Alvaro Hu

September 17, 2018

Springboard

Machine Learning in Capstone 1

In this section, I use Machine Learning to predict the salaries of unseen data using the train, test, split method. To begin, I separated out the target variable, the salary, from the rest of the features. I then decided to use the information I learned from the previous section (Inferential Statistics) to choose the features I would use for my regressions.

My first model was an Ordinary Least Squares Fit with only 1 variable. I decided to do this because of my conclusion that there was only 1 Principal Component in my data. I chose the points per game variable because, from my Exploratory Data Analysis, I saw that it had the strong Pearson Correlation Coefficient of all the variables. The results from this very basic model was a low R-Squared value of 0.406 and a very high Mean-Squared-Error (MSE) of 40 **Trillion**. Yes, trillion. Although this is very high, it is not as high as it seems, as my data is recorded in millions, and an error of 1 million adds 1 million squared to the MSE, which is already 1 trillion. With over 300 points, 40 trillion doesn't sound too bad. My second and third models were also with OLS, but with 3 variables I deemed important, and 5 variables that I yielded from the OLS in the last section by slowly removing the least significant variable, respectively. The results of these models were that the R-Squared increased by 0.05 with each respective model, and the MSE reduced to 37 Trillion and then to 33 trillion, so it seemed like adding variables was making the model fit better. For all of my models, I make a scatter plot of the predicted values against the true values and plot a diagonal line to show where the data should be if everything were predicted with 100% accuracy.

I changed the regression algorithm for my next model and did a basic Linear Regression. I used the 5 variables that I used in the last OLS model for this one. For this model, I utilized the train_test_split module from sklearn's model_selection and split the data into 70% training and 30% testing. I then fit the training data to the regression and scored the testing data. The results were worse than the OLS with the same features, outputting an R-squared of 0.43 and a MSE of 37 Trillion, while the OLS gave 0.50 and 33 Trillion. This could be the result of not having a smaller testing set than the OLS. I also used the .predict method from the regression and plotted the predicted y-values against the true y-values to see how they line up against the diagonal line. Since using the train_test_split method is very volatile (the score and predictions highly depend on which section of the data was selected as the training and test sets), I used cross validation (CV) to find the score and took the mean of the values. The mean R-squared from me cross-validation was 0.45. One thing to note, When I ran cross validation without disorganizing my data, I received terribly poor scores (think negative 4000). I assume that this is because the CV algorithm takes each chunk incrementally without shuffling, and some tiers of salary are not correlated with the data, but looking at the whole thing it does.

I moved onto more complex regression methods by trying out Lasso and Ridge regression. For my Lasso Regression, I used every variable (except the non-numeric or categorical ones), since Lasso Regression, in theory, should make the coefficients of the non-significant features 0, rendering them useless in the model. I used a for loop to test for the best alpha, but didn't get much of a difference in any of them. If anything, the regression seemed to score better as alpha got larger. So, I used the model with the best score (alpha = 1000), and did train_test_split with it. This regression yielded a much better score of 0.827 and a MSE of around 10 trillion. After cross-validation, I got a score closer to 0.813. Though the score is a sign of a better fit, this lower MSE may be cause because of the lower amount of data

points being predicted in the model. On the plot, the points seem to be plotting much closer to the diagonal line, though they seem to be over predicting until a certain point and even predicting some negative salary values. This may be caused because of the minimum salary a player can be paid in the NBA. You can see this in the other plots where there seems to be a floor for the actual salaries. So even if one player is getting better stats than another, they might still be getting paid the minimum salary.

My final model was a Ridge Regression. My score from cross-validation was higher with 0.828, but not by much. The MSE also went below 10 trillion. Looking at the graph, you can see that this model has the same problem as the Lasso Regression of over predicting until around the $1.5 million mark. I believe I was able to use Ridge and Lasso regression without selecting any particular features (besides removing the non-numeric and categorical features) because they both penalize the less important features by reducing their coefficients either to 0 or by making them smaller.