Jeremy Midvidy, jam658
EECS 349, Fall 2017
Assignment 3

---

1)
A. Program

I chose to use 5 folds. It seems like a good way of cross-validating data with 30 elements, so it could be split into 5 groups of 6 elements each.

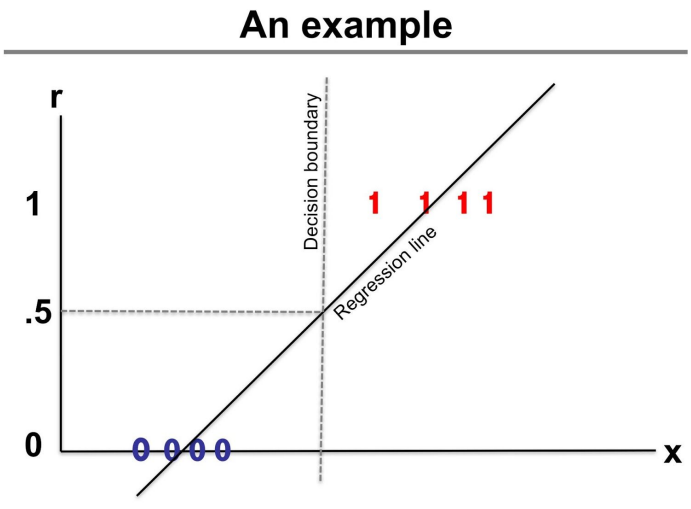
B. The value of k that yielded the best results was k = 5. According to my graph, this was the most accurate prediction (MSE = .5). And also logically, this makes sense as well. With a univariate instance vector, polynomials with large degrees would overfit compared to those closer to the 4-5 range, which is just a couple degrees of greater tuning than the attribute vectors themselves.

C. For X = 3, my prediction is Y = -7.5. I think this is an accurate prediction given my algorithms performance during cross validation, however, given the limited nature of the data set (most X values are between -1 and 1), this prediction stems from my algorithms continuing the trends of the data set. There are a lot of functions that change drastically when leaving the domain of {-1, 1} so it is entirely possible that this prediction doesn't fit the actual model my algorithm is trying to predict, but only fit the my algorithm's prediction of the model itself.

2)

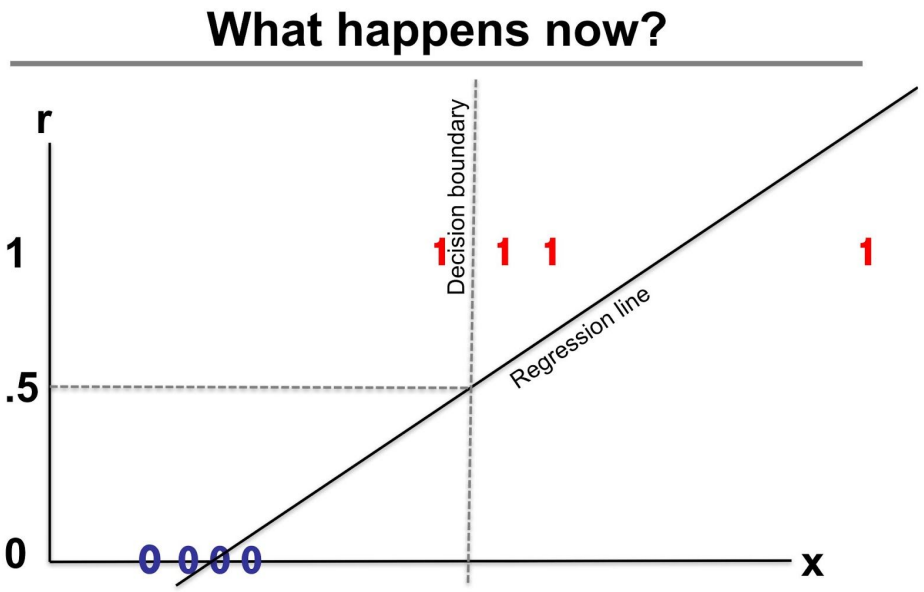
A. Classification via regression is a simple, algorithmic process. The ultimate goal of classification via regression is to be able to predict, given some input, the class of input, and appropriately label each input as given.

The algorithm trains on a set of inputs with known class labels. From the training set data, the algorithm creates a set of means squared errors (SSE's) that form a classification line. If the input is on either side of the regression line, the input will then be classified accordingly to which side of the line it is on (whichever is the nearest label number.)



In the picture above, the regression line has been formed, and inputs will be classified depending on which side of the regression line they correspond to.

B. One weakness with classification via regression is that it is a very rigid method of classifying data. If one of the points were mislabeled, even by a small amount, then the regression line that would ultimately fit upon the data would misrepresent how future inputs should be classified.



In this example above, one of the '1' labels are misclassified. Even though this is just one data point, it completely throws off how the regression will classify future inputs.

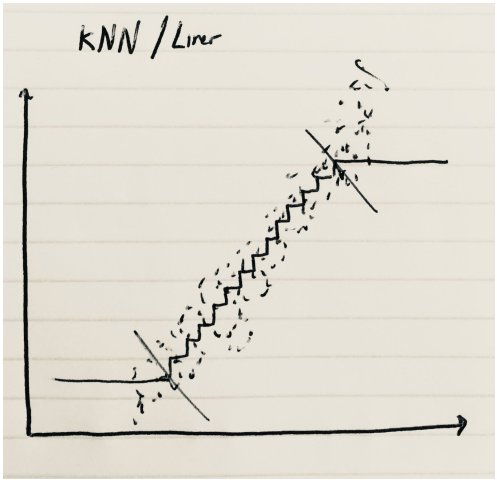
For this reason, one weakness of classification via regression is that it requires a very strict classification of training inputs in order to accurately predict future inputs.

3)

A. For some data set, you can set the parameters for polynomial regressions and KKN regression to both classify via the **average** of each data cluster, rather than the MSE for the classification via regression. This works because KKN clusters the data and then uses polynomial regression to classify future data sets off the value and the center-most (average) of the data cluster. If we code the classification via regression model to only build polynomial sets from the value of the average of the inputs (and not the SSE), then both approaches will output the same value for f(x), regardless of the input value. This would ensure that the linear

regression model uses what the KNN model would use as it's shortest distance to create its regression line.

B. One example where KNN output would be similar to that of a linear regression on the training data would be a step function. If both the KNN and the linear regression trained on the data between the two vertical lines, the KNN and the linear regression would produce similar results (represented by the scattered dots in the graph.) Yet, on the much larger range of the testing data, the linear regression model would behave differently. The linear regression would continue a straight line, while the KNN would adjust to the new nearest neighbors on the vertical line. In this case, the KNN would behave more accurately than the linear regression.



C. I would pick polynomial regression because the function pictured looks to behave like a cos(x) curve. Given the info that the range for values (x) is representative of the range for values that will be applied to the function, it looks like a polynomial regression would fit the data. I would not use a KNN because it looks like the function follows a predictable, modelable, line, so trying to classify inputs via distance to their nearest neighbors already given on the data set does not seem like the best way of classifying data for this data set.

D. I would pick a value of 3 for K. This function looks like some sort of sin/cos function with a coefficient and a dampener inside the function, so considering that the function is a first degree function, a polynomial of degree 3 would fit data points nicely and not overfit the data as well.

4)

A. Program

B. The perceptron algorithm did not converge. It went on forever and eventually produced a heap overflow error. The algorithm did not converge because the data set was not linearly separable (thus it could not produce one unique set of weights that correctly identified the data set.)

C. We can transform the data set so it works by ensuring that it is linearly separable. All that needs to be done is delete whichever X values occur more than once. That way, if an X value is inputted, we can still predict its value based of all the data in the set (except for those deleted.) It might be a little less accurate, but with enough unique data, it should still be useful in modeling output.

For this example, I altered the code to remove all instances of X (in X2) that were within .01961 of each other. This ensured that the data did not contain X values that were extremely similar. If the data did have X values within this distance, and those values were each classified differently, it would have been way too hard for the perceptron to figure out a weight vector that correctly classified every input considering the extremely close nature of some inputs which had different classifications. I removed every instance of a variable that had was too close to another values (both the instance and the other value that was too close). In this data set, there were 6 instances where x-values were within .01961 of each other, so the X vector passed to the perceptona function was of length 194 and not 200.

The code for this alteration looked like this:

```
indicies = []
 for x in range(0, len(X2)):
     for y in range(0, len(X2)):
         if((np.abs(X2[x] - X2[y]) < .01961) and x != y):
             indicies.append(x)
             break

 cleanX2 = []
 for i in range(0, len(X2)):
     if (i in indicies):
         continue
     else:
         cleanX2.append(X2[i])

 cleanY2 = []
 for i in range(0, len(Y2)):
     if (i in indicies):
         continue
     else:
         cleanY2.append(Y2[i])
 for row in cleanX2:
     a = [1 , row]
     xTwo.append(a)
```

This might affect the overall accuracy of the perceptron function, but with a large number of data sets, it should still be able to correctly classify an input variable considering the perceptron was operated on by 194 data points, which is nearly the original 200.