James Baker

11/21/17

CMSC478-01

[bjames6@umbc.edu](mailto:bjames6@umbc.edu)

**Challenge Problem 1 Model Description**

To create a prediction model for the given training data, I decided to use multiple linear regression to predict x and y values based on sensor data. Using these predicted x and y we can calculate an r distance from the origin. The data given was not “regression-ready” due to the large amount of “NA” data prevalent in every row of the training data. To work around that I figured that there must be a minimum number of sensors viewing the object for each row of data. After a manual inspection of the data I found that every row in the 7 path\_data.csv files has at least 3 sensors viewing the object.

Using this information, I decided to find a way to find the three closest sensors to the object then put their distances as well as the names of those sensors into the regression function. I wrote an R function that parses through the path data frame and finds the minimum n sensor values (for n in between 1 and 3) and their labels and puts them into a new data frame called path.reg including the n closest sensors (close1, closeN) and their labels (cLabel1, cLabelN), as well as the given x and y values to train the model. It then returns this new data frame, so it may be called whatever the user wants. It takes in a path data frame (organized roughly the same way as the training data, sensors in columns 2:14 and x and y in columns 15 and 16) and an integer n (which is the minimum number of sensors that produce a number each row). In the new data frame, the n closest distances are stored as numeric type, and their labels are stored as factors using the sensor names.

I concatenated all 7 path data frames into one then put it through my function xyfit\_from\_path(path, n). I stored it in an arbitrary data frame called myPaths.reg. After the “regression-ready” data frame is returned, I check to see what n is used, then create a linear model for x and y using the closest distances and their respective labels. I played with using interaction terms between the n close distances and their labels as well as just including them in the regression separately. On the next page I show some tradeoffs between number of coefficients in the model and Adj. R2 for different selections of n and inclusion of interaction terms and the terms by themselves.

I found that the best tradeoff for number of coefficients was n = 2 and if I did not include interaction terms. It had a model with 27 coefficients and 25 of them are meant for dummy variables (the dummy variables representing which two sensors were the 1st and 2nd closest). It also had an Adj. R2 of .87 for x and .89 for y, which was still close to what they would be if n = 3. I then tested my models for x and y using 10-fold cross-validation and got a test MSE of 3.57 for x and 3.88 for y (ignoring any bias). 5-fold cross validation produced pretty much the same results.

My model can be adjusted to improve accuracy or reduce the number of inputs in the regression function just by adjusting the value of n. The script just needs to run in the proper working directory and it will produce the linear models for x and y and show their test MSE.

lm.fitx = lm(x ~ close1\*cLabel1+close2\*cLabel2+close3\*cLabel3, data=myPaths.reg)

lm.fity = lm(y ~ close1\*cLabel1+close2\*cLabel2+close3\*cLabel3, data=myPaths.reg)

**74 coefficients, Adj. R2 was .93 for y and .93 for x**

lm.fitx = lm(x ~ close1++close2+close3+close1:cLabel1+close2:cLabel2+close3:cLabel3, data=myPaths.reg)

lm.fity = lm(y ~ close1++close2+close3+close1:cLabel1+close2:cLabel2+close3:cLabel3, data=myPaths.reg)

**39 coefficients, Adj. R2 was .88 for x and .87 for y**

lm.fitx = lm(x ~ close1+close2+close3+cLabel1+cLabel2+cLabel3, data=myPaths.reg)

lm.fity = lm(y ~ close1+close2+close3+cLabel1+cLabel2+cLabel3, data=myPaths.reg)

**39 coefficients, Adj. R2 was .92 for x and .89 for y**

lm.fitx = lm(x ~ close1\*cLabel1+close2\*cLabel2, data=myPaths.reg)

lm.fity = lm(y ~ close1\*cLabel1+close2\*cLabel2, data=myPaths.reg)

**51 coefficients, Adj. R2 was .90 for x and .90 for y**

lm.fitx = lm(x ~ close1+cLabel1+close2+cLabel2, data=myPaths.reg)

lm.fity = lm(y ~ close1+cLabel1+close2+cLabel2, data=myPaths.reg)

**27 coefficients, Adj. R2 was .87 for x and .89 for y**

lm.fitx = lm(x ~ close1\*cLabel1, data=myPaths.reg)

lm.fity = lm(y ~ close1\*cLabel1, data=myPaths.reg)

**26 coefficients, Adj. R2 was .81 for x and .83 for y**

lm.fitx = lm(x ~ close1+cLabel1, data=myPaths.reg)

lm.fity = lm(y ~ close1+cLabel1, data=myPaths.reg)

**14 coefficients, Adj. R2 was .76 for x and .8 for y**