# ECE356 Lab4

Kordian Mazurkiewicz (kjmazurk@uwaterloo.ca, 20933336), Michael Tham (m3tham@uwaterloo.ca, 20945854)

B)

**<u>Task A:</u>**
Total Instances: 21010, Class 1 Nominated: 1528, Class 0 Nominated: 19482, Nominated Proportion: 0.0727

<u>Query:</u>

```
SELECT t as total_instances, n as nominated, t - n as notNominated, n / t as nominated_proportion, (t-n)/t as notNominated_proportion

FROM

        (SELECT COUNT(DISTINCT playerID) as t FROM People) as A,

        (SELECT COUNT(DISTINCT playerID) as n FROM HallOfFame) as h;
```

**<u>Task B:</u>**
Total Instances: 1528, Class 1 Inducted: 1528, Class 0 Not inducted: 1171, Inducted Proportion: 0.2336

<u>Query:</u>

```
SELECT t AS total_instances, n AS inducted, t - n AS notInducted, n / t AS inducted_proportion, (t - n) / t AS notInducted_proportion

FROM

  (SELECT COUNT(DISTINCT playerID) AS t FROM HallOfFame) AS A,

  (SELECT COUNT(DISTINCT playerID) AS n FROM HallOfFame

   WHERE REPLACE(REPLACE(Inducted, '\r', ''), '\n', '') = 'Y') AS h;
```

C)

The dataset provided for this lab while extensive did provide a couple cleansing challenges to work with when developing this decision tree. The two major challenges we faced in order of when we encountered them were identifying data to cleanse and hidden characters.

1) The first issue was identifying what data we wanted to cleanse. There is a lot of data in each table and lots of it isn't relevant when deciding whether a player will be nominated or inducted into the MLB hall of fame. We solved this issue by looking through the readme and identifying key metrics that could possibly determine a hall of fame worthy player. This was luckily made easier as we both had some baseline knowledge of baseball. Thus, choosing features to cleanse that would provide an impact on the decision tree was made a bit easier. Another small bug that emerged is not using the correct variant of left or base join and having too much of one dataset, causing incorrect results. This was even more difficult to debug since the final values are not null but incorrect counts or sums. This all culminated with us having to fully understand how the whole data set worked and how each table interacted with each other to properly cleanse the data and obtain the correct data in the features tables. This was achieved by doing accurate SQL queries such as the correct joins on the given dataset.

2) The most challenging cleansing issue we faced was fundamentally the easiest issue to solve. That being the hidden characters when working with the dataset. Our process to get the data in our MySQL database, while not the most direct, was as follows: We first had a python script reduce the data sets we needed to only relevant columns so that it was easier for us to manually look through the data sets to see if any pattern emerged. Such as higher fielding and hitting stats leading to a higher chance of HOF nomination. From the modified csv state, we then imported it into a MySQL database using SQL. Once the data was then into our database, we finally use the deliverable SQL file to export to the csv files that contained the classification labels and features. This was when we saw some hard to debug issues. Very essential data points were loading wrong, such as our case statement output determining which award a certain player has won, and the world series wins data showing that no player has ever won the world series. By inspection we could easily see these were wrong. The first step in the solution was simple sanity checks to prove the data is wrong. But issues for the world series showing zero wins were quite difficult to debug, was it our logic, did we take the first step of cleansing i.e. the importing phase wrong for that data? After hours of wondering what could be so wrong, we realized that there was almost always a hidden '\r' character following the 'Y' or 'N'. We aren't sure if this was caused by our python reducing columns script, our SQL import script, or from the original dataset, but once we were able to add REPLACE statements with this hidden character and others such as '\n' into our SQL, the values returned correctly for the awards count based off the 4 most important awards, as well as for world series wins.

D)

Initially we planned to combine several of the metrics in these tables and have a small set of features, but all of them being very strong and calculated. We did this to make sure that we are not over fitting the model. We did this by making a feature called O/D/PWAR and averaged these individual batting, fielding and pitching stats with weighted averages of what's more important, but while developing we realized it was easier to adjust hyperparameters rather than features. Thus, we pivoted to a large amount of features list. To compensate we planned to reduce overfitting in the hyper-parameter section as that's easier to change on the fly while testing. This will be further explained in part g).

Initial Feature Selection:

1) **Team Metrics Features:**
   WinPercent: The player's team's winning percentage during their active years.
   Rationale: Winning teams are often associated with successful players.
   WSWins: The total number of World Series won.
   Rationale: Championship wins are a significant milestone that can increase a player's Hall of Fame candidacy.
2) **Career Batting Features:**
   H_Batting, HR_Batting, RBI_Batting, R_Batting: Totals for hits, home runs, runs batted in, and runs scored.
   Rationale: Fundamental performance stats.
   Doubles_Batting, Triples_Batting: The number of doubles and triples hit by the player.
   Rationale: Fundamental performance stats.
   BB_Batting, AB_Batting: Totals for walks (base on balls) and at-bats.
   Rationale: Fundamental performance stats.
3) **Career Fielding Features:**
   PO_Fielding, A_Fielding, E_Fielding, DP_Fielding: Totals for putouts, assists, errors, and double plays.
   Rationale: Fundamental performance stats.
   SB_Fielding, CS_Fielding: Totals for stolen bases allowed and caught stealing.
   Rationale: Fundamental performance stats.
4) **Career Pitching Features:**
   ERA_Pitching: Career earned run average (ERA).
   Rationale: ERA is one of the most important metrics for evaluating a pitcher's effectiveness.
   H_Pitching, SV_Pitching, BAOpp_Pitching: Totals for hits allowed, saves, and opponent batting average.
   Rationale: Fundamental pitching stats.
5) **Postseason Features:**

H_Battingp, HR_Battingp, RBI_Battingp, R_Battingp: Postseason totals for hits, home runs, RBIs, and runs.
Rationale: Success in high-stakes postseason games can boost a player's candidacy for the Hall of Fame.
Doubles_Battingp, Triples_Battingp, BB_Battingp, AB_Battingp: Additional postseason batting metrics.
PO_Fieldingp, A_Fieldingp, E_Fieldingp, DP_Fieldingp: Postseason fielding totals.
ERA_Pitchingp, H_Pitchingp, SV_Pitchingp, BAOpp_Pitchingp: Postseason pitching statistics.
Rational: Post-season stats are more important then regular season stats.

6) **Milestone Features:**
GP_AllStar: The number of All-Star games the player participated in.
Rationale: Being selected for an All-Star game indicates excellence and recognition by peers, fans, and coaches.
careerAwards: Total career awards won for the most important ones. (MVP, Cy Young, Gold Glove, Silver Slugger).
Rationale: Awards are direct indicators of players individual skills and HOF potential.

E)

As given in the lab report we were tasked to use an 80/20 testing/training split. We first needed to do this split with the given data before we sampled these subsets. Before splitting we first shuffled the csv data by using the function df.sample(frac=1, random_state=42) on our df = pd.read_csv('features/taskA.csv'). We do this to avoid the bias's created by the automatic ordering of playerID names. Afterwards we then divided the randomized dataset into two column parts. X, that contains all our features, and Y which contains the class information (1 if nominated, 0 if not for Task A and inducted vs not for task B). From this point, the data is separated into the 80/20 split as required in the lab manual. We then initialize the decision tree classifier with the set depth and leaf nodes. Afterwards we fit the decision tree using the given function: clf.fit(X_train, y_train), then generate the prediction results by y_pred = clf.predict(X_test). After setting up the prediction results, we compare the values of this prediction method to the true value of y with these random dataset values. This gives us the confusion matrix using this function: confusionMatrix = confusion_matrix(y_true, y_pred). Afterwards we plot the tree. We believe this is a good way because we randomized the splitting portion of the dataset, thus, it trains on randomized data and not a specific portion of the data. This is especially important since there could be biases on the base csv ordering like playerID's, as there could be better stats for people with earlier names just coincidentally. Another possibility is that it could favor better hitters. Thus, our sampling strategy of randomizing the csv data then splitting provides mostly a random and general training and testing subsets.

F)

Task A Confusion Matrix/Other Metrics:

| | |
|---|---|
| 3795 | 94 |
| 90 | 223 |

Accuracy: 95.62%, Precision: 70.35%, Recall: 71.28%, Specificity: 97.58%, F1 Score:70.79%

Task B Confusion Matrix/Other Metrics:

| | |
|---|---|
| 201 | 35 |
| 14 | 56 |

Accuracy: 83.9%, Precision: 61.54%, Recall: 80%, Specificity: 85.1%, F1 Score: 69.47%

Formulas:

Acc = TP + TN /TI, Prec = TP / TP + FP, Rec = TP / TP + FN, Spec = TN / TN +FP,
F1 = 2*Prec+Rec/Prec + Rec

Why would one measure be preferred over another:

Accuracy: is best for balanced datasets or as a general metric but can be misleading in imbalanced cases.

Precision: is best when false positives are more important, while recall is better when missing positives is costly.

Recall is essential when missing true positives is more critical than avoiding false positives.

Specificity is important in contexts where identifying negatives correctly (e.g., avoiding false positives) is very important.

F1 Score is a good choice when both precision and recall are important and need to be balanced.

G)

Task A Prediction Problem:

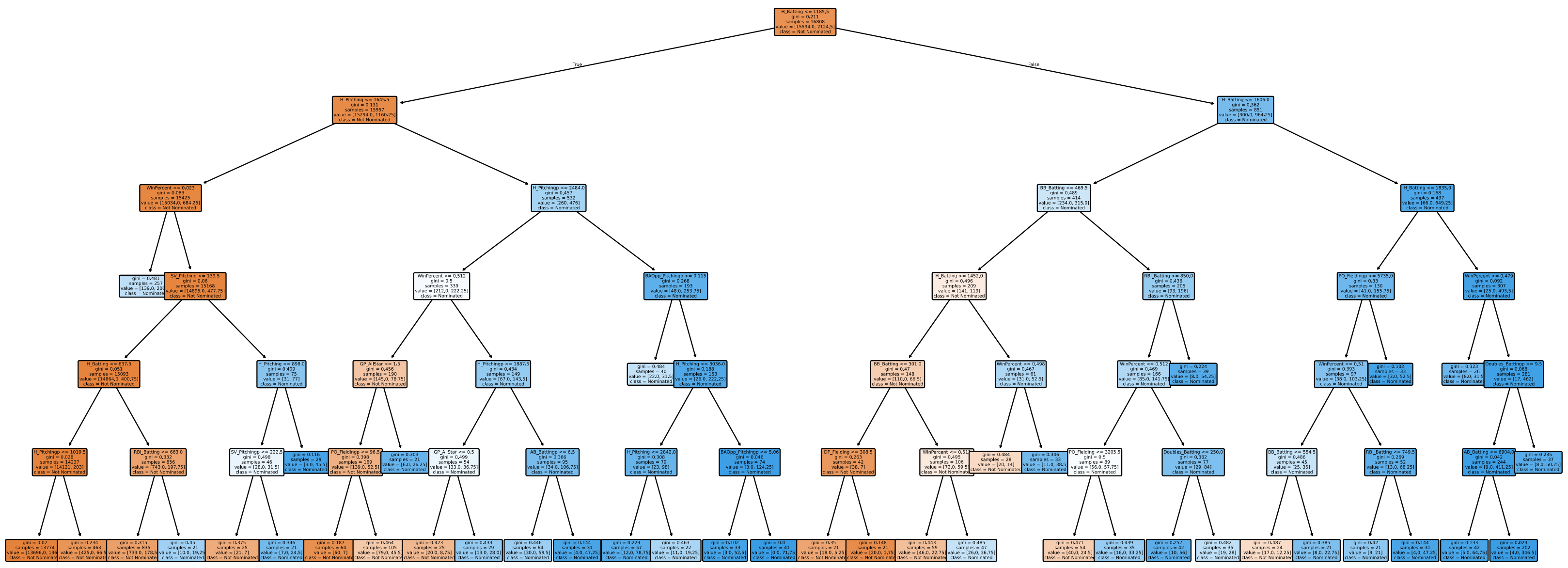
As mentioned in part D, we chose a large feature list as we thought it would be easier to address the problem of overfitting on the tree classification stage rather than modify our features. As well we thought the results would be better. When we tuned these hyperparameters, we found that at tree depths < 8 we get more accurate results then the smaller values of 5-7. But the increase in accuracy, recall and precision wasn't worth the over fitting of the model. Thus, we settled on a depth of 6. In task A this is especially more important as the data set is larger. To re-enforce this made our minimum number of leaf samples to be set at 21, which is a reasonable amount compared to the depth. The last parameter we tuned was the class weight. Due to the large amount of non-nominated players in the dataset (compared to the nominated players) we decided to increase the impact on the tree that a nominated (1) classification had. This provided us with accuracy, precision and specificity that provides a good output to the model, gaining a small reciprocal amount by increasing the depth while ensuring the model doesn't overfit. Thus, even though we have a plethora of features, reducing the min leaf nodes while keeping the depth small gives us an accurate tree that's not over fit, as it may choose what features to utilize more. Additionally, following up to what is mentioned in part D, if we were to reduce our feature count, our ability to play with the depth and leaf nodes would be limited since we would need to increase the depth to make up for the lack of features, causing inadvertent over-fitting. Overall, since our accuracy, recall and precision were still very high we kept our initial feature list.

Task B Prediction Problem:

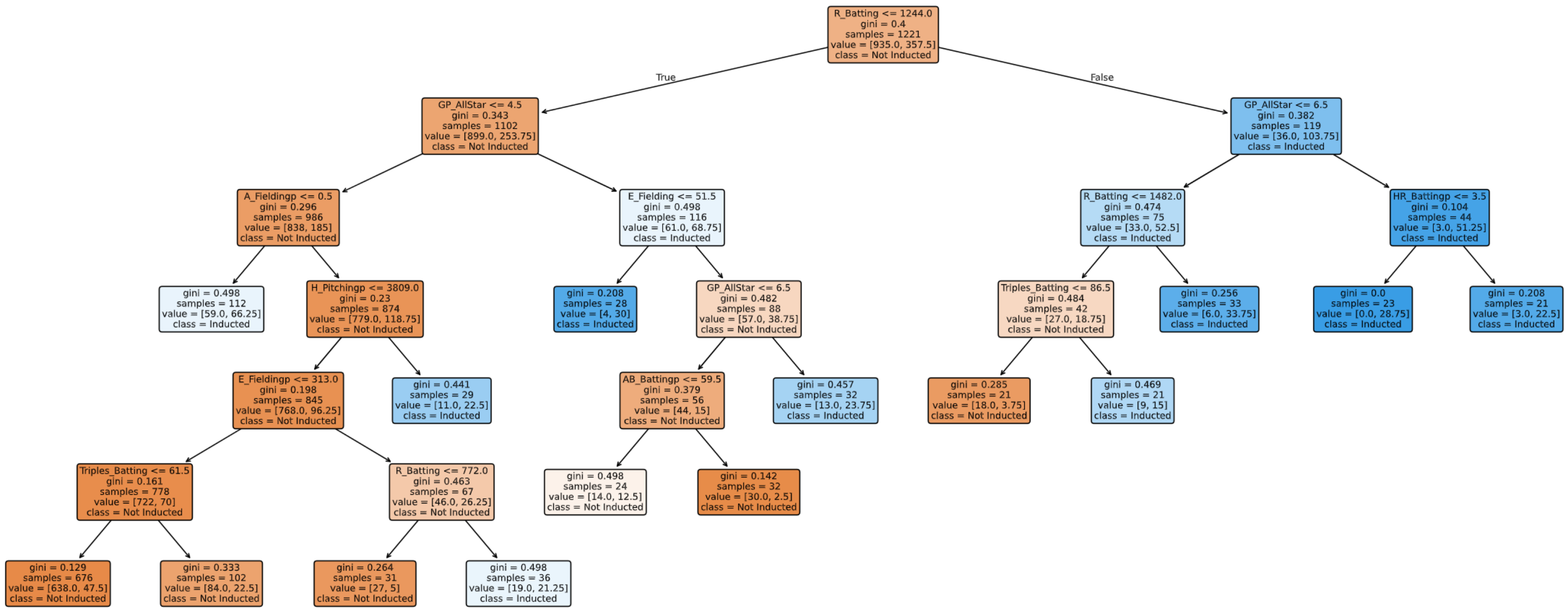
Similarly to task A, we knew that we would need to have a relatively smaller depth given our large feature set. After trying alternative depths, we found that the same trade-offs applied. Better accuracy, recall and precision for a larger tree risked overfitting the data. Like task A, we found that the test results from over-fitting did not outweigh the negative consequences at a max depth size of 6. The same followed for the min leaf nodes so we kept it at 21. This procedure is far more important for task A because the dataset of nominees is a lot smaller then then all players so it can become very easy to over fit the data, since it can easily fit itself exactly on the small dataset. While testing alternative amounts at depth 4 and 5, the stats worsened greatly, making under fitting unreasonable. We ended up with the same hyperparameters except for class weight, as since there were less people in the dataset overall, we decided to decrease the weighting of an inducted classification. This tuning is especially important, since with a large feature count, we could easily over fit the model. Shrinking the tree allows the model to utilize the most important features when certain patterns are identified. After testing and tuning the trade-offs of over fitting vs accuracy, we still found that our extensive feature list always balanced itself off when we forced the tree to be a smaller with a max depth of 6 and min leaf nodes of 21. Thus, since our accuracy, recall and precision were still very high we kept our initial feature list.

H)

Task A Tree:



Task B Tree:


R_Batting <= 1244.0
gini = 0.4
samples = 1221
value = [935.0, 357.5]
class = Not Inducted
True
False
GP_AllStar <= 4.5
gini = 0.343
samples = 1102
value = [899.0, 253.75]
class = Not Inducted
GP_AllStar <= 6.5
gini = 0.382
samples = 119
value = [36.0, 103.75]
class = Inducted
A_Fieldingp <= 0.5
gini = 0.296
samples = 986
value = [838, 185]
class = Not Inducted
E_Fielding <= 51.5
gini = 0.498
samples = 116
value = [61.0, 68.75]
class = Inducted
R_Batting <= 1482.0
gini = 0.474
samples = 75
value = [33.0, 52.5]
class = Inducted
HR_Batting <= 3.5
gini = 0.104
samples = 44
value = [3.0, 51.25]
class = Inducted
gini = 0.498
samples = 112
value = [59.0, 66.25]
class = Inducted
H_Pitchingp <= 3809.0
gini = 0.23
samples = 874
value = [779.0, 118.75]
class = Not Inducted
gini = 0.208
samples = 28
value = [4, 30]
class = Inducted
GP_AllStar <= 6.5
gini = 0.482
samples = 88
value = [57.0, 38.75]
class = Not Inducted
Triples_Batting <= 86.5
gini = 0.484
samples = 42
value = [27.0, 18.75]
class = Not Inducted
gini = 0.256
samples = 33
value = [6.0, 33.75]
class = Inducted
gini = 0.0
samples = 23
value = [0.0, 28.75]
class = Inducted
gini = 0.208
samples = 21
value = [3.0, 22.5]
class = Inducted
E_Fieldingp <= 313.0
gini = 0.198
samples = 845
value = [768.0, 96.25]
class = Not Inducted
gini = 0.441
samples = 29
value = [11.0, 22.5]
class = Inducted
AB_Battingp <= 59.5
gini = 0.379
samples = 56
value = [44, 15]
class = Not Inducted
gini = 0.457
samples = 32
value = [13.0, 23.75]
class = Inducted
gini = 0.285
samples = 21
value = [18.0, 3.75]
class = Not Inducted
gini = 0.469
samples = 21
value = [9, 15]
class = Inducted
Triples_Batting <= 61.5
gini = 0.161
samples = 778
value = [722, 70]
class = Not Inducted
R_Batting <= 772.0
gini = 0.463
samples = 67
value = [46.0, 26.25]
class = Not Inducted
gini = 0.498
samples = 24
value = [14.0, 12.5]
class = Not Inducted
gini = 0.142
samples = 32
value = [30.0, 2.5]
class = Not Inducted
gini = 0.129
samples = 676
value = [638.0, 47.5]
class = Not Inducted
gini = 0.333
samples = 102
value = [84.0, 22.5]
class = Not Inducted
gini = 0.264
samples = 31
value = [27, 5]
class = Not Inducted
gini = 0.498
samples = 36
value = [19.0, 21.25]
class = Inducted