**CSCI 29 Final Project Transcript**

Carter Ithier

This project creates a machine learning model to predict the outcome of games from last year’s FIRST robotics high school competition. Each game starts with an autonomous period of 15 seconds (in which the robot follows preprogrammed commands) and is followed by a teleoperated period in which students can control the robots via remote control. Points can be scored by shooting balls into a boiler (known as adding pressure), delivering gears to an airship which can result in points for the rotors, and climbing a rope and hitting the touchpad.

There were 255 events in the 2017 season and each one has a unique event key in the Blue Alliance API we will be using to scrape data. We’ll start by getting all of the event keys.

Before we use these keys to actually scrape the data we should go over some helper functions first. The function get\_experience creates a dataframe for all of the teams in the competition containing how many years of experience they have. Other features that may be important are how many wins a team has, how many losses, how many ties, and points they’ve earned during the qualification matches. These points can be broken up into categories by match points, points in autonomous mode, points from adding pressure to the boiler, points from delivering enough gears to turn a rotor, and points from climbing the rope. All of these features are obtained in the get\_stats function. The get\_matches function gets all of the match information for the event, namely what teams are on what alliance and what the scores were. The alliance function takes the average and sum of different statistics for the teams comprising alliances to give one value for each feature for the alliance. For example the feature of blue wins is the sum of all the wins the teams on the blue alliance had and the feature blue rank is the average rank of the teams on the blue alliance. The calculateAlliance function calls the alliance function using the apply method on the matches dataframe so that the alliance function knows which teams comprise each alliance.

There are three features we are going to engineer: Offensive Player Rating (OPR), Defensive Player Rating (DPR), and Calculated Contribution to Winning Margin (CCWM). OPR is a calculation of a team’s contribution to their alliance’s final score. To understand how we calculate OPR let’s look at an example. Say we have alliances of three teams and the matches yield the following results:

A + B + C = 100

A + C + D = 130

B + C + E = 70

A + B + D = 150

B + D + E = 60

Here, the letter stands for how many points that team contributed to the alliance score and the number after the equals sign is the score of the alliance.

It’s likely that our system will be overdetermined so we will have to find the least squares solution and that will give us the OPR for teams A through E. To do this we will need to sum all of the games a team has been in. For example for A we would get 3A + 2 B + 2C + 2D = 380 and we would put this in matrix form in the first row of the matrix. We then do this same process for B through E. With our coefficient matrix and solution vector in hand we can then use the least square function in scipy to solve for the OPRs.

Now let’s look at how this was coded. The OPR function takes a list of team names and a dataframe of the qualification match information. First we create a square matrix m full of zeros that has the dimensions of the number of teams plus 1 to account for the score. We have a loop that goes through all of the teams. For each team it creates a dataframe of all of the games a specific team has been in. Then we create a square dataframe full of zeros with the same dimensions as m. Its column names are the teams and the last column is the score. We create two lists (blue and red) comprised of the teams making up those alliances. If the team we are currently on is in the blue alliance, a 1 replaces the 0 in the dataframe matrix for each blue team on the alliance and the score is replaced with the blue score. If it is on the red team, then a 1 replaces the 0 in the dataframe matrix for each red team and the score is replaced by the red score. Next a variable called new\_row is created which is the sum of all of the columns of the dataframe matrix. The new\_row is then assigned to the appropriate row in the m matrix. After this process is done for all the teams the m matrix is divided up into two matrices- the coefficient matrix and the solution vector. Then the least square solution is found using the least square function in scipy.

DPR is a calculation of a team’s contribution to their opponent’s score. Thus the lower the DPR, the better the team is at defense. It is calculated the same way as OPR except instead of having the sum of your alliance’s scores you have the sum of your opponents score.

CCWM is a measure of how much a positive impact a robot has on its alliance. It is found by taking the different between OPR and DPR for a team.

The get\_OPR function calculates OPR, DPR, and CCWM and returns a dataframe with this information for each team.

Our last helper function calculates the win margin of the blue alliance compared to the red alliance.

Now to get the data. We loop through each event key in our list that we obtained earlier and try to get all the data we need. We have a try/except clause in case some of the data we need for an event is missing. It’s possible that a team competed in multiple events and their ranking and statistic might be different for each event so we will process each event separately and save the event data to a csv. We first do get\_experience, get\_stats, and get\_matches. We figure out which are the qualification matches and pass that and the team list to get\_OPR. We then merge all of our dataframes on “team” and calculate the win margin for each match.

Next we load all of the csv files generated and combine them into one big dataframe. We check that the data is ready for the sklearn library. We find that it isn’t because there are null values. We also notice some of the files that have null values have negative scores listed which isn’t possible. We drop the null values and only take the rows where the blue and red score are greater than or equal to zero.

We need to separate the data into testing and training. The API only allows us to get data on the teams for the qualification matches as a whole. As a result we’ll have to use only elimination observations for training and testing (because you can’t use info from a game that hasn’t happened yet). We filter out the elimination data by finding when “level” is equal to eighth final, quarter final, semi final, or final and put this in a dataframe called data.

Then we divide our data into features and targets and split our data into training and testing.

The first regression we do is SVR regression. We use the optunity library to determine what parameters we should use for C, epsilon, and gamma. We next build our model. For SVM it is important to scale our data so we use Pipeline from sklearn.pipeline and the scaler from preprocessing. We fit our data to the pipeline and use it to predict.

In order to evaluate how our model did using a classification report we need to classify our outcome. The classify function takes y\_test and y\_pred and classifies each entry as either a win (2), tie (1), or loss (0). After running this function we can then do a classification report. We get a precision of 61%, a recall of 65%, a f1-score of 62% and from the confusion matrix we can see an accuracy of 64.8%

Let’s try a decision tree regression. To determine what our parameters should be we use RandomizedSearchCV. We use this output to create a decision tree regressor and then fit and predict. We classify the predictions and create another classification report. This time we find a precision of 58%, a recall of 62%, a f1-score of 59%, and from the confusion matrix we can see an accuracy of 61.9%.

Last, let’s try a random forest regression. We will use RandomizedSearchCV again to pick our parameters. We take the output and use it to fit and predict our random forest regressor and then we classify the predictions. We find a precision of 63%, recall of 64%, f1-score of 52% and from the confusion matrix we can see an accuracy of 63.5%.

The SVM regression performed the best although its accuracy isn’t too different from the Random Forest Regression. It had the highest recall, f1 score, and accuracy of the three tried. The Random Forest Regression had a slightly higher precision, recall, and accuracy than the Decision Tree Regression. In all regressions, the model over predicted losses and never predicted a tie.