1):

A):

Please notice: Since the data set is shuffled before each test, to avoid RNG, for each degree, the program take 10 times k-fold Cross Validation with re-shuffled input dataset, and get an average Mean Square Error.
So, instead of calling it Mean Square Error, I print it as Average Mean Square Error in my program, because it's an average of ten test. The dataset is shuffled before each test.

You can test the function under the homework folder: set verbose to 1 to see required plot.

~ python nfoldpolyfit.py 'linearreg.csv' '9' '10' '1'

B):

According to the program output:

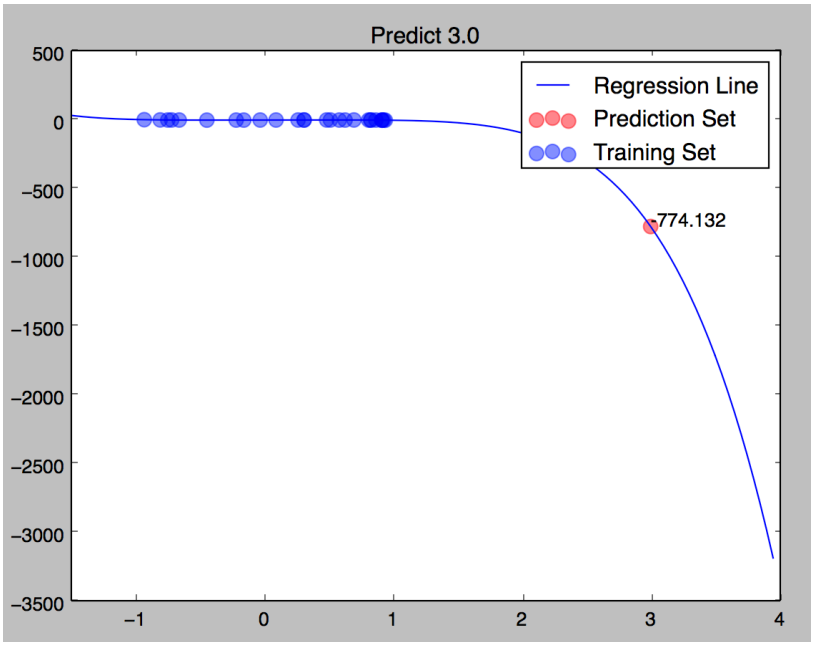
```
~ python nfoldpolyfit.py 'linearreg.csv' '9' '10' '1'
Averaged Mean Square Error for degree 0 10-fold cross validation is: 1.445481
Averaged Mean Square Error for degree 1 10-fold cross validation is: 1.214868
Averaged Mean Square Error for degree 2 10-fold cross validation is: 1.397174
Averaged Mean Square Error for degree 3 10-fold cross validation is: 0.317199
Averaged Mean Square Error for degree 4 10-fold cross validation is: 0.360155
Averaged Mean Square Error for degree 5 10-fold cross validation is: 0.209757
Averaged Mean Square Error for degree 6 10-fold cross validation is: 0.556949
Averaged Mean Square Error for degree 7 10-fold cross validation is: 0.299451
Averaged Mean Square Error for degree 8 10-fold cross validation is: 1.819683
Averaged Mean Square Error for degree 9 10-fold cross validation is: 2.287159
Degree 5 is the best, which has averaged Mean Square Error of: 0.209757
```

k = 5, which means 5th polynomial regression, yielded the best result, in average. This result is based on 10-fold cross validation.

Please run the program with verbose='1' to see the required plot. Please close the first graph to see the second graph.

C):

Input 3.0 was predicted as -774.132. Using the result from 5th polynomial regression.



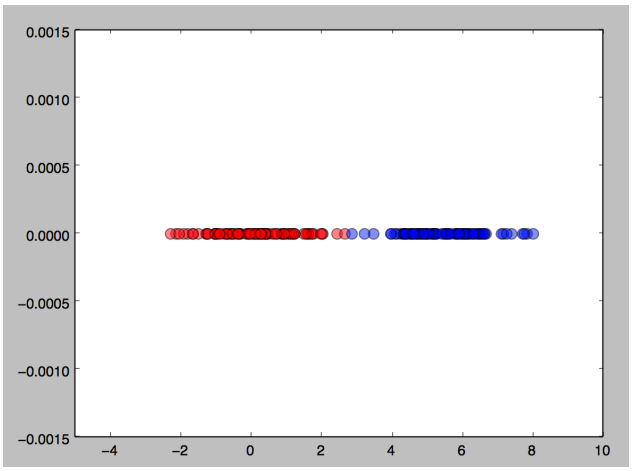
I don't think this is a reasonable prediction, considering it's way too far from the other point, which all stay within range [-1.5, 3]. (they were pushed to a line on the chart because the scale is heavily enlarged by regression line)

The reason might caused because: We don't have any training data nearby 3, if we want to predict 3 to a more reasonable answer, we might try to add some input data nearby 3 to get a better regression line.

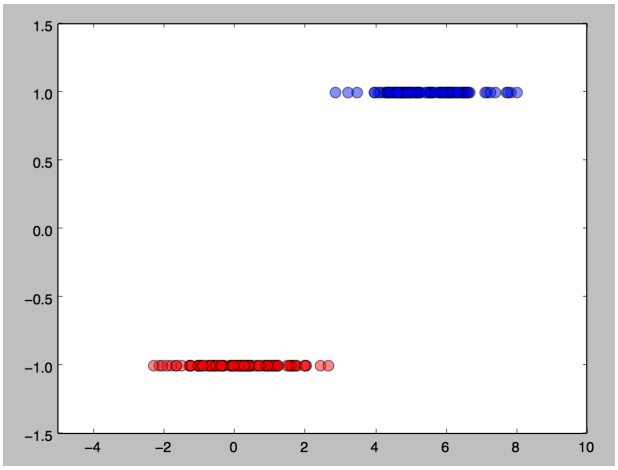
2):

A):

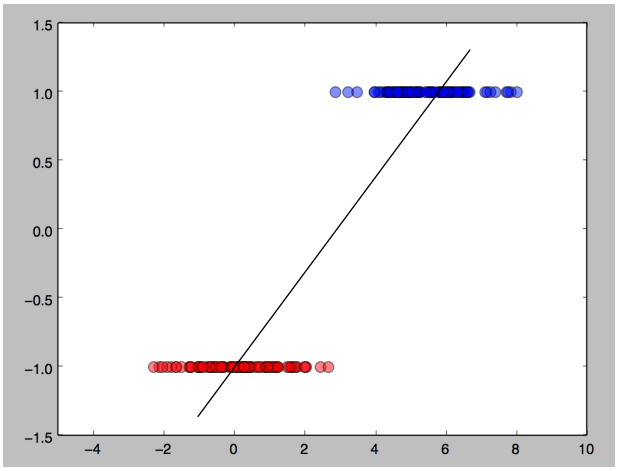
0: We have a lot of training data, they have only one attribute for us to consider.



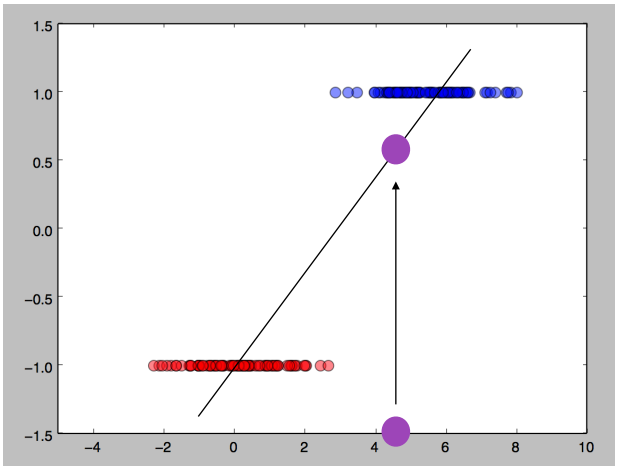
1: Label each class by a number, we use 1.0 and -1.0.



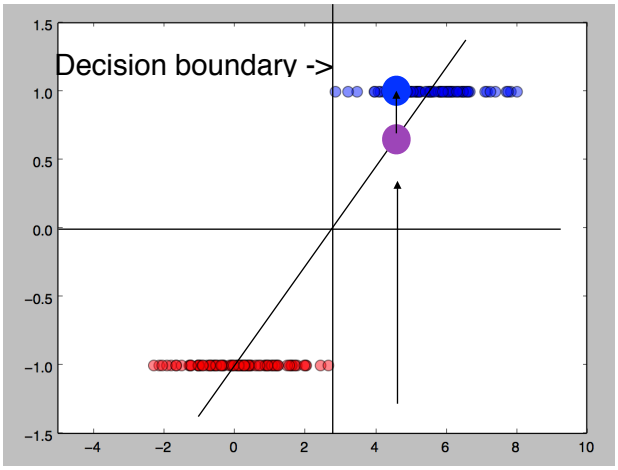
2: Using attributes in input x, and the number labeled according to their classification, draw a regression line.



3: Using this regression line to calculate a prediction value of incoming data point



4: label this point to the label number nearest to the prediction value. (blue)



This could be easier if we calculate a decision boundary after we get the regression line, and use that as threshold.

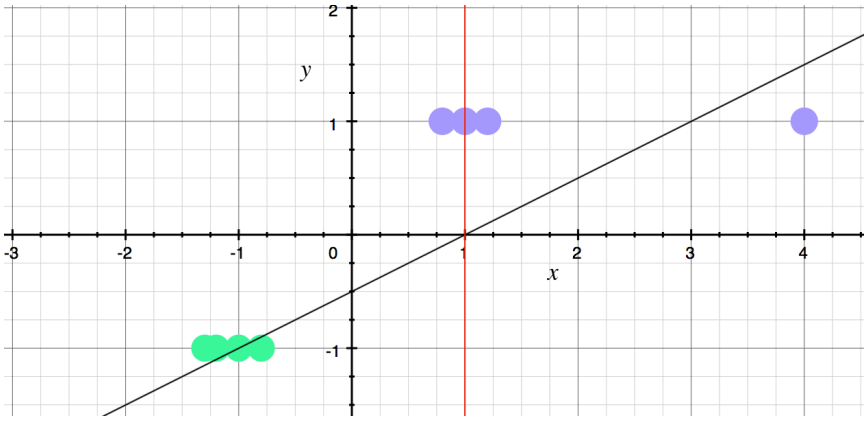
2):

B):

1): Some time, the regression line can be dragged away by the training data.

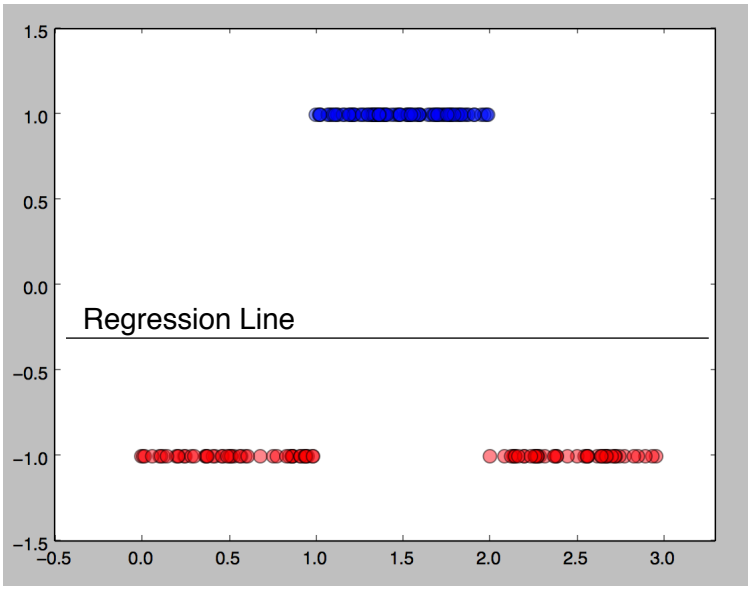
And the decision boundary we made according to that regression line is no longer valid.

E.g.
One purple points on the chart will be misclassified as green point. (picture created using mac grapher)



2): Some case is even worse, this method by its nature can't handle dataset that's not linearly separable, since it's using boundary line or boundary hyper-plain to separate dataset.

E.g. Boundary can't be found on the chart not, even we can found it outside the chart, it's meaningless.



3):

A):

Regression method performing pair-wise on each class pair when class is bigger than 3. This may works fine (at least not obviously worse than LDA) when the data set looks like this (picture from ESLII)

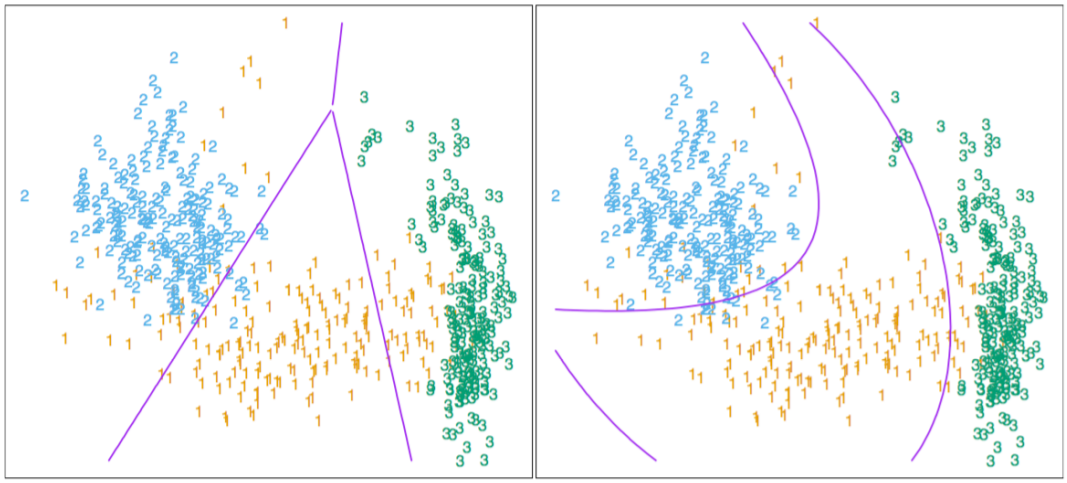


FIGURE 4.1. *The left plot shows some data from three classes, with linear decision boundaries found by linear discriminant analysis. The right plot shows quadratic decision boundaries. These were obtained by finding linear boundaries in the five-dimensional space* $X_1, X_2, X_1X_2, X_1^2, X_2^2$. *Linear inequalities in this space are quadratic inequalities in the original space.*

But when the means of each class tend to line up on a straight line, regression method won't work for this. (N convex decision regions method can avoid this, but not using pair-wise method can not deal with classes that are not singly connected convex regions, as described in Lecture slide 6, page 29~30)

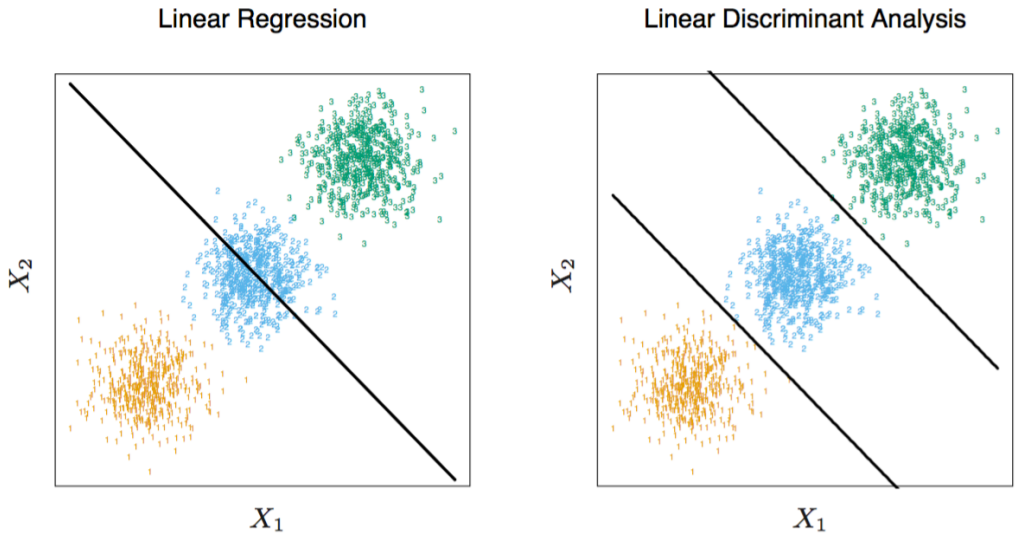


FIGURE 4.2. *The data come from three classes in* $\mathbb{R}^2$ *and are easily separated by linear decision boundaries. The right plot shows the boundaries found by linear discriminant analysis. The left plot shows the boundaries found by linear regression of the indicator response variables. The middle class is completely masked (never dominates).*

B):
LDA assume:

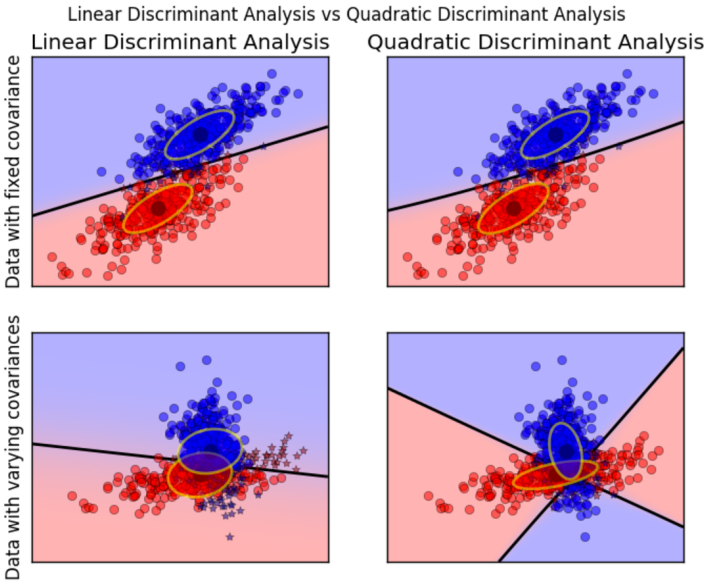
Each classes of data is normally distributed with a common covariance.

C):
QDA assume:

Each classes of data is normally distributed, but unlike LDA, QDA do not assume the covariance of each class is identical.

Pros: more flexible (reference from http://scikit-learn.org/stable/modules/lda_qda.html)

As we can see from the chart on the right, QDA shows better result than LDA when the covariance of input datas on each class are naturally difference. While LDA just tried to create a more "round" covariance and the result is less reasonable than QDA. (But, we can map the data to higher dimension in LDA to get a curve boundary line, as described in problem 3-d.)



Cons: QDA requires more resource (time, memory) for more complex computation (reference from http://rbras.org.br/lib/exe/fetch.php/projetos:machlearn:comparisonofclassifiers.pdf), in the situation that input data are naturally have same covariance, QDA and LDA tend to produce same result, so we may want to use LDA to avoid using extra computational resource.

D): Similar to applying linear regression to find polynomial regression line, we can "map" polynomial dimension using the features on our hand, for example we have X_1, X_2, we can map them to 2d dimension to create features like X_1X_2, X_1^2, X_2^2 to our input, this will resulting a more reasonable output like QDA.

(picture reference from ESLII, they have an example for this)

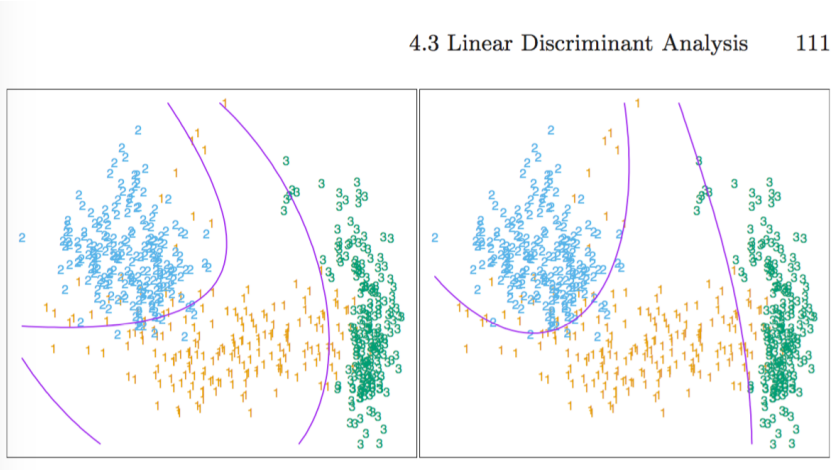


FIGURE 4.6. *Two methods for fitting quadratic boundaries. The left plot shows the quadratic decision boundaries for the data in Figure 4.1 (obtained using LDA in the five-dimensional space $X_1, X_2, X_1X_2, X_1^2, X_2^2$). The right plot shows the quadratic decision boundaries found by QDA. The differences are small, as is usually the case.*

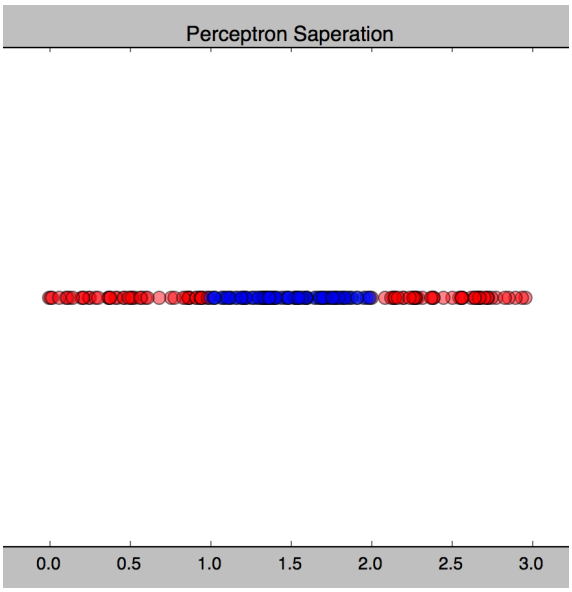
4):

A): Please run following command on my homework directory to test the program.

~ python perceptrona.py “linearclass.csv"
the final weights is [-11. 4.01152], founded on 1012th iterations

There will also be a chart pop out after you see the weights output.

Please notice weights are showing as a list, each element on the list corresponding to w_0, w_1, etc.



B):

The chart on the right shows how the dataset [X_2, Y_2] looks like.

When applied to this dataset, the program will fall to dead circle because the data is not linearly separable. The program will keep looking for that separation point forever.

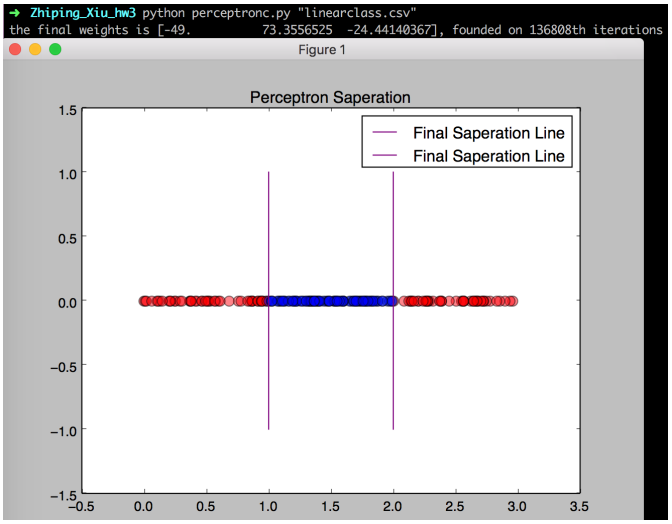
There are no such boundary can separate those data set.

C):
We can transform the input data to 2-order polynomial.

For example, our input f(X_2) = Y_2 can thus be transformed to f(X_2, X_2^2) = Y_2.

So, I created a new column X_2^2, using the value of X_2. And extend weight list to fit 2-order polynomial. After we found the valid w who can correctly separate all the input result, we can use np.roots() to find the boundary line. And then we can see the output showing on the right chart.

(Since this is 1-d input, it’s more accurate to use the phrase boundary point instead of boundary line. But we use line to visualize the data, because it’s more visible, just remember we are still in 1-d space.)



If you want to test this function yourself:

~ python perceptronc.py “linearclass.csv"

You can change the code to let it do n-order polynomial calculation for 1-d input by changing m_init, I comment the method in the code. (Since we are not allowed to change the input form, you have to change the code manually.)