

UNIVERSITY of WATERLOO

ECE 356

DATABASE SYSTEMS

Lab #04

Student:

Daniel Alonso dos Santos

dalonsod@uwaterloo.ca

21172347

Instructor:

Wojciech Golab

05 DECEMBER 2024



UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Number of instances

For each section, the dataset used is different. This is due to the nature of the problem statement itself. Therefore, this section will be divided into two subsections: Task A and Task B.

Note: In both subsections, the following considerations have been made:

- 'Player' refers to someone who has at least one recorded match played as a player.
- When "Given an arbitrary player" is mentioned in Task A and Task B, it refers to "Player" as defined above.
- If a player has also been a coach, only their data as a player will be considered, not their data as a coach.
- Consequently, from the HallOfFame table, we will only retain rows where category='Player'.

Task A

For this task, taking into account the above considerations, a total of 5,149 instances were obtained. While it is true that there are 20,731 players with data in Appearances, the vast majority of them did not have salary data. As we will see in the following sections, salary is an important feature, and it is challenging to assign a value when it is null. Assigning a value of 0 would reduce accuracy, and using the mean salary, for example, would also negatively impact the results. We will discuss this further later on.

Since only 5,149 "Players" had salary data, this became the dataset size.

Here is the query used:



```
WITH
... -- Here is the principal query (see createCSV_taskA.sql)

SELECT COUNT(*) AS total_instances
FROM data_set;
```

Regarding the number of instances in each class and their corresponding proportions, the following was obtained:

Class	Class count	Class proportion
Y	511	0.09924
N	4638	0.90076

It is important to note that some players (52, specifically) listed in the HallOfFame table did not have data in the other tables, so they were excluded.

The SQL query used:

```
WITH
... -- Here is the principal query (see createCSV_taskA.sql)

SELECT class,
       COUNT(*) AS class_count,
       COUNT(*) / (SELECT COUNT(*) FROM data_set) AS class_proportion
FROM data_set
GROUP BY class;
```

Task B

For Task B, we were asked: "Given a player who has been nominated for the Baseball Hall of Fame, predict whether the player will be inducted." This means we only need to use the Hall of Fame data where the category is 'Player'.

If we look closely, these data are exactly the 'Y' class from the previous section, which corresponds to 511 instances. That is precisely the number obtained when executing the query: 511 instances.

Note: The queries used to calculate the values for this section are identical to those in the previous section, except for the part in the WITH clause where the data_set is defined.

Regarding the number of instances in each class and their corresponding proportions, the following was obtained:

Class	Class count	Class proportion
Y	66	0.1292
N	445	0.8708

Data cleansing issues

For this section, it is important to note the following: MySQL was executed from the terminal using SQL files in the format source file.sql. Neither MySQL Workbench nor similar applications were used. As a result, loading the data was a labor-intensive process.

Initially, a createTables.sql file was created based on the table descriptions in the TXT file, but two main issues arose:

- In some cases, the number of columns in the TXT file's table descriptions did not match the total number of columns in the corresponding CSV files. Sometimes there were more columns, sometimes fewer.
- Occasionally, the column order was inconsistent.
- Sometimes, column names differed between the TXT and the CSV files.

As a result, the INSERT statements had to be configured manually, one by one, according to the TXT headers.

Data cleansing process

- **Missing Players:** Some players (52 in total), such as "bellco99," appeared in the Hall of Fame table but had no information in the other tables. These players were not included in the dataset.
- **Salary Information:** Out of 20,731 players, only 5,149 had salary information. As previously mentioned, salary is an important feature, and assigning a value when it is null is challenging. Assigning a value of 0 would reduce accuracy, and using the mean salary would also negatively impact the results. Therefore, all rows of players without salary information were excluded.

- **Other Missing Values:** For the remaining missing values, a default value of 0 was assigned to numeric fields. Unlike salary, if a player has no information about pitching, it is likely because they do not play in that position, making it reasonable for their statistics to be 0.
- **Dates and Years:** Missing year and date values posed challenges, as they would be converted to 0 or 00/00/0000 when loaded from the CSV file. Since no date columns were used in the analysis, this issue was not addressed. However, if necessary, missing year values could be replaced with the mean or median of the player's other rows. For example, if a player has rows for 2000 and 2002, the missing year could be set to 2001. If no other rows with years exist for the player, the entire row could be discarded, as it would be difficult to determine a reasonable date.
- **Outliers:** Regarding data cleansing, apart from handling missing data, no significant outliers were observed (e.g., extremely high values, incorrect or nonexistent dates, etc.).

Selection of features

For this section, it is important to note that I have no prior knowledge of baseball or similar sports. As an exchange student from Spain, the most comparable sport I am familiar with might be soccer (although they are very different). To approach this task, I attempted to draw parallels with what is needed in soccer to win the Ballon d'Or, to understand what might be required to enter the Baseball Hall of Fame. Some of the selected features are based on this analogy. Additionally, I researched online to identify key player statistics, though I am not entirely confident that the selection is optimal.

A total of 27 features were selected, detailed as follows:

General Statistics

- **salary_ratio:** Assumes that players earning higher salaries are better and therefore more likely to enter the Hall of Fame. To avoid bias from inflation over the years, this feature calculates the player's average salary divided by the average salary of other players for each year, and then computes the mean of these ratios.
- **points_ratio:** Derived from the AwardsSharePlayers table, this is calculated as the total SUM(pointsWon) divided by SUM(pointsMax) across all awards the player has participated in.
- **total_first_place_votes:** The total number of first-place votes a player received throughout their career.
- **total_awards:** The number of awards the player won during their career.

All-Star Game Statistics

- **total_GP:** The total number of All-Star games played.
- **perc_starting:** The percentage of games played in a starting position out of the total All-Star games.

Batting Statistics

- **TotalBatGames:** Total number of games as a batter. The more games, the higher the likelihood of entering the Hall of Fame.
- **total_bat_games_post:** Total number of postseason games as a batter.
- **BattingAverage:** Calculated as $\text{SUM(Hits)}/\text{SUM(At Bats)}$.
- **AvgRunsPerGame:** Average number of runs per game.
- **AvgHomeRunsPerGame:** Average number of home runs per game.
- **AvgRBIsPerGame:** Average number of runs batted in per game.
- **AvgNoStrikeoutsPerGame:** Average number of games without a strikeout.

Pitching Statistics

- **TotalPitGames:** Total number of games as a pitcher.
- **total_pit_games_post:** Total number of postseason games as a pitcher.
- **WinLossRatio:** General win-to-loss ratio.
- **AvgERA:** Global earned run average.
- **AvgSO:** Average number of strikeouts per game.
- **AvgCG:** Average number of complete games per season.
- **AvgSHO:** Average number of shutouts per game.

Fielding Statistics

- **TotalFieGames:** Total number of games played in a fielding position.
- **total_fie_games_post:** Total number of postseason games played in a fielding position.
- **avg_PO:** Average number of putouts per game.
- **avg_A:** Average number of assists per game.

Lab #4

- **avg_E:** Average number of errors per game.
- **avg_DP:** Average number of double plays per game.
- **avg_ZR:** Global average zone rating.

Notes on feature selection for batting, pitching y fielding

- Players with better average stats per game are generally considered better, increasing their likelihood of entering the Hall of Fame. That's why the average stats were selected.
- However, players could achieve exceptional averages from only playing one or two games. To address this, the total number of games is also included to reward consistency.

Sampling Training and Test

A commonly used method for training data is to randomly split the dataset into training (80%) and testing (20%) subsets. This approach prevents overfitting by ensuring that the model is trained and tested on separate data and provides a good estimate of model accuracy.

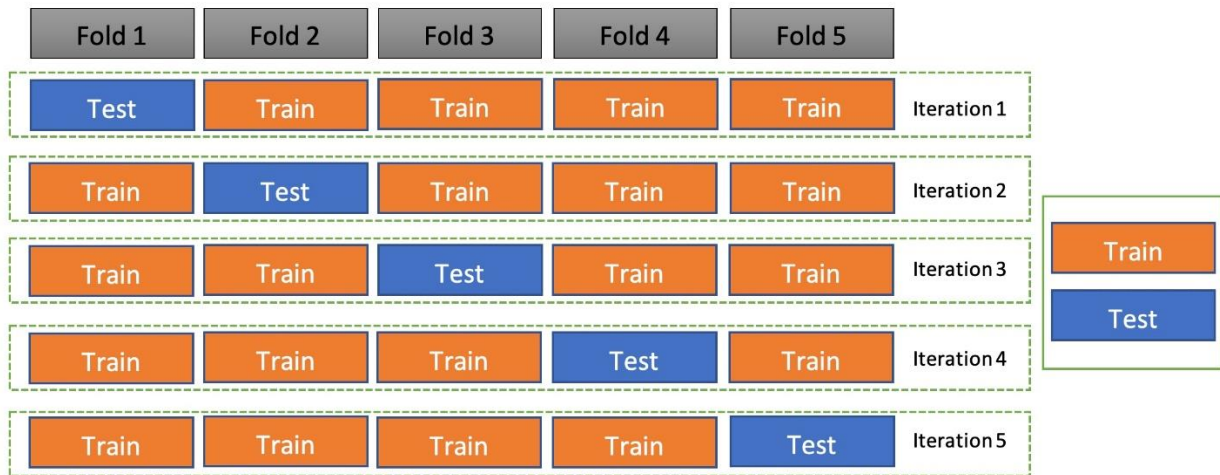
However, even with random splitting, the specific train-test division might inadvertently favor the model, potentially inflating the observed accuracy beyond its true value.

To achieve a more robust evaluation, 5-fold cross-validation has been employed for this lab. This method works as follows:

1. The dataset is divided into 5 equal subsets (folds).
2. Five rounds of training and testing are performed. In each round:
 - a. A different fold is used as the test set.
 - b. The remaining four folds are used as the training set.
3. After each iteration, the model is trained on the training data and evaluated using the test fold.
4. The final accuracy is calculated as the mean accuracy across all five rounds.

This approach ensures that every instance in the dataset is used for both training and testing at least once, providing a more reliable measure of model performance.

Visualization of the Cross-Validation Process:



The code implementation for this process is available in the file: classifierTaskA.py.

Validation results

The evaluation data obtained is shown below:

Task A:

Accuracy: 0.9404

Recall: 0.6437

Precision: 0.7281

F1 Score: 0.6805

Sum Confusion Matrix. Since 5 folds were performed, it is shown as the sum of the 5 obtained matrices (the average could also have been calculated).

	Real N	Real Y
Predicted N	4513	125
Predicted Y	182	329

Task B:

Accuracy: 0.9022

Recall: 0.5462

Precision: 0.6429

F1 Score: 0.5801

Sum Confusion Matrix. Since 5 folds were performed, it is shown as the sum of the 5 obtained matrices (the average could also have been calculated).

	Real N	Real Y
Predicted N	425	20
Predicted Y	30	36

In both cases, it can be observed that the accuracy is very high (close to 95%), but the Confusion Matrix is not entirely as we would like, as there are too many false positives compared to true positives. This is because the classes are imbalanced, with many more samples from the 'N' class (in both tasks) than from the 'Y' class. Therefore, it is much easier for the model to find an 'N' and get it right than to find a 'Y' and get it right.

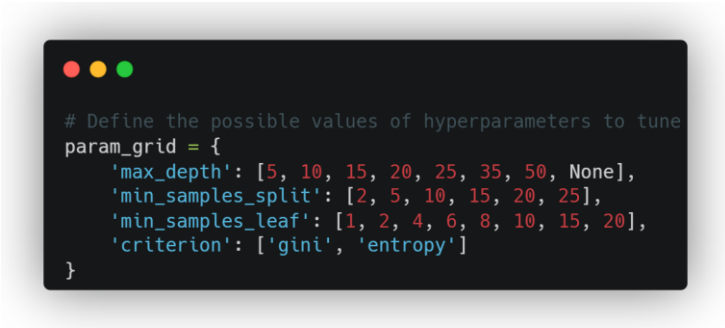
To try to address this, two measures were taken:

- Add the `scoring='balanced_accuracy'` attribute to the `grid_search`, so that it does not take into account total accuracy, but rather accuracy weighted by groups. (More about grid search will be discussed in the next section).
- Add `class_weight='balanced'` as a hyperparameter to account for a similar number of instances from each class (although it had no effect).
- Other techniques like SMOTE could have been applied to balance the data, but that is outside the scope of this course.

Hyperparameters tuning


To search for the best hyperparameters, GridSearch from sklearn was used.

First, the possible values for each hyperparameter are defined, and the following were considered:



```
# Define the possible values of hyperparameters to tune
param_grid = {
    'max_depth': [5, 10, 15, 20, 25, 35, 50, None],
    'min_samples_split': [2, 5, 10, 15, 20, 25],
    'min_samples_leaf': [1, 2, 4, 6, 8, 10, 15, 20],
    'criterion': ['gini', 'entropy']
}
```

Then, an internal 5-fold cross-validation is created and GridSearch is performed. This should not be confused with the outer evaluation, which also used 5-fold cross-validation. In this case, for each predefined training set, it is further divided into 5 folds, and trees are created using the possible hyperparameter combinations. The hyperparameters with the best average accuracy are finally chosen.



```
# Start the external cross-validation process
for train_idx, test_idx in outer_cv.split(X, y):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

    # Define internal cross-validation with GridSearchCV
    inner_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=inner_cv, scoring='balanced_accuracy', n_jobs=-1)
    ...
```

It should be noted that, as the model's complexity increases, so does the risk of overfitting.

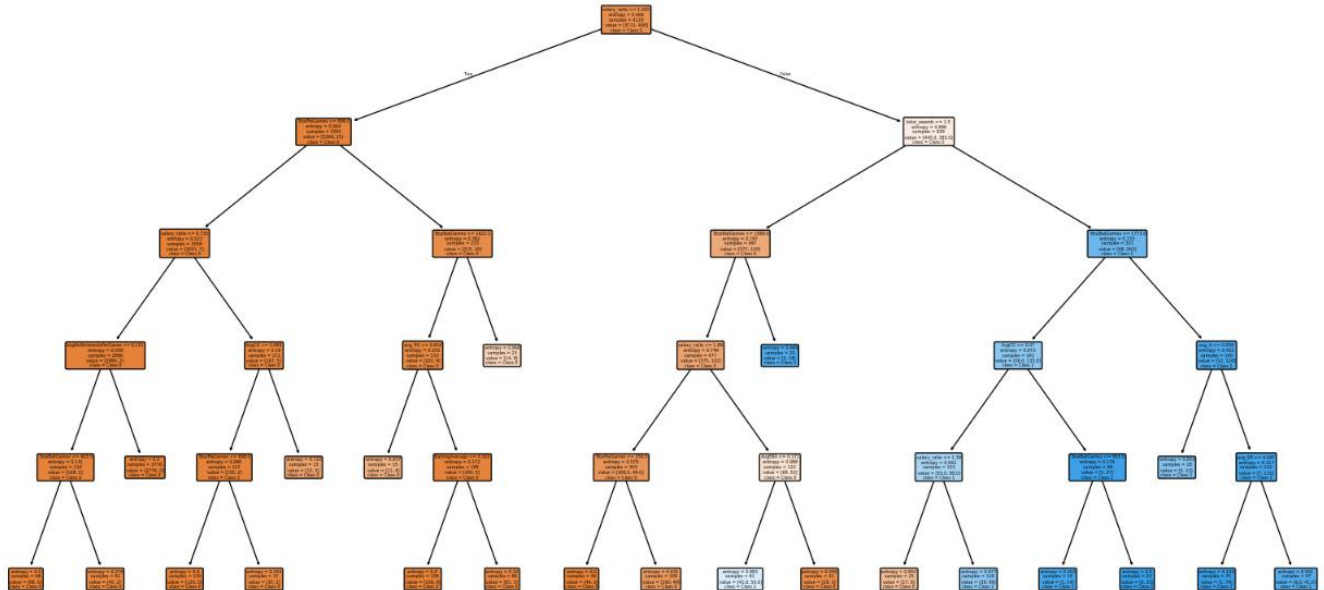
To avoid overfitting, low hyperparameter values (simple models) and high hyperparameter values (complex models) were included. This way, GridSearch will search for the combination that yields the best accuracy. If a model has high overfitting, it won't provide as good accuracy, so it will be discarded.

Additionally, since we used 5-fold cross-validation for the inner evaluation, the evaluator's measure will be robust, which will help discard hyperparameters that lead to overfitting models.

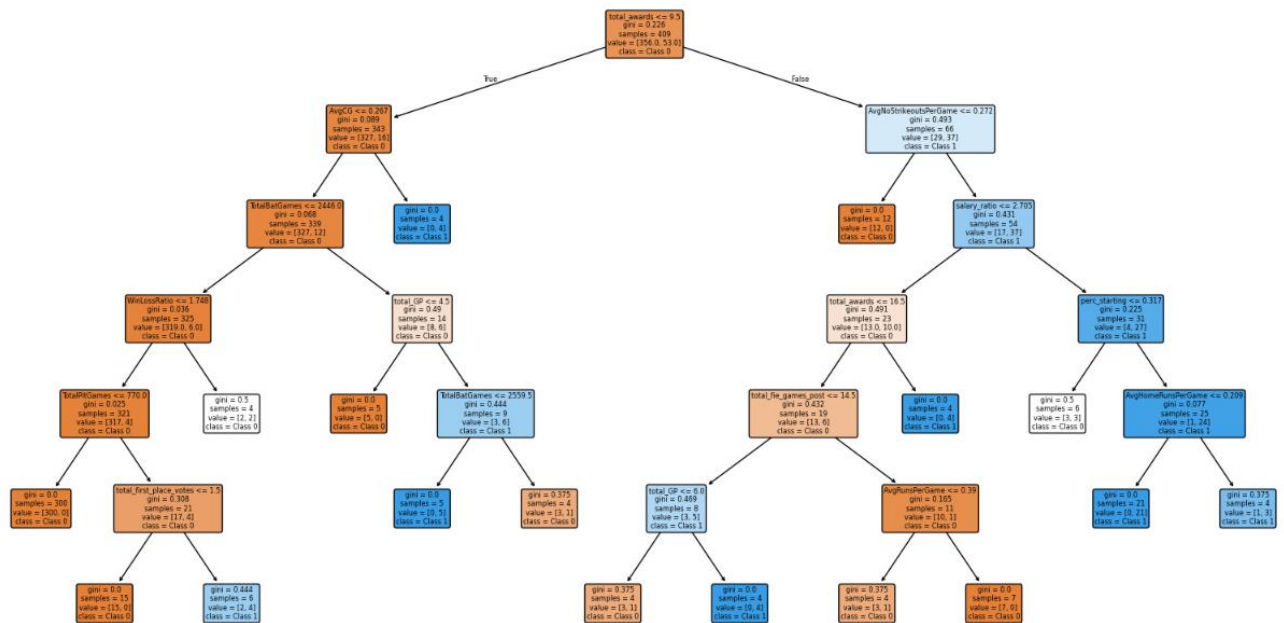
Lastly, it's important to highlight that the independence between the train and test data makes it more difficult for overfitting to occur. If the train and test data were very similar, overfitting models would show very high accuracy.

Decision trees

Decision tree for Task A:



Decision tree for Task B:



Question #12

Question 12: