror along y-axis

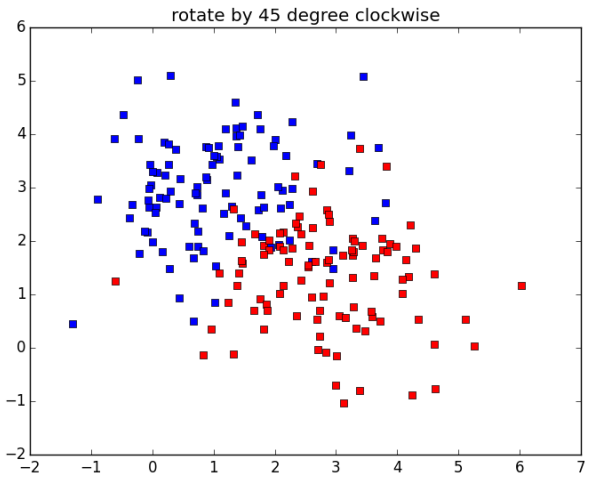


3) scale x-axis by 0.5

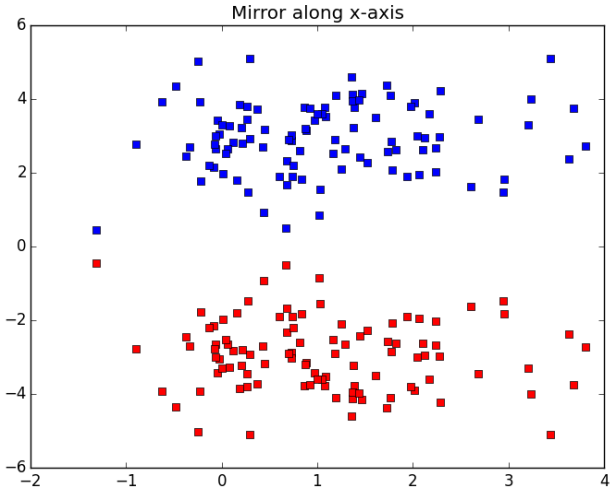
Notice the changes of x-axis



4) rotate by 45 degree clockwise



5)mirror along x-axis



6) after transformed by A5