Batting Above Average

Group 12

Brendan O’Connell – 16319077

Jack Mac Namara – 16321927

Introduction

The problem to tackle is determining if the outcome of a baseball game can be predicted using only statistics that could be provided to the model before the game starts. We believe that this problem is interesting as baseball is a very statistic-heavy sport, and we would like to investigate to what extent these statistics can really paint an accurate picture of future games.

The inputs to our model are the offensive statistics of players in a game’s starting line-up and the defensive statistics of the team they are playing against. The output is whether the team won or lost the game.

Collecting Data

We used Selenium Webdriver to go through all of the games for the 2018 MLB season on baseball-reference.com and scrape the starting line-up for both teams and separately scrape the defensive stats for each team and the offensive stats for all players.

What we did for this scraping, was isolate the areas in the 2018 games schedule page that linked to the stat sheet for each of those individual games. We pulled the link to each individual game and stored them in a csv file. We then opened the link for each game, and scraped the starting line-ups, team names, and score (which we used to deduce wins/losses). This was made easy by the layout of the website and for the most part individual class names or tag-class combinations were used only once which helped us to scrape the data from the website. Scraping the links to each game took only a few minutes, however, to scrape the starting line-ups from each of the 2463 games played in the 2018 MLB season took almost 9 hours. This time could have been significantly reduced by creating a scraper that could have scraped game data concurrently (opening more than one game at a time) as the main obstacles for running the code quickly was the response time of baseball-reference.com and Chrome Webdriver, not the computing power.

For data storage we used the Python package Pandas to parse all the data needed into and out of csv format. This gave us an easy storing method for a dataset this size, had it been larger we probably would have used a SQL database. This method however meant we could take the data out of the file and use it in our machine learning easily.

Formatting and Aggregating Data

The aim for the collected data is to format it in such a way that for each baseball game, there exists a row in our data corresponding to the home team’s offensive stats, the away team’s defensive stats and the outcome of the game (win or loss), and vice versa.

The first formatting issue that we had to fix, was to get rid of all the empty lines in the csv file, and to remove the index column as it is unnecessary for this problem. Then, we checked the entire dataset for duplicate games (in case the scraper had gotten confused when starting and stopping). We found that there were a small number of duplicate games that had all occurred when the scraper was stopped and the started from the incorrect place. The duplicate games were removed.

The first step of aggregating the data was to add each teams' defensive statistics to the row representing each game. However, we noticed that in the table which shows each teams' defensive stats for the season, they use the teams' abbreviated names, not the full text version that we had downloaded with our scraper. So, first we added a new column to each row: 'home_abbr' and 'away_abbr' which represents the abbreviated name of each team (e.g. Arizona Diamondbacks is abbreviated to ARZ). Then once each game had been given the correct abbreviated name for both the home and away teams, we selected 2 defensive stats: runs allowed per game (RA/G) and ERA (earned runs allowed, this is a good measure of how effective a team's pitchers are at stopping runners from scoring) and added them to the row.

The next step was to add each player from each team's offensive statistics to every game. First, we noticed an issue with this approach. Not every player that appeared in a game has end-of-season statistics, as Major League Baseball has a certain number of minimum plate appearances for recording statistics. We found a few instances of these such players being included in a starting line-up. Here we made the decision, that rather than assigning them average statistics or random statistics, we would give them zero in all categories. We made this decision because, if the player had such a small impact on the team that they are not included in the end-of-year statistics, then we should assume that they are (at least) below average. We decided this was the fairest way to implement this thinking. We found a (non-unique) player who had no stats in a given game only 30 times over the course of the season, which proves the effects of issue are negligible.

Next, we had to choose offensive stats to include for each player. Thankfully, meaningful offensive statistics are much more numerous than defensive ones, so rather than cherry-picking a couple of statistics at this step, we decided to add 9 offensive statistics for each player, and we would be able to cherry-pick in the training step. The statistics we chose were batting average (BA), on-base percentage (OBP), slugging percentage (SLG), on-base plus slugging (OPS), on-base plus slugging plus (OPS+), ground into double-play (GDP), hit by pitch (HBP), sacrifice bunts (SH) and sacrifice flies (SF). These statistics are held in arrays where the index of the names array matches the index of each player e.g. row['name'][1] = "Alex Rodriguez" implies that row['BA'][1] gives Alex Rodriguez's batting average.

The next step was to separate each game into two rows. In our model, we are analysing the offense of a team versus the defence of a team, therefore for each game, there are two matchups. This is a simple task, where we take one team's offensive player stats, their opponents' team defensive stats, and the outcome of the offense over the defence (win or loss). This doubles the number of rows we have to train with from 2463 to 4926.

An issue we thought we had accounted for, but we had to deal with at this stage was the fact that Major League Baseball is split into 2 different leagues, the National League and the American League. These 2 leagues have slightly different rules on starting line-ups. The American League allows 10 players in a starting line-up while the National League allows only 9 players. This means that half of all of our player statistic arrays have 10 indices, while the other half have only 9. Because of this, we decided to average out every offensive statistic across the starting line-up to give a fair reflection of the team's performance.

Now, we have a dataset, where the inputs are an array of 11 statistics (BA, OBP, SLG, OPS, OPS+, GDP, HBP, SH, SF, RA/G, ERA) and the output is either win or loss (1 or 0) as there are no draws in MLB.

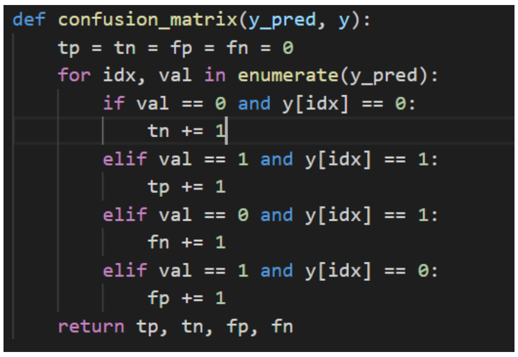
Methods and Results

We expect that the offensive stats of a team should be positively correlated to the team's odds of winning, and their opponent's defensive stats should be negatively correlated to the odds of them winning. This assumption is based off our empirical knowledge of sports.

We tested our data against 3 different types of classification models: Ridge, Lasso and Logistic Regression (with an L2 penalty). Ridge and Lasso are linear models that apply different penalties to the model coefficients. Ridge attempts to minimise all coefficients using an L2 penalty (shown below to the left) this leads to coefficients for unwanted features approaching zero. Lasso minimises all coefficients using an L1 penalty (shown below to the right) this leads to coefficients for unwanted features being zero. Logistic regression uses a non-linear categorical model and an L2 penalty. We performed k-fold cross-validation on each model using different sized folds, and varying costs being supplied to the model.

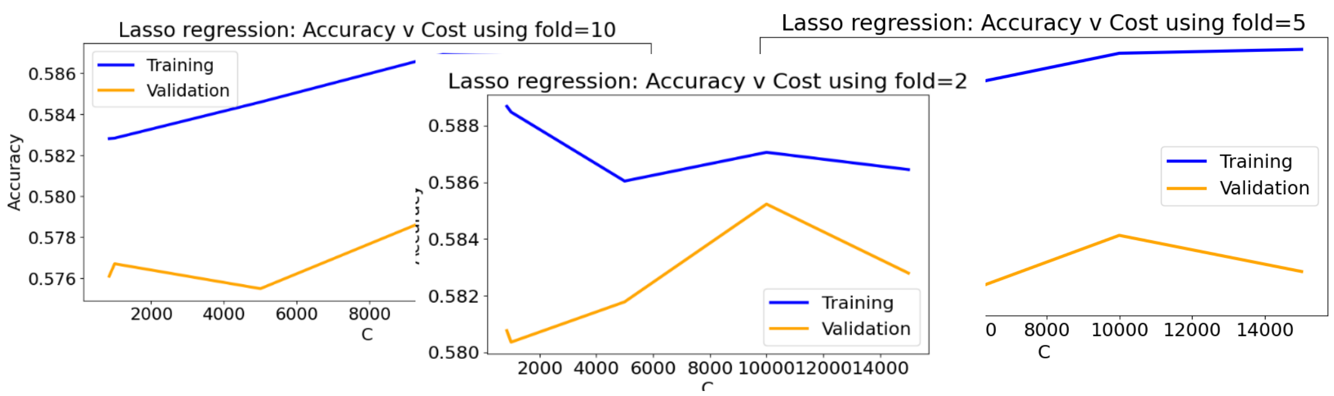
$$\boxed{+\lambda \sum_{i=1}^{n} \theta_i^2} \qquad \boxed{\lambda \sum_{i=1}^{n} |\theta_i|}$$

The statistics we used to measure the performance of the models were: accuracy, precision, recall and f1 score. We attempted to calculate these metrics using the in-built functions supplied by sci-kit learn. However, we found that their functions often threw errors due to the formatting of our data. Our solution was to find the confusion matrix of each model using our own code, and then calculate each of these metrics using the output of the confusion matrix. This gave us much more control of our metrics than using a 'black-box' function.



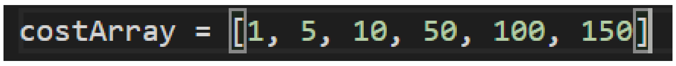
```
def confusion_matrix(y_pred, y):
    tp = tn = fp = fn = 0
    for idx, val in enumerate(y_pred):
        if val == 0 and y[idx] == 0:
            tn += 1
        elif val == 1 and y[idx] == 1:
            tp += 1
        elif val == 0 and y[idx] == 1:
            fn += 1
        elif val == 1 and y[idx] == 0:
            fp += 1
    return tp, tn, fp, fn
```

We decided that 5 folds are a suitable number of folds for each of the chosen models in this problem across a very large range of costs.

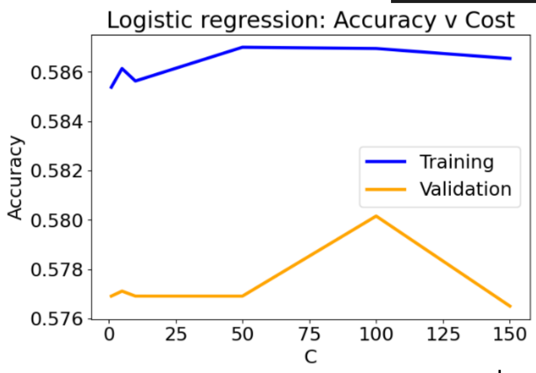


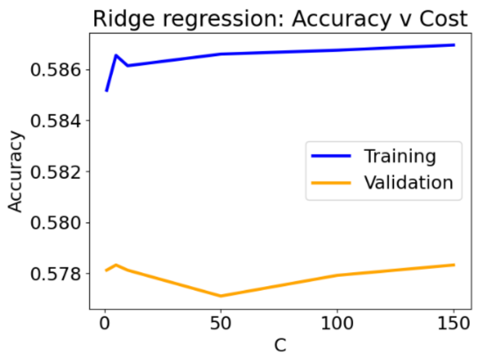
The next step was to analyse the different costs of each model after we had chosen 5 folds would be our hyperparameter for k-fold cross-validation. We fitted a model using our collected data and varying the cost applied to it by each model. We found that costs less than 1 caused some of the models to diverge, so we discarded all costs less than 1. We also found that providing very large costs (~10,000) did not cause any of the models to perform significantly better than if they are given a small or medium costs. The above diagrams might appear to be misleading, in that they seem to give much better performance for very high costs, however, we found that they may appear more accurate because of the small range on the y-axis in the graphs.

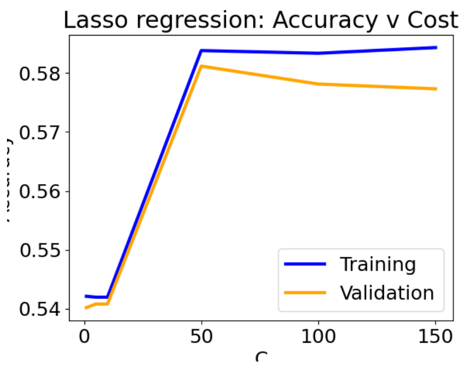
Below, we demonstrate the performance of our models across the new, smaller range of cost values shown here:



```
costArray = [1, 5, 10, 50, 100, 150]
```



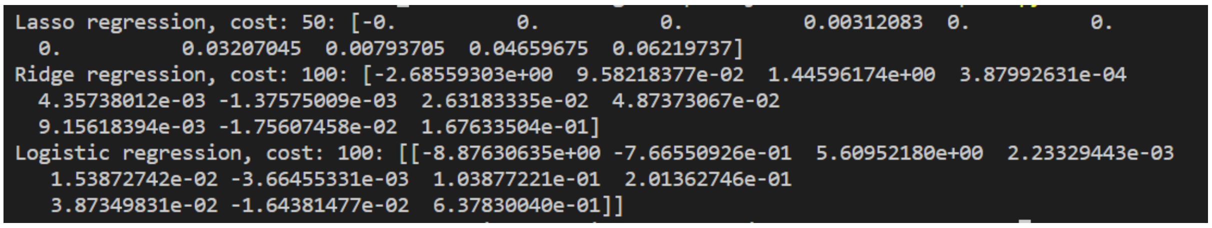




We found that Logistic Regression performs best with an L2 penalty using a cost of 100. Lasso regression performs best using a cost of 50. Ridge regression's performance stays almost constant for all values of C, therefore we will choose a C value of 100 for this model's best-fit.

These are the models produced by the best cost hyperparameters we deduced from the cross-validation.

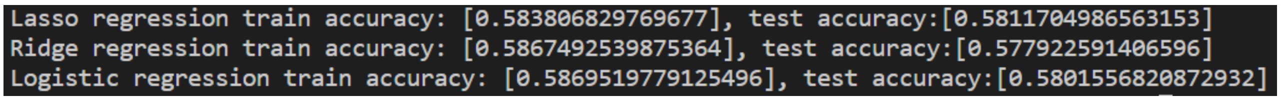
Something that we noted here is that our prediction that the offensive stats would have positive correlations, and the defensive stats would have negative correlations is incorrect. The models have all assigned no value or negative value to the batting average (offensive), and positive values to ERA (defensive).



```
Lasso regression, cost: 50: [-0.          0.          0.          0.00312083  0.          0.
  0.          0.03207045  0.00793705  0.04659675  0.06219737]
Ridge regression, cost: 100: [-2.68559303e+00  9.58218377e-02  1.44596174e+00  3.87992631e-04
  4.35738012e-03 -1.37575009e-03  2.63183335e-02  4.87373067e-02
  9.15618394e-03 -1.75607458e-02  1.67633504e-01]
Logistic regression, cost: 100: [[-8.87630635e+00 -7.66550926e-01  5.60952180e+00  2.23329443e-03
   1.53872742e-02 -3.66455331e-03  1.03877221e-01  2.01362746e-01
   3.87349831e-02 -1.64381477e-02  6.37830040e-01]]
```

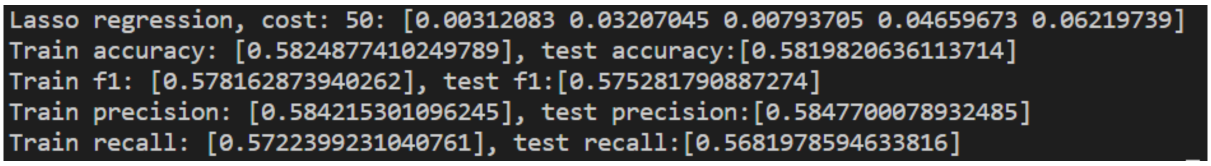
The 11 numbers given in each model represent the weightings given to each statistic we supplied to the model i.e. lasso_model[0] applies to the team's batting average and lasso_model[9] applies to the team's opponent's runs allowed per game. The below output shows that all of our best-fit models perform very similarly, but Lasso regression performs marginally better than the others.

We have chosen Lasso regression with a cost of 50 (α = 1/100) as our best functioning model. When we printed out the coefficients used by each model, the Lasso model showed that it had 6 coefficients that were valued at zero (i.e. made no contribution to the prediction). As such we



```
Lasso regression train accuracy: [0.583806829769677], test accuracy:[0.5811704986563153]
Ridge regression train accuracy: [0.5867492539875364], test accuracy:[0.577922591406596]
Logistic regression train accuracy: [0.5869519779125496], test accuracy:[0.5801556820872932]
```

believe that removing these 6 values and re-training will give a similar prediction.



```
Lasso regression, cost: 50: [0.00312083 0.03207045 0.00793705 0.04659673 0.06219739]
Train accuracy: [0.5824877410249789], test accuracy:[0.5819820636113714]
Train f1: [0.578162873940262], test f1:[0.575281790887274]
Train precision: [0.584215301096245], test precision:[0.5847700078932485]
Train recall: [0.5722399231040761], test recall:[0.5681978594633816]
```

As shown here, removing the statistics that the model found to have no effect on predicting caused the model to perform exactly the same as before. The statistics that the model found not to be important were: BA, OBP, SLG, OPS+, GDP and HBP. The statistics that were found to have a significant relationship to the win/loss odds of a team were: OPS, SH, SF, RA/G and ERA. This result was very unexpected, but it still gives an accuracy significantly better than that of a random guess (which would only achieve an accuracy of 50%).

The results here also show that for all of the other comparative statistics we used, the model gives similar answers for both training and testing. This shows that we had a large enough training dataset to create a strong model.

Summary

The best accuracies we were able to achieve were in the range of 57-59%. The baseline model we would compare against would be a model that randomly guesses one team is the winner of each game. We would expect this model to perform at 50% accuracy. Our model performs significantly better than a random guess, but it is not as effective as we hoped. We think that one of the main causes of this is that statistics can only provide a partial picture as to how a given sports game will unfold because of the large human element involved in playing sports. Another possible weakness in our model regarding this point, is that we used each player's end-of-season statistics.

If we were to look at this problem again, we would look into accommodating a player's current 'form'. We would do this by taking an average of a given player's stats over the past n games they have played where we would determine which n gives the best representation of recent 'form' using cross-validation.

A noteworthy part of our investigation was that we trained models where we did not average the offensive statistics across the team, but we supplied all statistics for each player. This improved the accuracy of our models by roughly 1-2%. This led to issues with the American League's and the National League's differing rules for starting line-ups as our input arrays were of different lengths for some games. We decided to use averages to avoid the line-up rules and to decrease the size of our models. If we were to perform this experiment again, we might take more considerations for the order of a starting line-up i.e. players at the start of the line-up might have greater weightings on the outcome of a game.

Our first attempt at this project involved us predicting the scores of games given the same statistics we have already listed. We found that creating a regression model to predict scores often only predicted scores very near to the average number of runs scored per game no matter how the costs were varied. This model was very inaccurate and so, led to us creating a binary classification model (win or loss). This is why the linear regression models are included in our final analysis.

Contributions

Jack Mac Namara – Created the web scraper which collected all of the data

Brendan O'Connell – Performed the aggregating and formatting of the collected data

Together – Applied machine learning models to the collected data, performed cross-validation on these models and created the final report.

Code

See zip attached with submission