Predicting the Success of Baseball Pitchers

Yilmaz Machine Learning 1 - 2022-2023

Wilson Chen and Bradley Cao

Contents

1 Introduction **3**
- 1.1 Overview & Rationale . . . 3
- 1.2 Data Set Information . . . 3

2 Preprocessing **4**
- 2.1 Missing and Redundant Values . . . 4
- 2.2 Class Labels . . . 5
- 2.3 Dimensionality Reduction . . . 5
  - 2.3.1 Self Selection . . . 5
  - 2.3.2 WrapperSubsetEval . . . 6
  - 2.3.3 InfoGainAttributeEval . . . 7
  - 2.3.4 GainRatioAttributeEval . . . 8
  - 2.3.5 CorrelationAttributeEval . . . 9

3 Splitting into Testing and Training **9**
- 3.1 SelfSelection . . . 10
- 3.2 WrapperSubsetEval . . . 11
- 3.3 InfoGainAttributeEval . . . 12
- 3.4 GainRatioAttributeEval . . . 13
- 3.5 CorrelationAttributeEval . . . 14

4 Classification **15**
- 4.1 Types of Classification Methods . . . 15
  - 4.1.1 J48 . . . 15
  - 4.1.2 LogisticFunction . . . 15
  - 4.1.3 NaiveBayes . . . 15
  - 4.1.4 OneR . . . 16
- 4.2 Methods . . . 16
- 4.3 SelfSelected . . . 16
  - 4.3.1 J48 - SelfSelected . . . 16
  - 4.3.2 LogisticFunction - SelfSelected . . . 17
  - 4.3.3 NaiveBayes - SelfSelected . . . 19
  - 4.3.4 OneR - SelfSelected . . . 21
  - 4.3.5 Analysis . . . 23
- 4.4 WrapperSubsetEval . . . 24
  - 4.4.1 J48 - WrapperSubsetEval . . . 24
  - 4.4.2 LogisticFunction - WrapperSubsetEval . . . 25
  - 4.4.3 NaiveBayes - WrapperSubsetEval . . . 27
  - 4.4.4 OneR - WrapperSubsetEval . . . 29
  - 4.4.5 Analysis . . . 31
- 4.5 InfoGainAttributeEval . . . 32
  - 4.5.1 J48 - InfoGainAttributeEval . . . 32
  - 4.5.2 LogisticFunction - InfoGainAttributeEval . . . 34

4.5.3 NaiveBayes - InfoGainAttributeEval . . . 36
4.5.4 OneR - InfoGainAttributeEval . . . 38
4.5.5 Analysis . . . 40
4.6 GainRatioAttributeEval . . . 41
4.6.1 J48 - GainRatioAttributeEval . . . 41
4.6.2 LogisticFunction - GainRatioAttributeEval . . . 43
4.6.3 NaiveBayes - GainRatioAttributeEval . . . 45
4.6.4 OneR - GainRatioAttributeEval . . . 47
4.6.5 Analysis . . . 49
4.7 InfoGainAttributeEval . . . 50
4.7.1 J48 - InfoGainAttributeEval . . . 50
4.7.2 LogisticFunction - InfoGainAttributeEval . . . 52
4.7.3 Analysis . . . 54
4.7.4 NaiveBayes - CorrelationAttributeEval . . . 55
4.7.5 OneR - CorrelationAttributeEval . . . 57
4.7.6 Analysis . . . 59

5 Conclusion **59**

6 Discussion **59**

7 Sources **60**

8 Bibliography **61**

1 Introduction

1.1 Overview & Rationale

Major League Baseball (MLB) has become more and more statstics-driven over the years. Coaches use these statistics to evaluate player performance and make informed decisions about players, for example in draft reports. Contracts in MLB are often worth millions of dollars, so having statistics that measure player performance is extremely important. We wanted to see just how accurate predictions with these statistics are. For our project, we decided to look at a dataset on baseball pitchers. This dataset with 547 instances and 33 attributes contained information on different statistics recorded for pitchers. It came from the 2021 regular season, and only contained pitchers who had faced at least 100 batters. We wanted to see if we could use these statistics to predict whether or not a pitcher had a winning record.

1.2 Data Set Information

We got our original dataset from https://www.baseballsavant.mlb.com. It contained 33 attributes:

- **last_name**
- **first_name**
- **player_id**
- **year**
- **p_game**
- **p_total_hits**
- **p_home_run**
- **p_strikeout**
- **p_walk**
- **p_k_percent**
- **p_bb_percent**
- **batting_avg**
- **slg_percent**
- **on_base_percent**
- **on_base_plus_slg**
- **p_earned_run**

- **p_win**
- **p_loss**
- **p_era**
- **p_rbi**
- **p_called_strike**
- **p_unearned_run**
- **exit_velocity_avg**
- **launch_angle_avg**
- **sweet_spot_percent**
- **barrel_batted_rate**
- **hard_hit_percent**
- **meatball_percent**
- **pitch_hand**
- **fastball_avg_speed**
- **n_fastball_formatted**
- **offspeed_avg_speed**
- **n_offspeed_formatted**

These statistics were collected by MLB or MLBStatCast, a partner company that specializes in collecting data from MLB games.

2 Preprocessing

2.1 Missing and Redundant Values

We first started by removing redundant values, such as OPS. The OPS attribute is just the sum of the On Base Percentage (**OBP**) and Slugging (**SLG**), so we can safely remove it without losing any information. Furthermore, there were missing values for the attributes Offspeed% and Offspeed Average MPH. We decided to remove these attributes altogether, as these values are missing because not all pitchers have an offspeed pitch; filling this data with an average would not be representative, and removing these attributes helps with dimensionality reduction. We also removed all attributes that did not relate to pitcher's performance (**name**, **year**, **player ID**, etc.). We can safely remove these attributes since they provide zero information.

2.2 Class Labels

Since our data did not come with class labels, we had to create them ourselves. Using the **p_win** and **p_loss**, we created our own column, titled **winning** to keep track of whether or not the pitcher had a winning record. For pitchers that had an even record, with the same number of wins and losses, we classified them as having a winning record. We then removed the attributes **p_win** and **p_loss** because the information they give us is given by our new class variable.

2.3 Dimensionality Reduction

2.3.1 Self Selection

The first method we used was self selection. We picked 6 attributes that we believed, with our knowledge of baseball, would be the best indicators of a pitcher having a winning record.

2.3.2 WrapperSubsetEval

WrapperSubsetEval is the first of the WEKA built-in attribute selection methods. It uses a learning scheme to evaluate attribute sets, and uses cross validation to estimate its accuracy for each set.

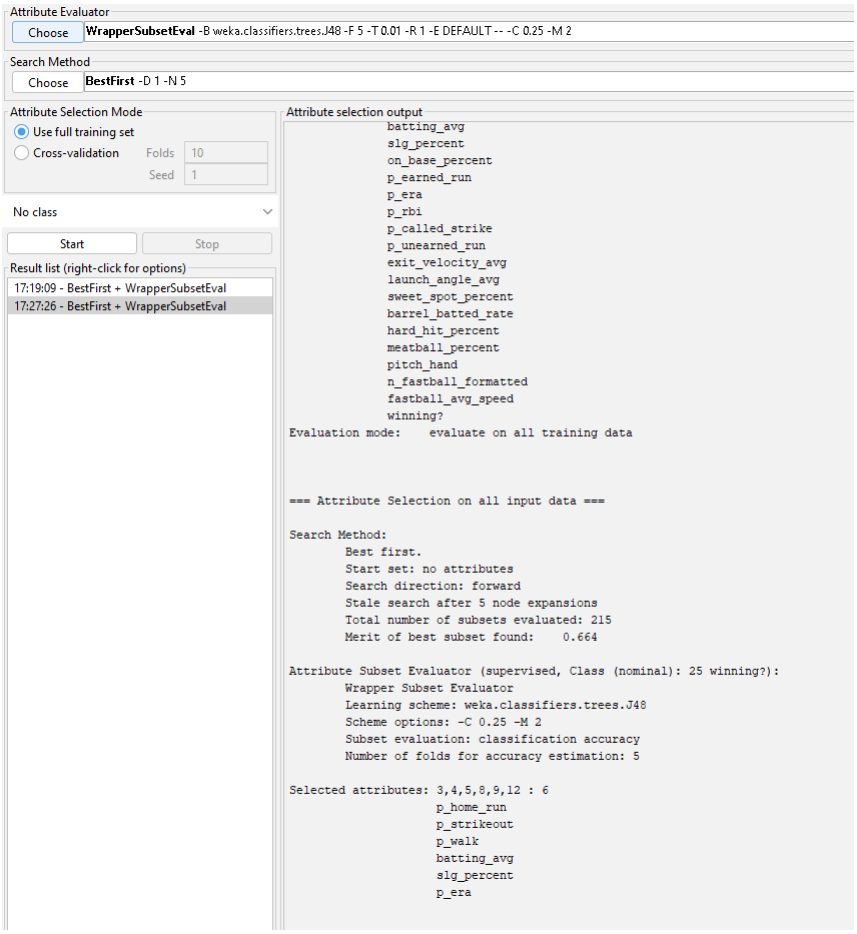


Figure 1: WEKA screenshot showing the results of WrapperSubsetEval

2.3.3 InfoGainAttributeEval

InfoGainAttributeEval is the second WEKA built-in attribute selection method. It measures the information gain of each attribute relative to the class variable and gives us the ones with the most information gain, which are the most predictive of the class variable.

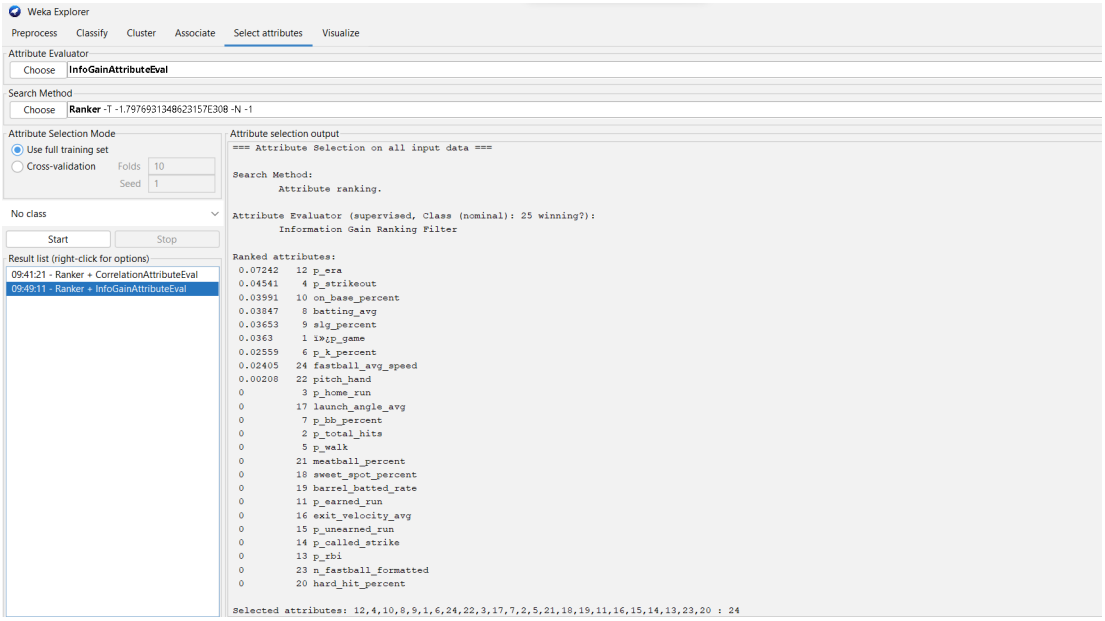


Figure 2: WEKA screenshot showing the results of InfoGainAttributeEval

2.3.4 GainRatioAttributeEval

GainRatioAttributeEval is the third WEKA built-in attribute selection method. This method is similar to the previous, but instead measures the gain ratio for each attribute relative to the class variable. The gain ratio is the ratio of information gain to intrinsic information.

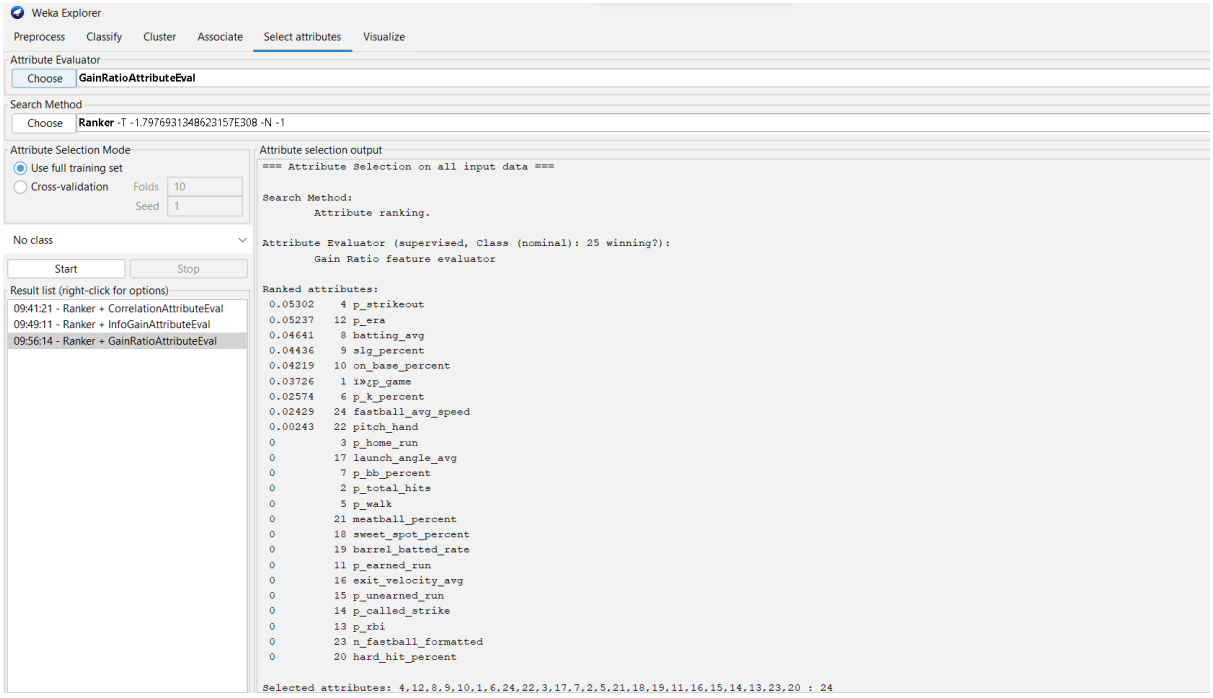


Figure 3: WEKA screenshot showing the results of GainRatioAttributeEval

2.3.5 CorrelationAttributeEval

CorrelationAttributeEval is the final WEKA built-in attribute selection method we used. This method measures the Pearson's correlation between an attribute and the class variable. The Pearson's correlation measures the strength of a linear relationship between two variables, and can range from -1 to 1.

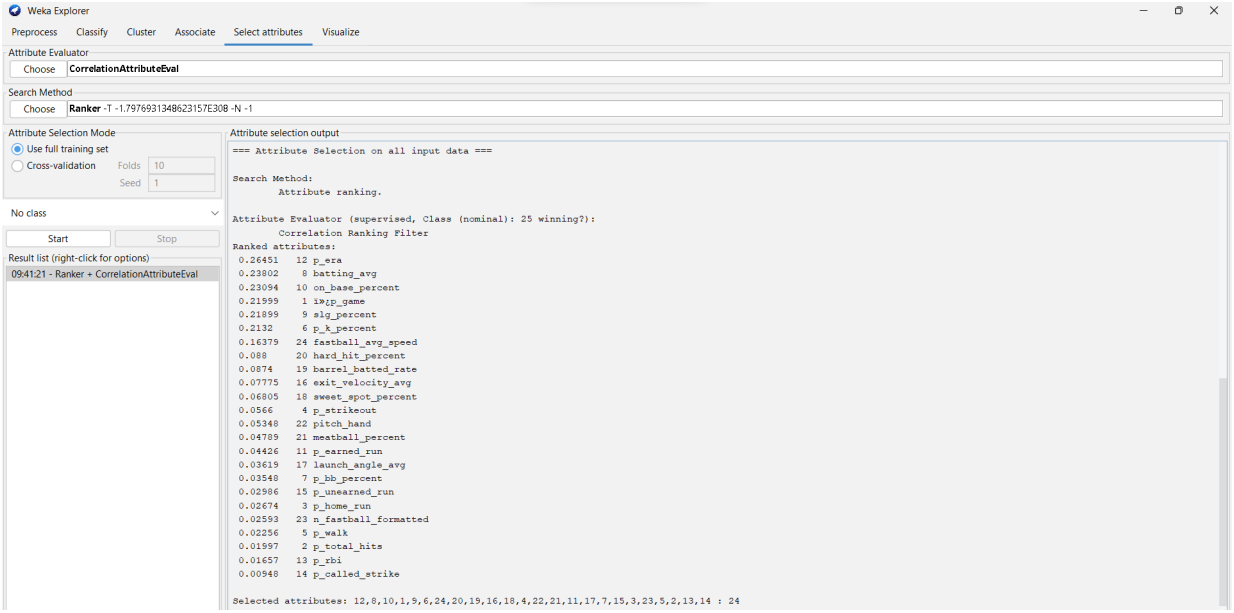


Figure 4: WEKA screenshot showing the results of CorrelationAttributeEval

3 Splitting into Testing and Training

For each one of these attribute selection methods, we removed all attributes except the ones selected and the class variable. We then split the instances into a testing set and a training set using WEKA. For this, we used WEKA's supervised filter **Resample**, which we configured to produce a random subset of the dataset without replacement. We ran it once, with the size of the output dataset to be 33% of the whole to get the testing sets, then used the **InvertSelection** feature to get the training sets. The training sets had 368 instances, with 226 classified as **yes** and 142 classified as **no**, while the testing sets had 179 instances, with 110 classified as **yes** and 69 classified as **no**.

3.1 SelfSelection

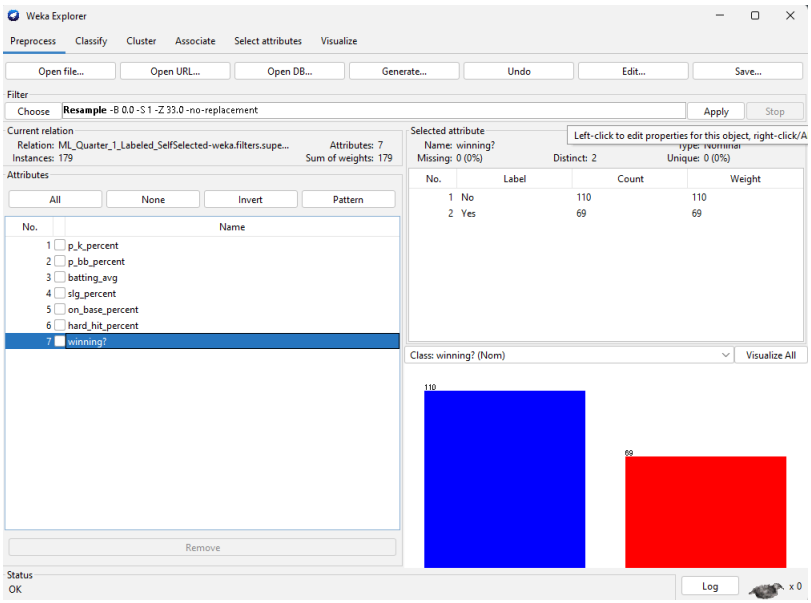


Figure 5: SelfSelection Test Set

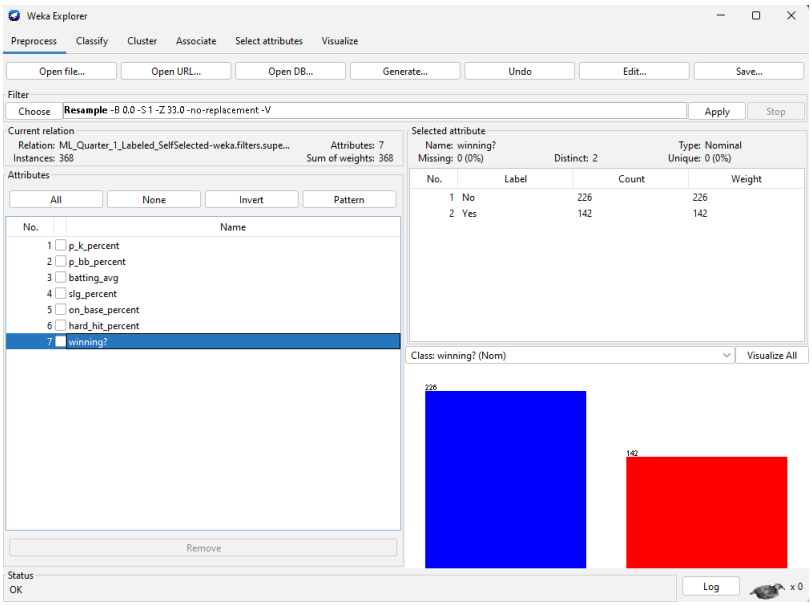


Figure 6: SelfSelection Train Set

3.2 WrapperSubsetEval

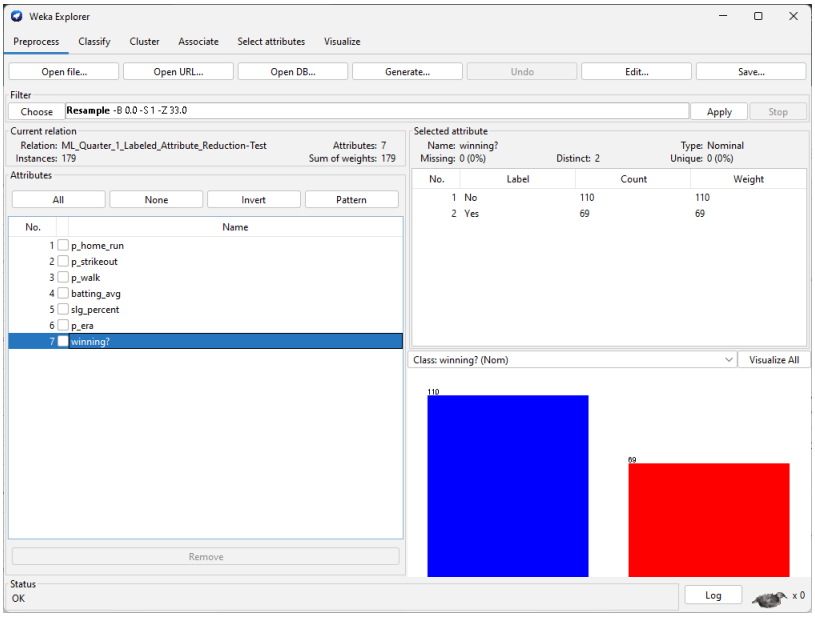


Figure 7: WrapperSubsetEval Test Set

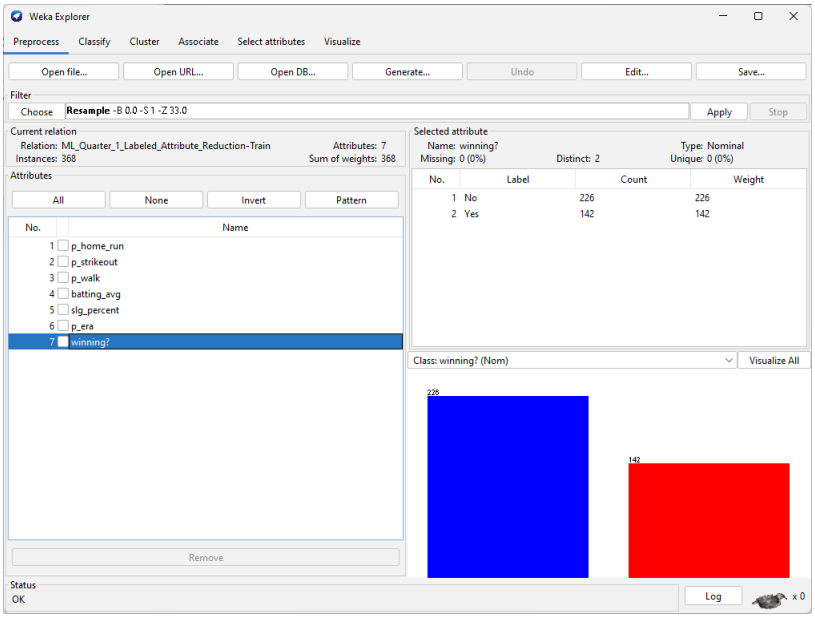


Figure 8: WrapperSubsetEval Train Set

3.3 InfoGainAttributeEval

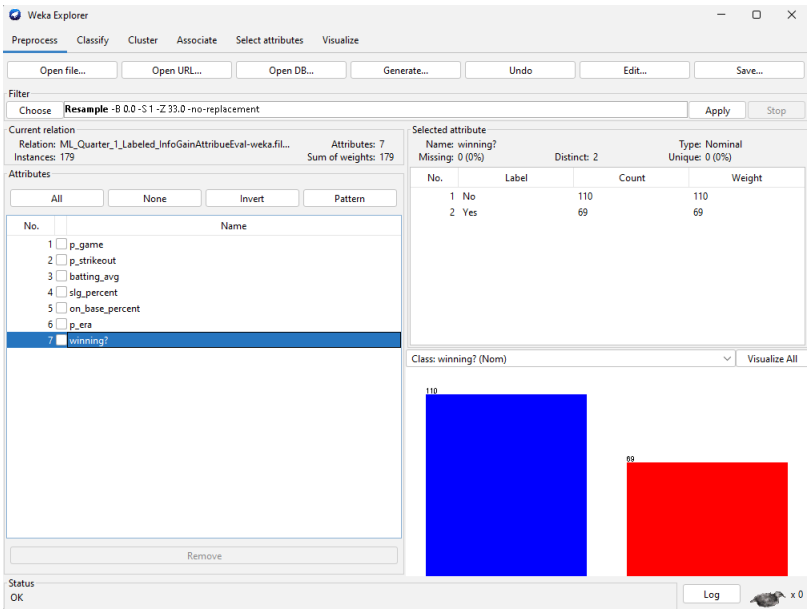


Figure 9: InfoGainAttributeEval Test Set

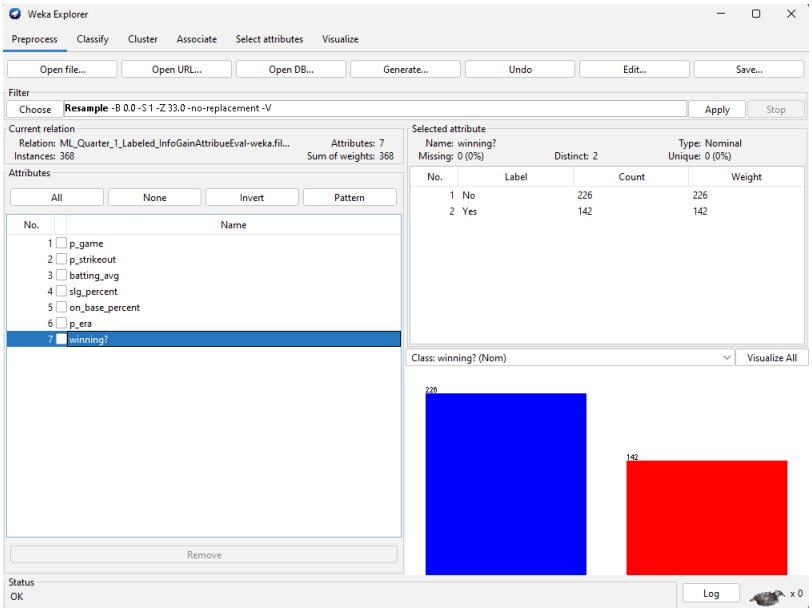


Figure 10: InfoGainAttributeEval Train Set

3.4 GainRatioAttributeEval

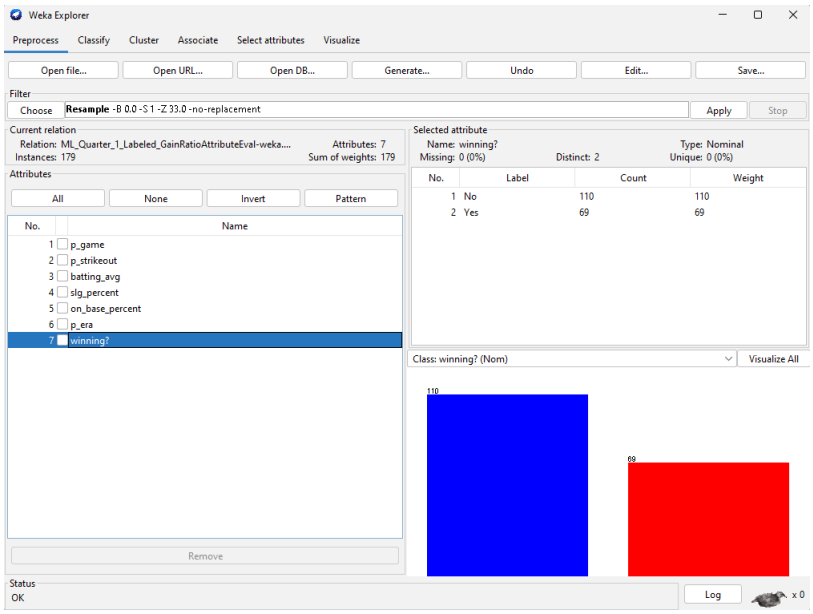


Figure 11: GainRatioAttributeEval Test Set

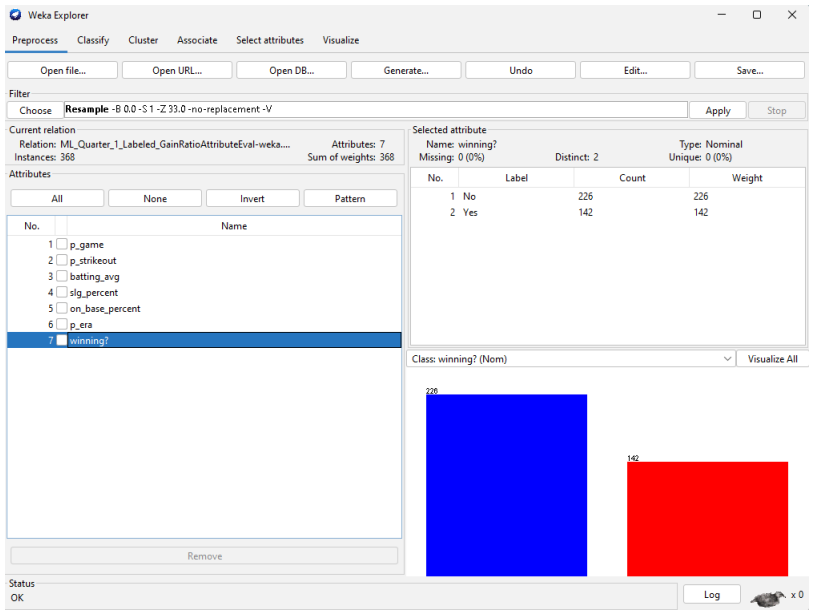


Figure 12: GainRatioAttributeEval Train Set

3.5 CorrelationAttributeEval

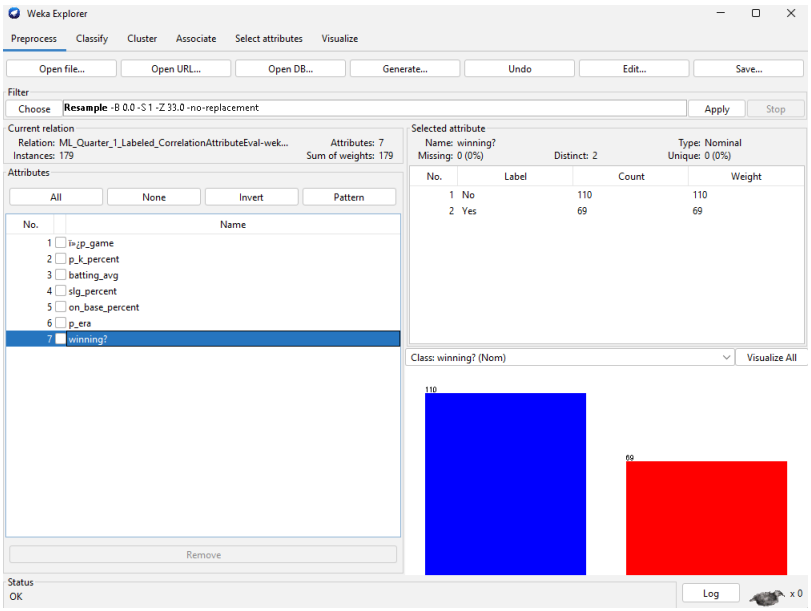


Figure 13: CorrelationAttributeEval Test Set

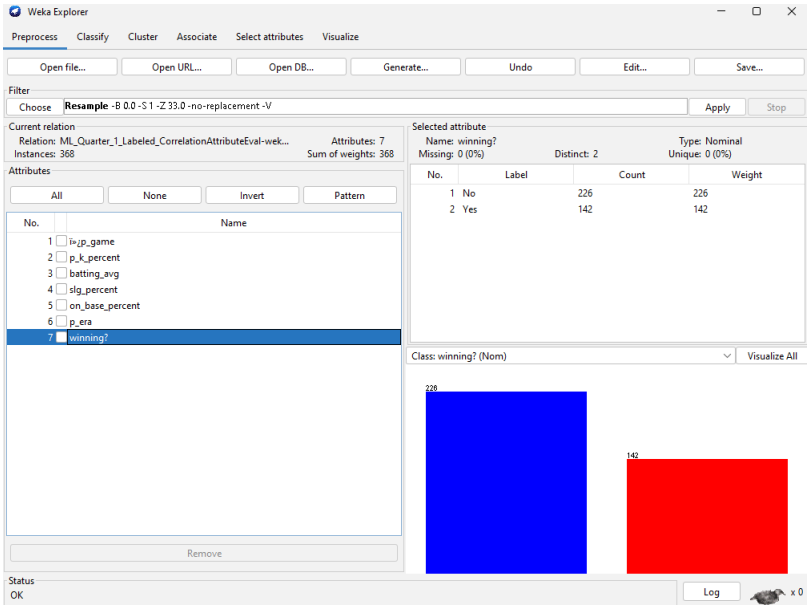


Figure 14: CorrelationAttributeEval Train Set

4 Classification

For each method of attribute reduction, we ran 4 classification methods using WEKA. These methods included:

- **J48**
- **LogisticFunction**
- **NaiveBayes**
- **OneR**

We compared each of these methods to determine which one worked the best for each attribute reduction method. They were trained using supplied training sets we created earlier. We then tested each classification method on each of the testing sets that we created, running 20 classifications in total.

4.1 Types of Classification Methods

4.1.1 J48

J48 classification is WEKA's implementation of the C4.5 algorithm, where the J stands for Java. It builds a decision tree based on information gain. At each node of the decision tree, it splits the attributes into subsets, with the split being decided by the difference in entropy. **J48** is seen as an improvement over the C4.5 algorithm because it accounts for missing values, allows for pruning, continuous value ranges, and many other extra features.

4.1.2 LogisticFunction

LogisticFunction is most commonly used when the class variable is binary, which, in our case, it is. It uses the sigmoid function,

$$S(x) = \frac{1}{1 + e^{-x}}.$$

Initial weights are modified using the error, and the coefficients are improved through each instance in the test set. We then can use these coefficients to make predictions. A decision boundary is set, like $y = 0.5$, and everything above the decision boundary is rounded to a decision of 1 and everything below the decision boundary is rounded to a decision of 0. These decisions are mapped to the class variable.

4.1.3 NaiveBayes

NaiveBayes is based on Bayes'theorem, or

$$\mathbf{P}(A|B) = \frac{\mathbf{P}(B|A)\mathbf{P}(A)}{\mathbf{P}(B)}.$$

It assumes that each attribute is independent of the rest, making the train time much faster. This makes it a good baseline to compare other classification methods to. It repeatedly applies Bayes'theorem with these naive assumptions.

4.1.4 OneR

OneR, also known as **One Class** classification, looks at just one attribute to make a prediction about the instance. It uses the training set to determine which attribute is the best predictor of the class variable, determined with accuracy. It is an extremely lightweight classification algorithm, sacrificing overall accuracy for speed. Like **NaiveBayes**, it can provide a baseline for other predictors.

4.2 Methods

We ran each of the 4 classification methods in WEKA on each of our resulting datasets from our 5 attribute selection processes, giving us 20 classifications run total. We then analyzed the results from each attribute selection method individually, then together as a large group.

4.3 SelfSelected

4.3.1 J48 - SelfSelected

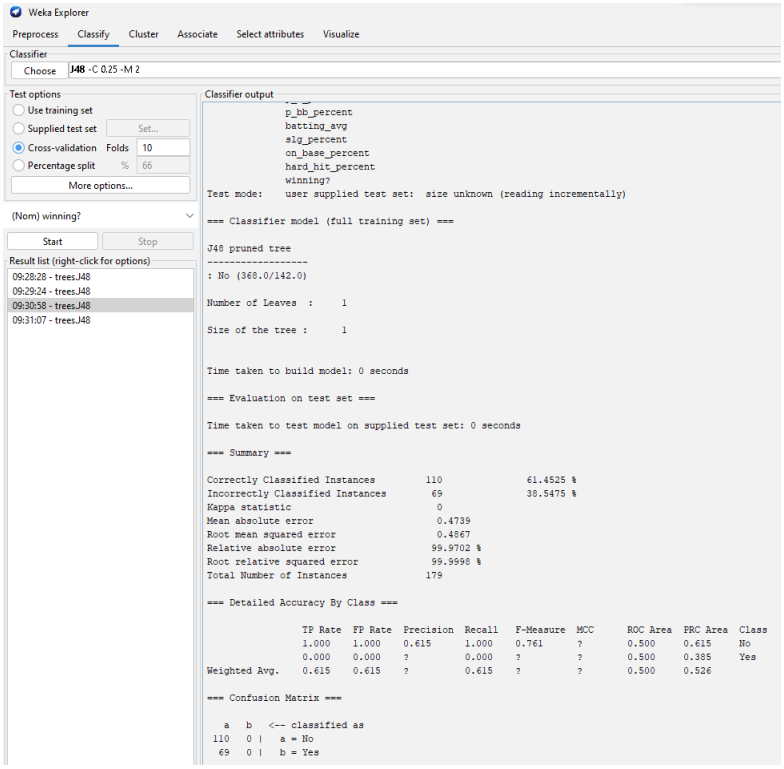


Figure 15: WEKA J48 Classification Results on SelfSelected Attributes

4.3.2 LogisticFunction - SelfSelected

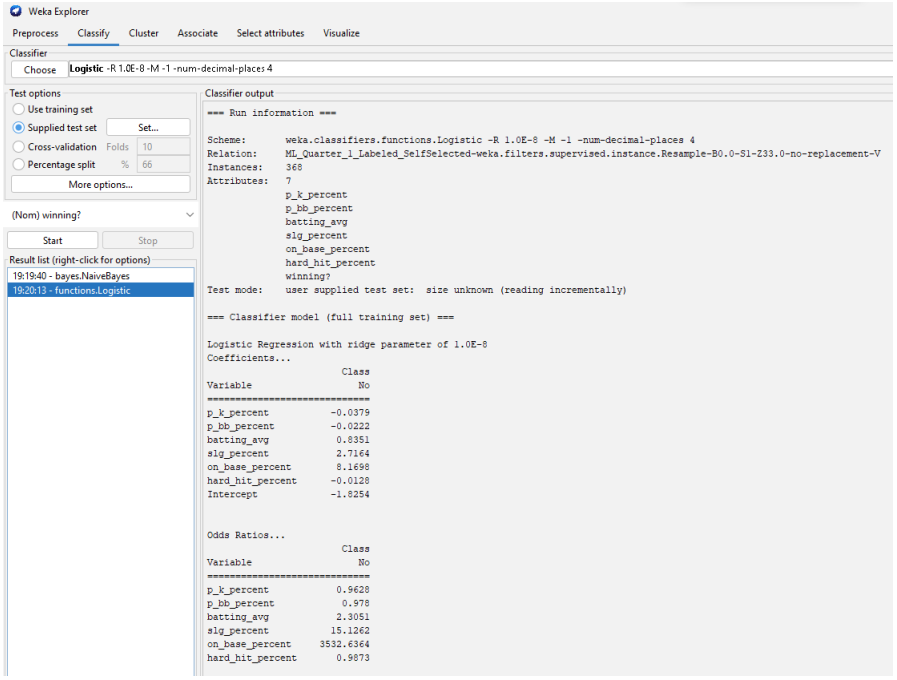


Figure 16: WEKA LogisticFunction Model on SelfSelected Attributes

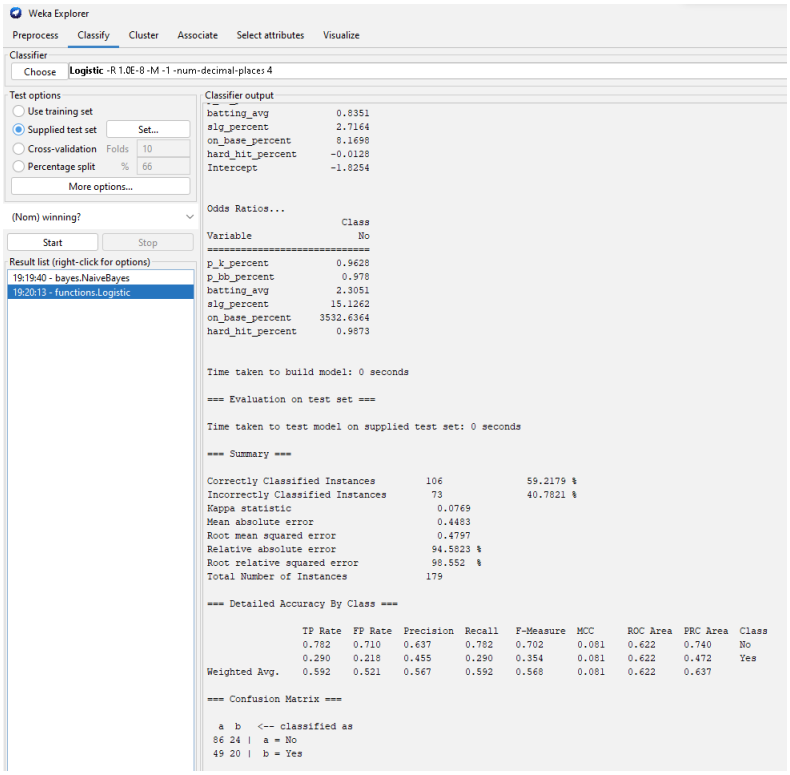


Figure 17: WEKA LogisticFunction Summary Results on SelfSelected Attributes

4.3.3 NaiveBayes - SelfSelected

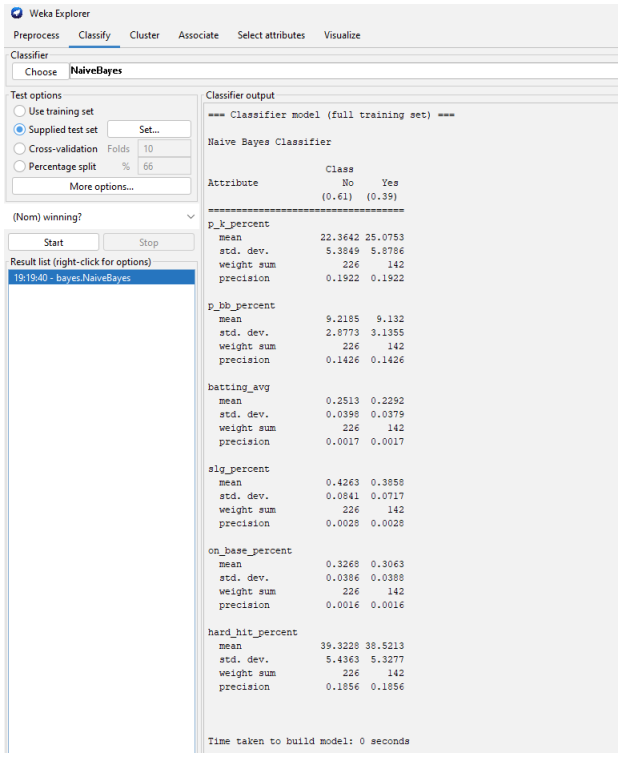


Figure 18: WEKA NaiveBayes Model on SelfSelected Attributes

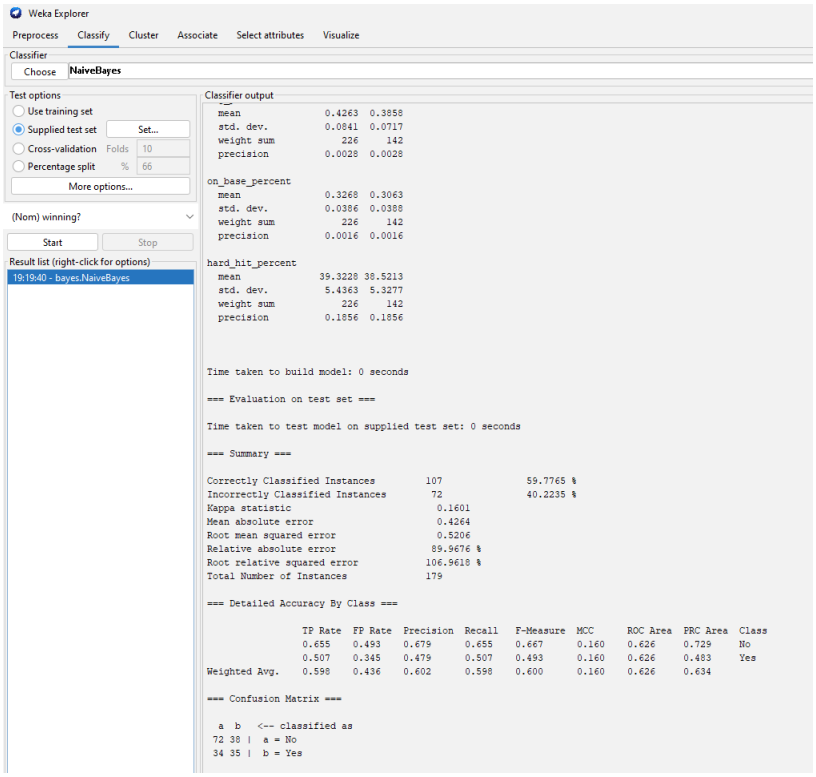


Figure 19: WEKA NaiveBayes Summary Results on SelfSelected Attributes

4.3.4 OneR - SelfSelected

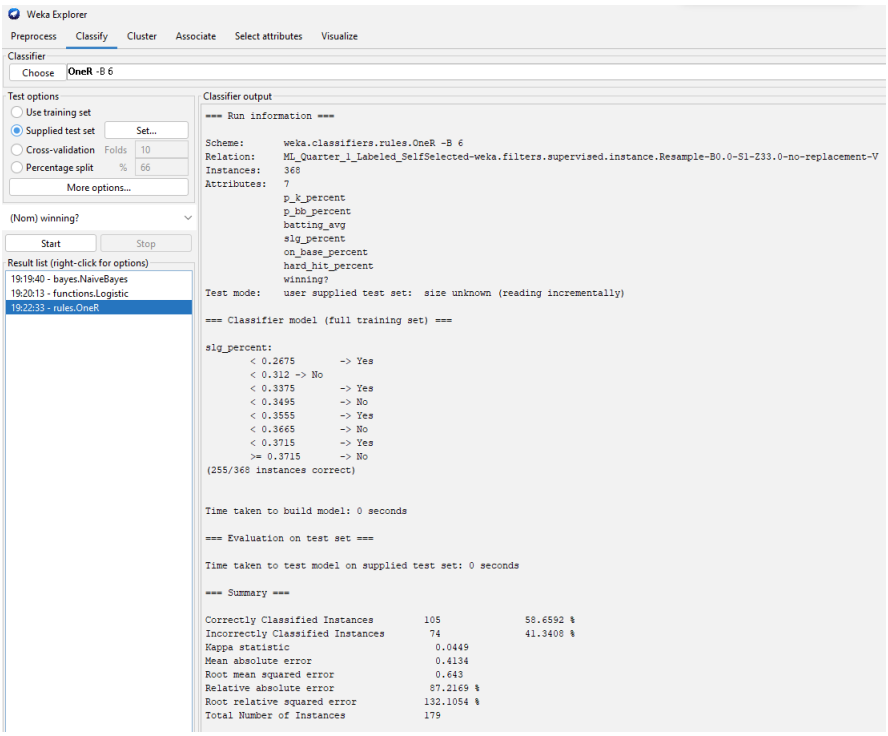


Figure 20: WEKA OneR Model on SelfSelected Attributes

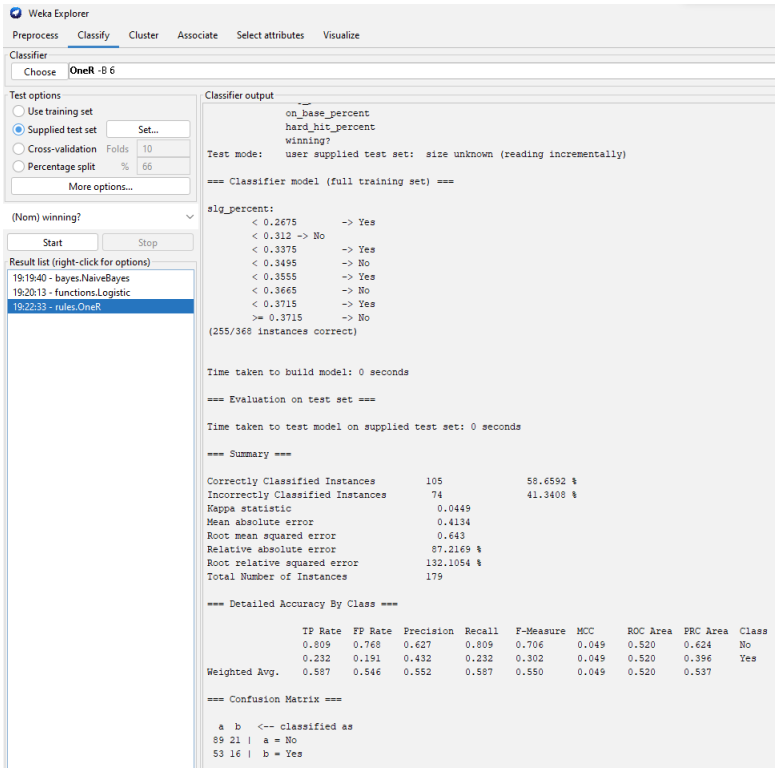


Figure 21: WEKA OneR Summary Results on SelfSelected Attributes

4.3.5 Analysis

We can see that overall **J48** performed with the highest accuracy. However, one thing to note is that there are no true negatives for **J48** classification.

4.4 WrapperSubsetEval

4.4.1 J48 - WrapperSubsetEval

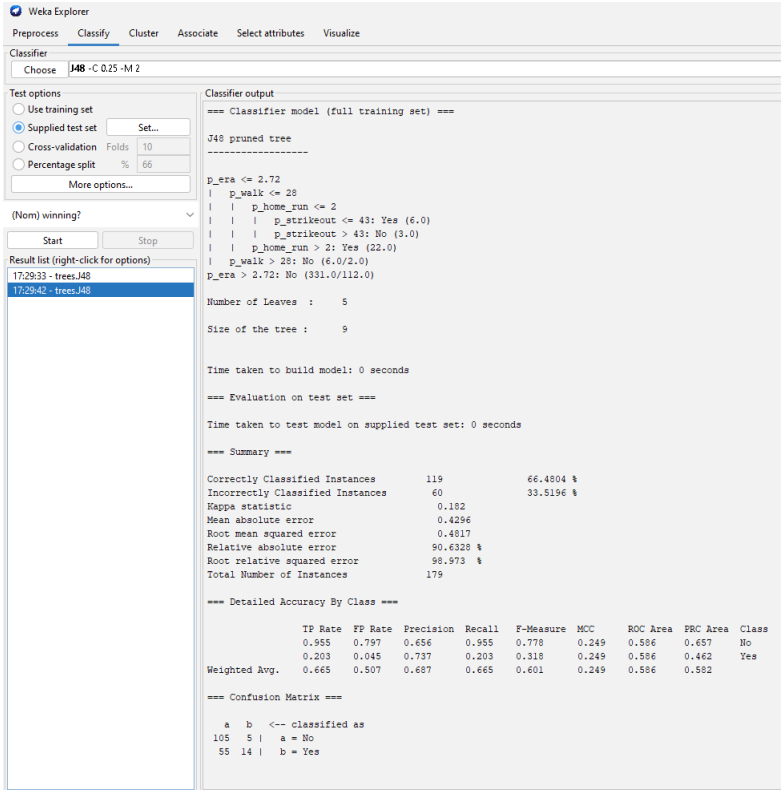


Figure 22: WEKA J48 Classification Results on WrapperSubsetEval Attributes

4.4.2 LogisticFunction - WrapperSubsetEval

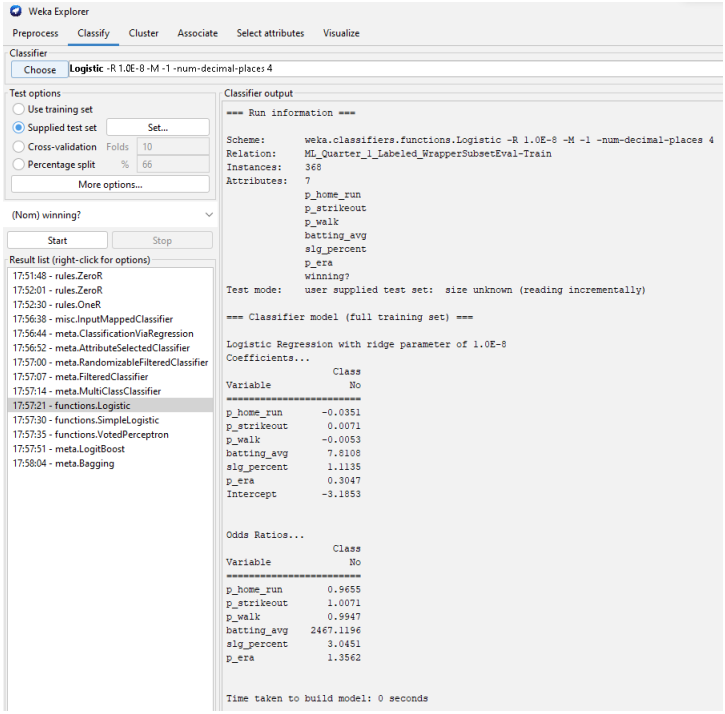


Figure 23: WEKA LogisticFunction Model on WrapperSubsetEval Attributes

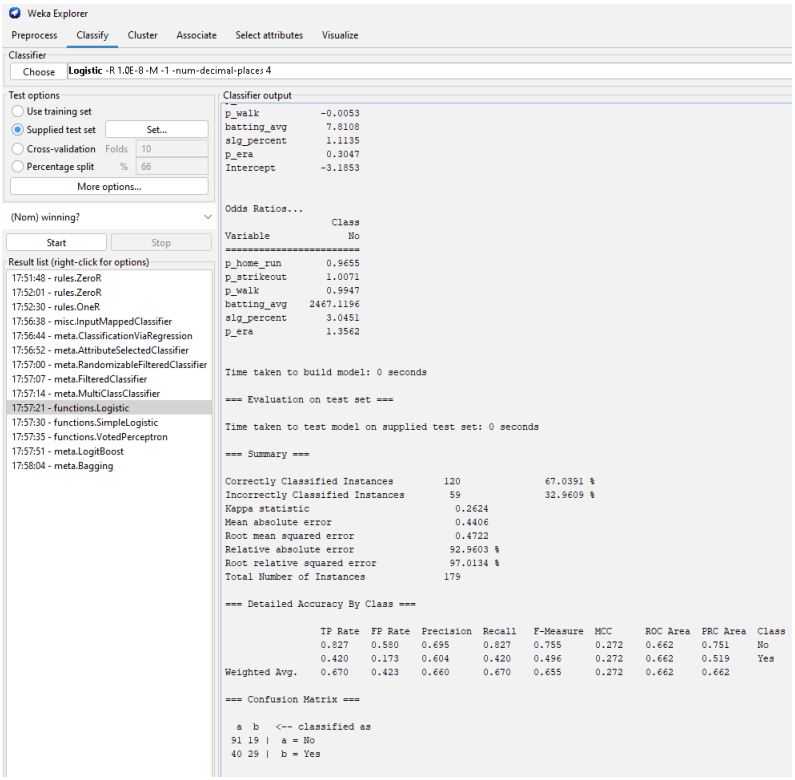


Figure 24: WEKA LogisticFunction Summary Results on WrapperSubsetEval Attributes

4.4.3 NaiveBayes - WrapperSubsetEval

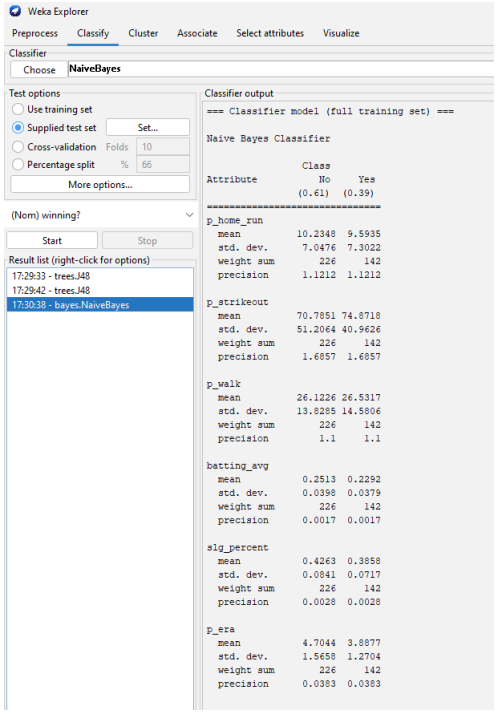


Figure 25: WEKA NaiveBayes Model on WrapperSubsetEval Attributes

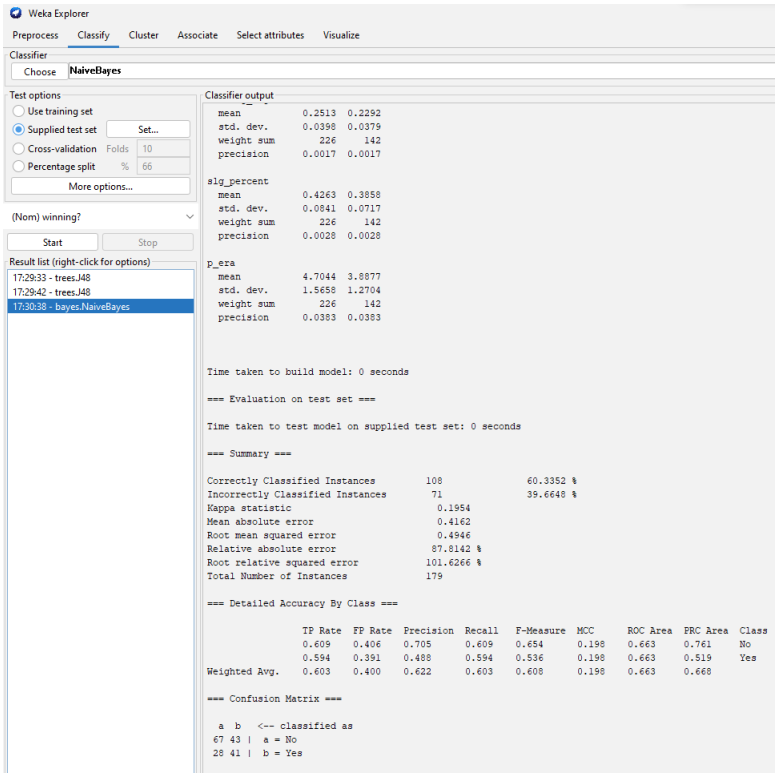


Figure 26: WEKA NaiveBayes Summary Results on WrapperSubsetEval Attributes

4.4.4 OneR - WrapperSubsetEval

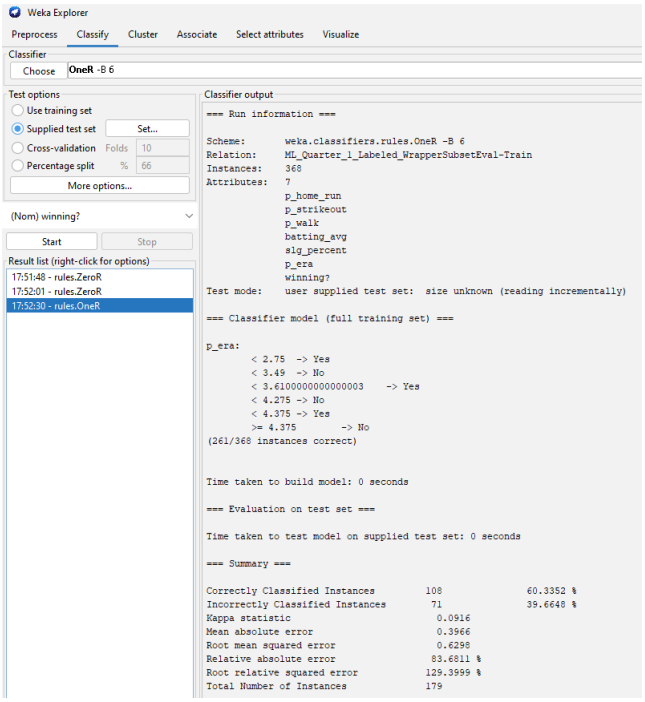


Figure 27: WEKA OneR Model on WrapperSubsetEval Attributes

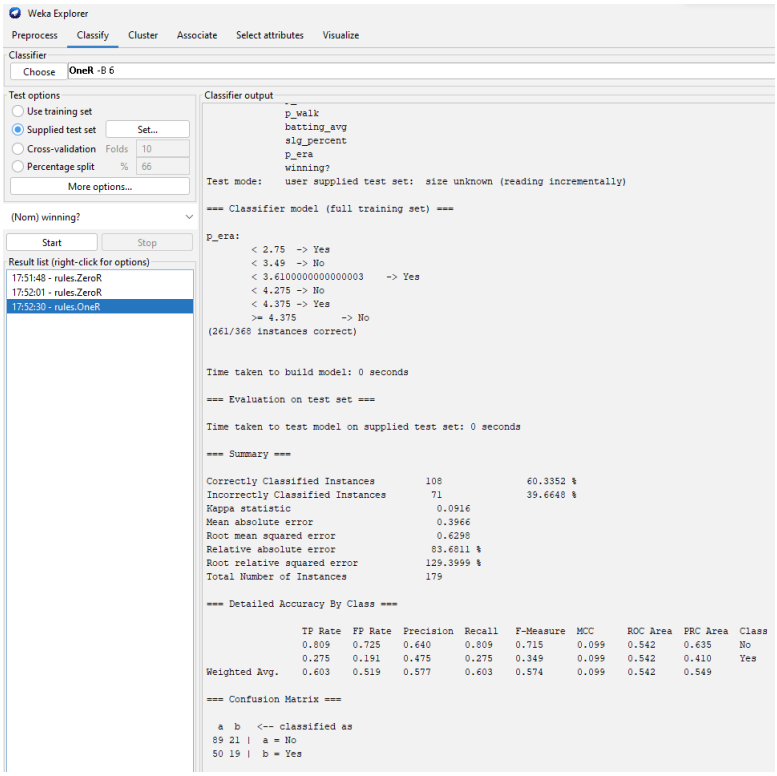


Figure 28: WEKA OneR Summary Results on WrapperSubsetEval Attributes

4.4.5 Analysis

We see that **LogisticFunction** performed best here, but **J48** was a close second with only one more instance misclassified. **J48** was a lot better at determining true positives while **LogisticFunction** was a lot better at determining true negatives.

4.5 InfoGainAttributeEval

4.5.1 J48 - InfoGainAttributeEval

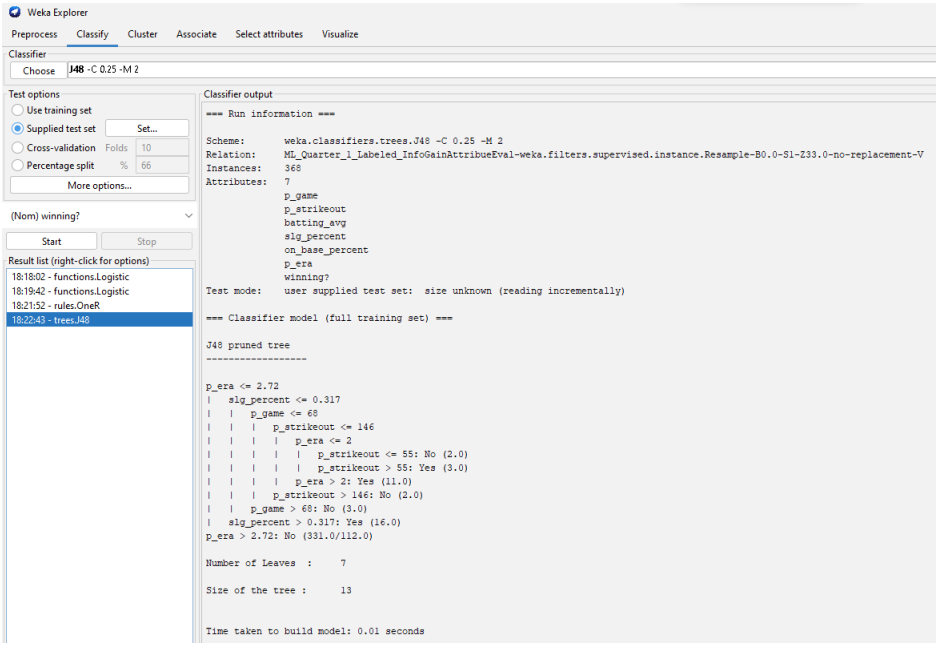


Figure 29: WEKA J48 Model on InfoGainAttributeEval Attributes

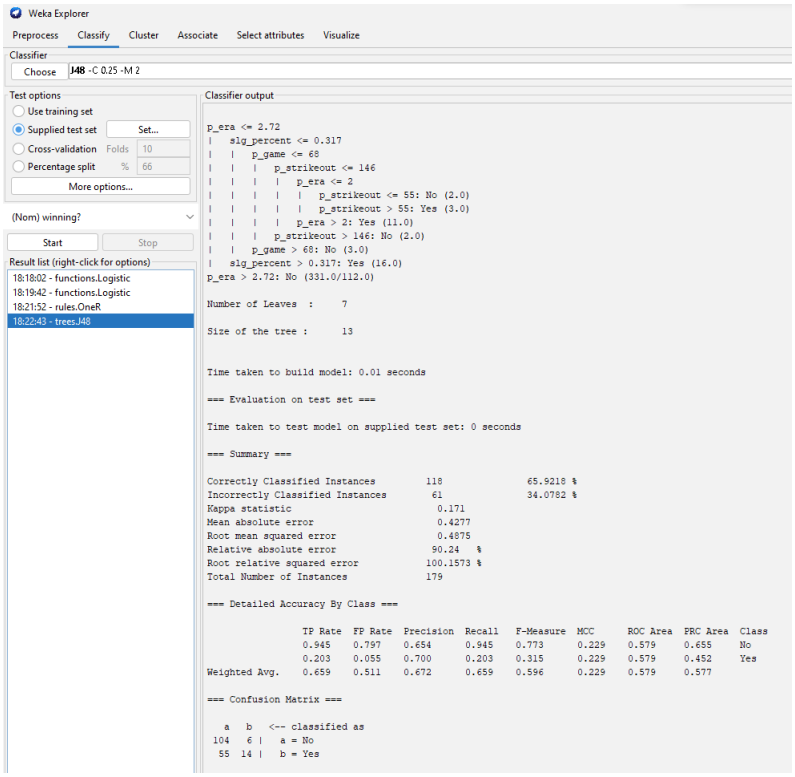


Figure 30: WEKA J48 Classification Results on InfoGainAttributeEval Attributes

4.5.2 LogisticFunction - InfoGainAttributeEval

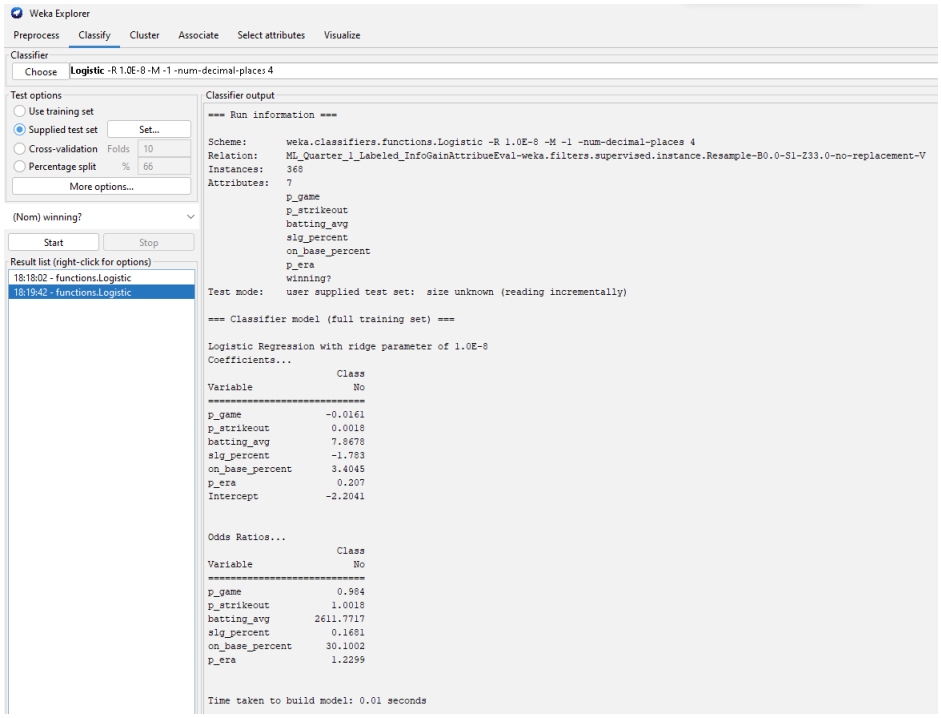


Figure 31: WEKA LogisticFunction Model on InfoGainAttributeEval Attributes

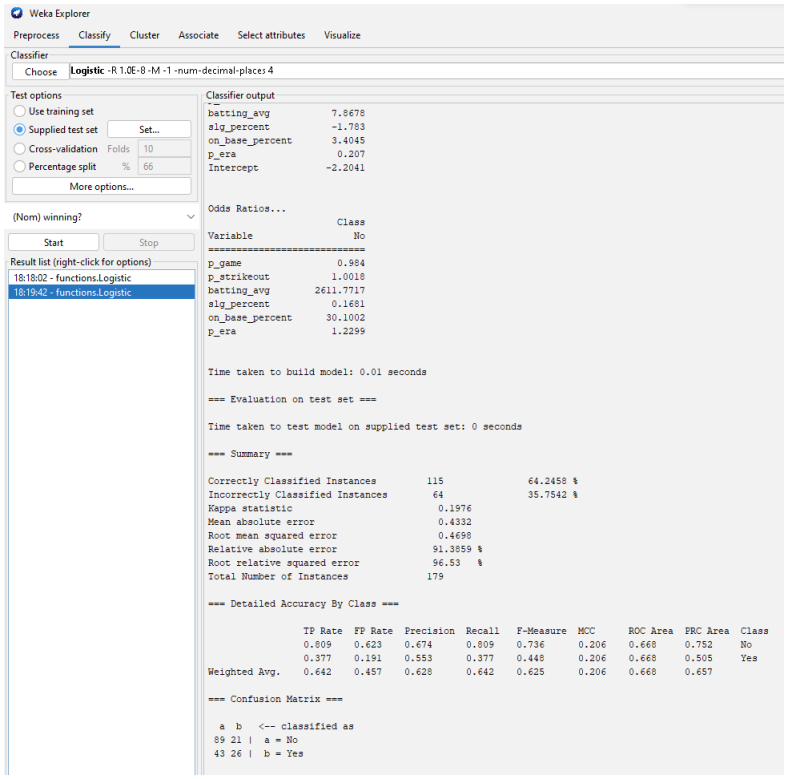


Figure 32: WEKA LogisticFunction Summary Results on InfoGainAttributeEval Attributes

4.5.3 NaiveBayes - InfoGainAttributeEval

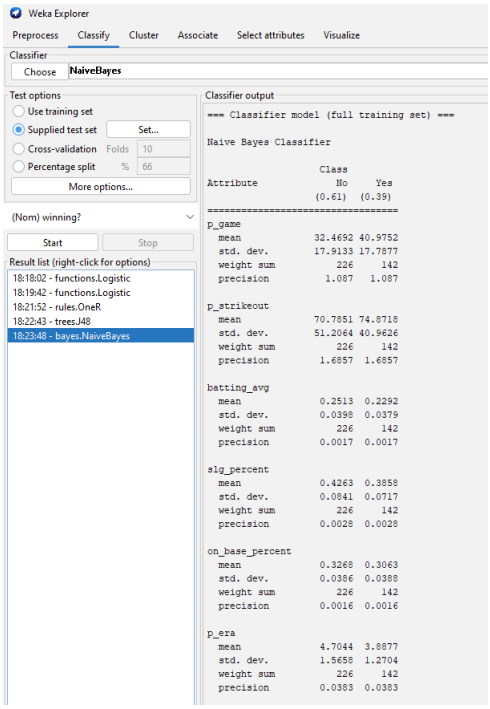


Figure 33: WEKA NaiveBayes Model on InfoGainAttributeEval Attributes

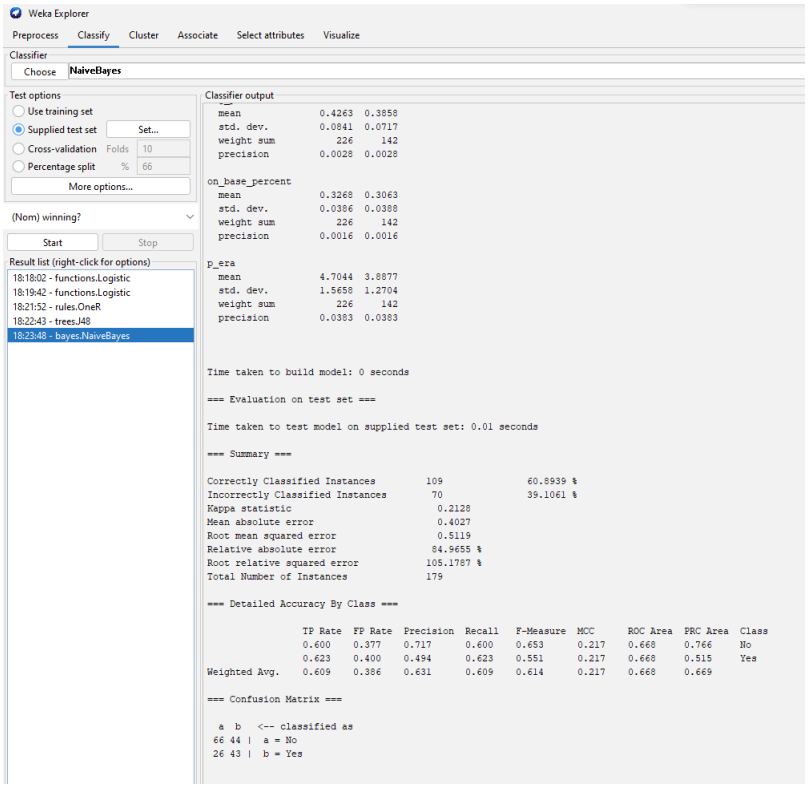


Figure 34: WEKA NaiveBayes Summary Results on InfoGainAttributeEval Attributes

4.5.4 OneR - InfoGainAttributeEval

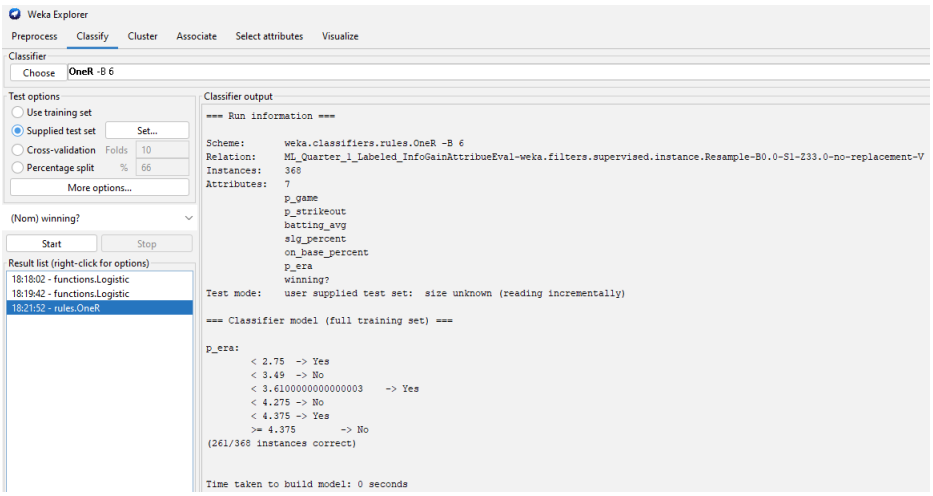


Figure 35: WEKA OneR Model on InfoGainAttributeEval Attributes

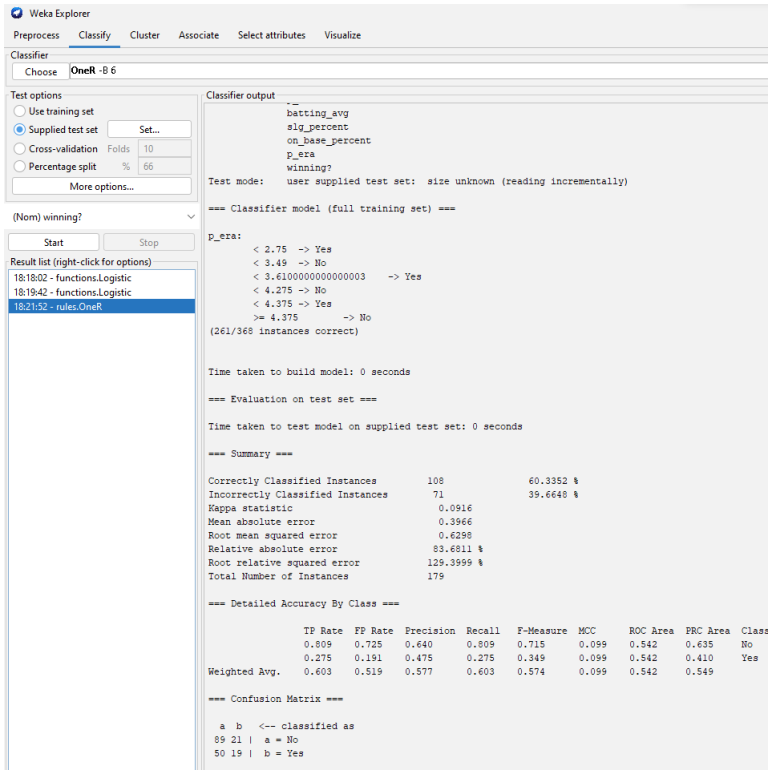


Figure 36: WEKA OneR Summary Results on InfoGainAttributeEval Attributes

4.5.5 Analysis

J48 performed the best with this attribute selection method, with **LogisticFunction** being a close second. Once again, **J48** was a lot better at categorizing true positives while **LogisticFunction** was a lot better at categorizing true negatives.

4.6 GainRatioAttributeEval

4.6.1 J48 - GainRatioAttributeEval

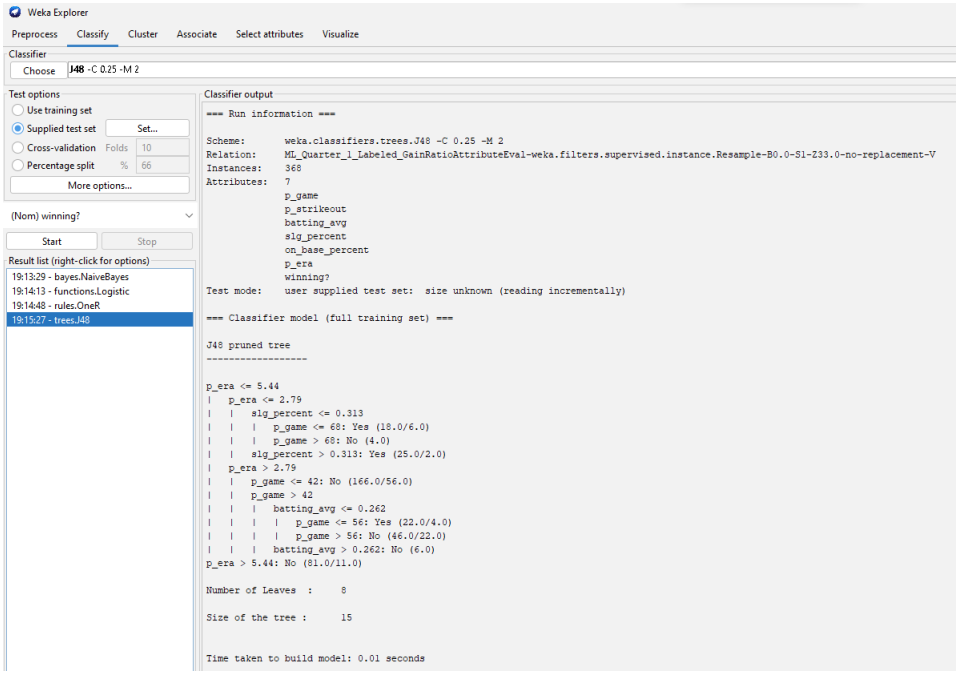


Figure 37: WEKA J48 Model on GainRatioAttributeEval Attributes

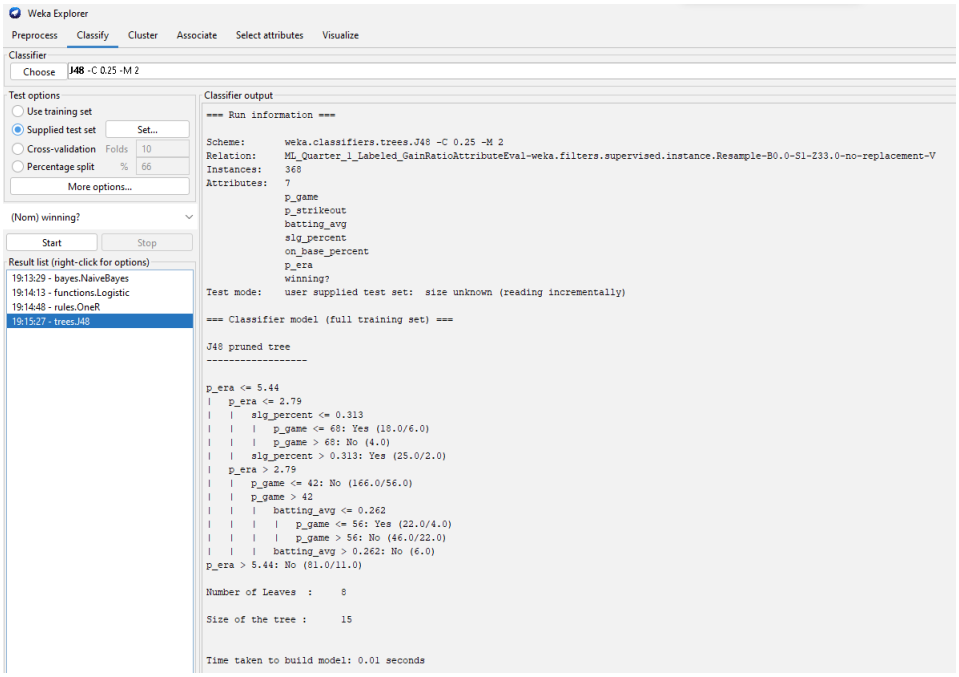


Figure 38: WEKA J48 Classification Results on GainRatioAttributeEval Attributes

4.6.2 LogisticFunction - GainRatioAttributeEval

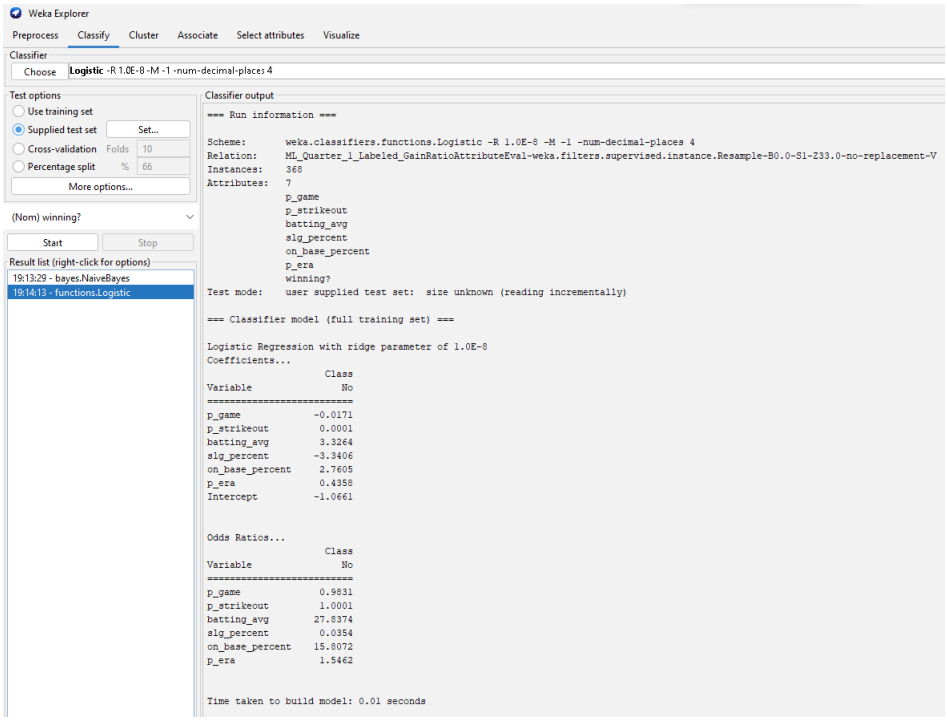


Figure 39: WEKA LogisticFunction Model on GainRatioAttributeEval Attributes

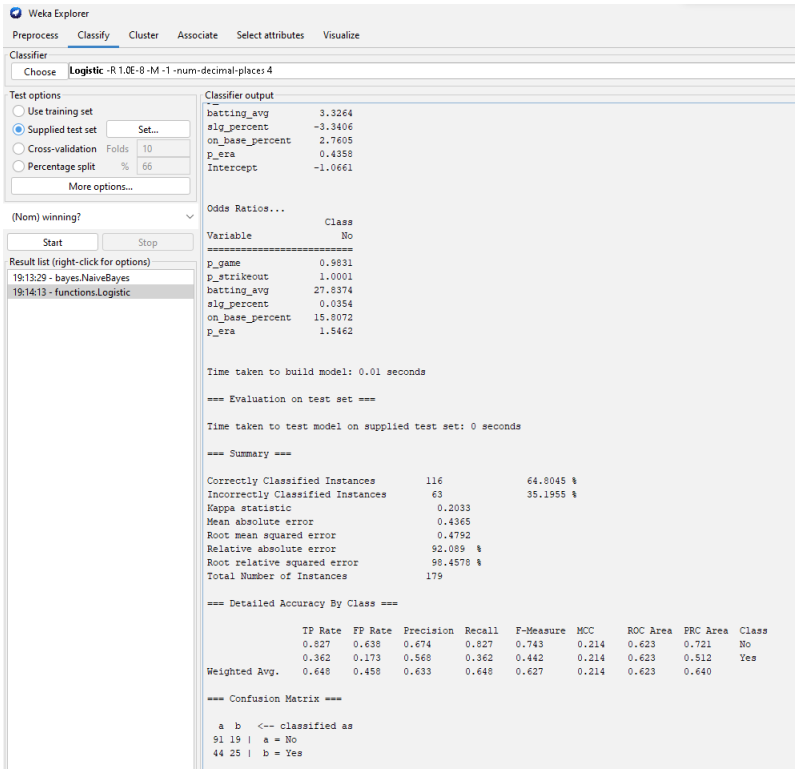


Figure 40: WEKA LogisticFunction Summary Results on GainRatioAttributeEval Attributes

4.6.3 NaiveBayes - GainRatioAttributeEval

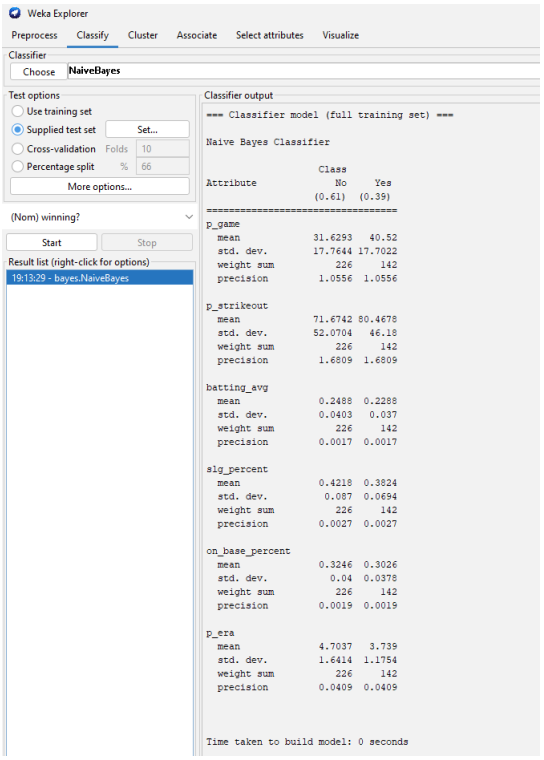


Figure 41: WEKA NaiveBayes Model on GainRatioAttributeEval Attributes

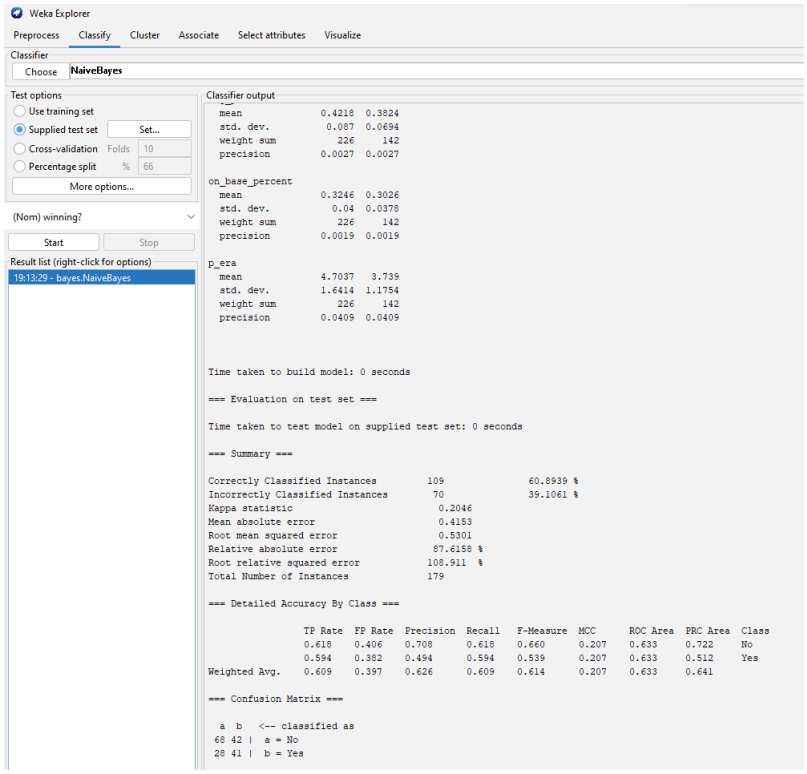


Figure 42: WEKA NaiveBayes Summary Results on GainRatioAttributeEval Attributes

4.6.4 OneR - GainRatioAttributeEval

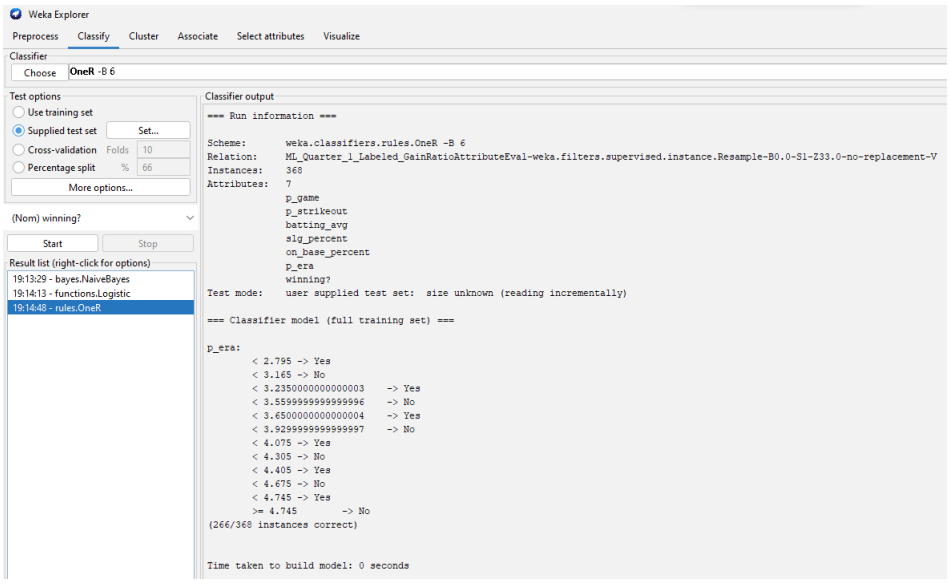


Figure 43: WEKA OneR Model on GainRatioAttributeEval Attributes

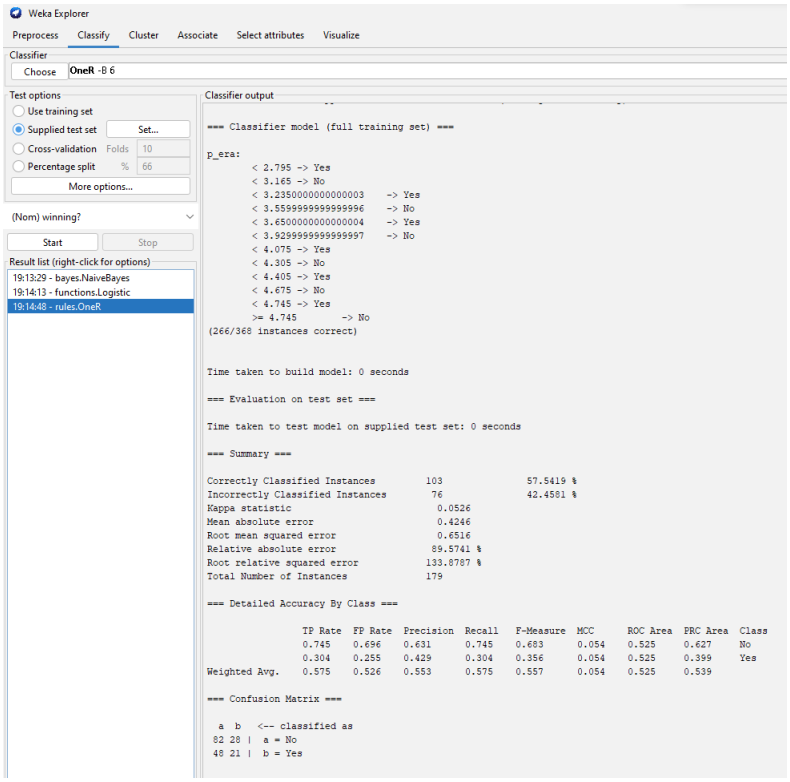


Figure 44: WEKA OneR Summary Results on GainRatioAttributeEval Attributes

4.6.5 Analysis

J48 performed the best with this attribute selection method, with **LogisticFunction** being a close second. Much like in previous cases, **J48** was a lot better at categorizing true positives while **LogisticFunction** was a lot better at categorizing true negatives.

4.7 InfoGainAttributeEval

4.7.1 J48 - InfoGainAttributeEval

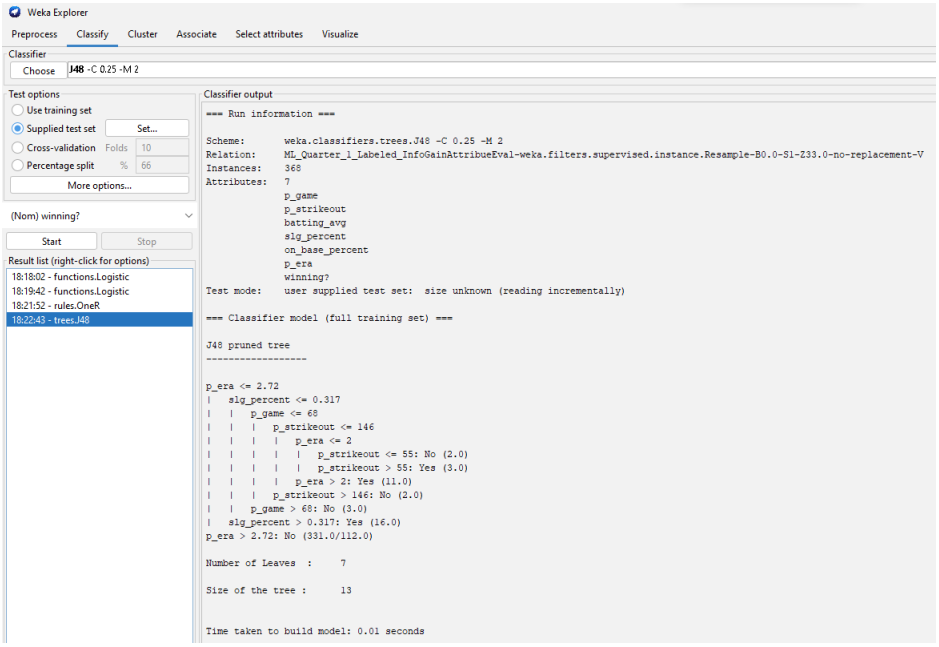


Figure 45: WEKA J48 Model on InfoGainAttributeEval Attributes

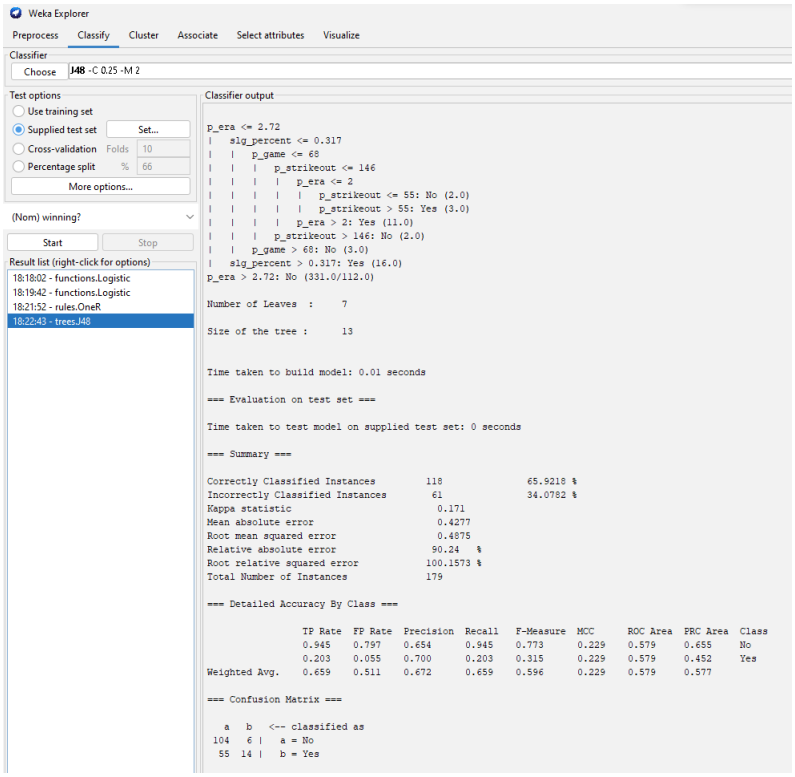


Figure 46: WEKA J48 Clasifcation Results on InfoGainAttributeEval Attributes

4.7.2 LogisticFunction - InfoGainAttributeEval

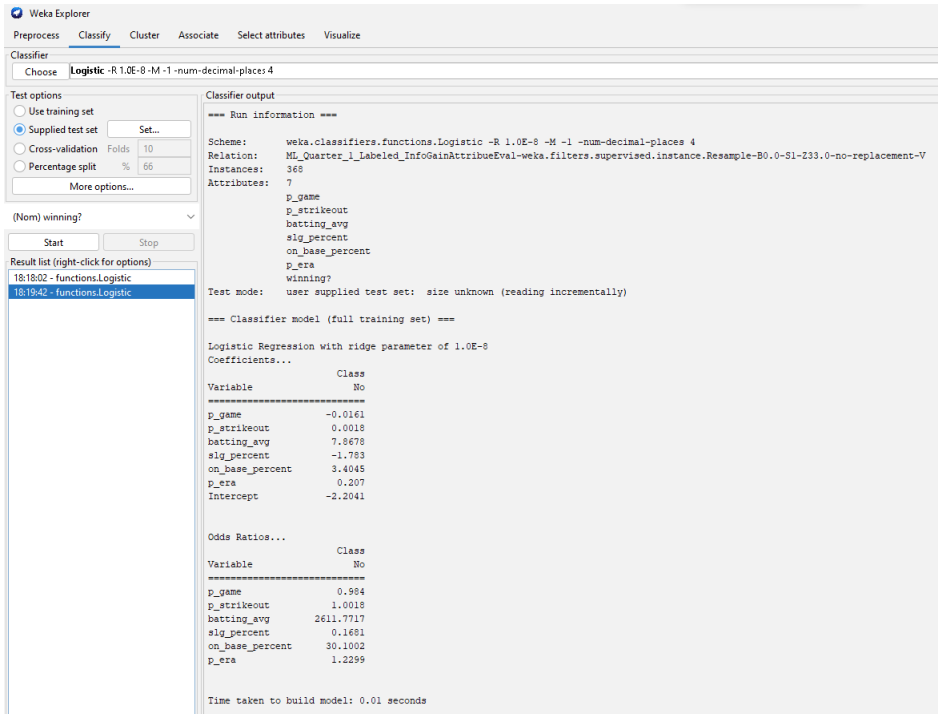


Figure 47: WEKA LogisticFunction Model on InfoGainAttributeEval Attributes

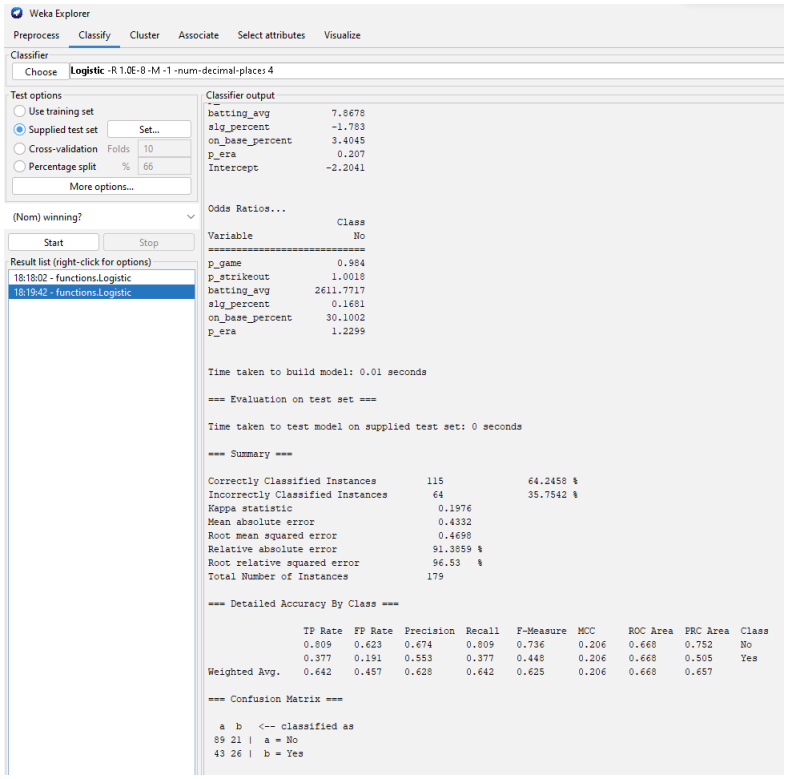
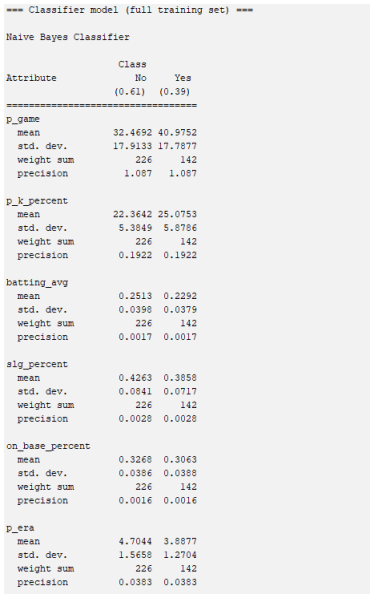


Figure 48: WEKA LogisticFunction Summary Results on InfoGainAttributeEval Attributes

4.7.3 Analysis

J48 performed the best with this attribute selection method, with **LogisticFunction** being a close second. **J48** was a lot better at categorizing true positives while **LogisticFunction** was a lot better at categorizing true negatives.

4.7.4 NaiveBayes - CorrelationAttributeEval



```
=== Classifier model (full training set) ===

Naive Bayes Classifier

                    Class
Attribute             No      Yes
                   (0.61)   (0.39)
===================================
p_game
  mean             32.4692 40.9752
  std. dev.        17.9133 17.7877
  weight sum           226     142
  precision          1.087   1.087

p_k_percent
  mean             22.3642 25.0753
  std. dev.         5.3849  5.8786
  weight sum           226     142
  precision         0.1922  0.1922

batting_avg
  mean              0.2513  0.2292
  std. dev.         0.0398  0.0379
  weight sum           226     142
  precision         0.0017  0.0017

slg_percent
  mean              0.4263  0.3858
  std. dev.         0.0841  0.0717
  weight sum           226     142
  precision         0.0028  0.0028

on_base_percent
  mean              0.3268  0.3063
  std. dev.         0.0386  0.0388
  weight sum           226     142
  precision         0.0016  0.0016

p_era
  mean              4.7044  3.8877
  std. dev.         1.5658  1.2704
  weight sum           226     142
  precision         0.0383  0.0383
```

Figure 49: WEKA NaiveBayes Model on CorrelationAttributeEval Attributes

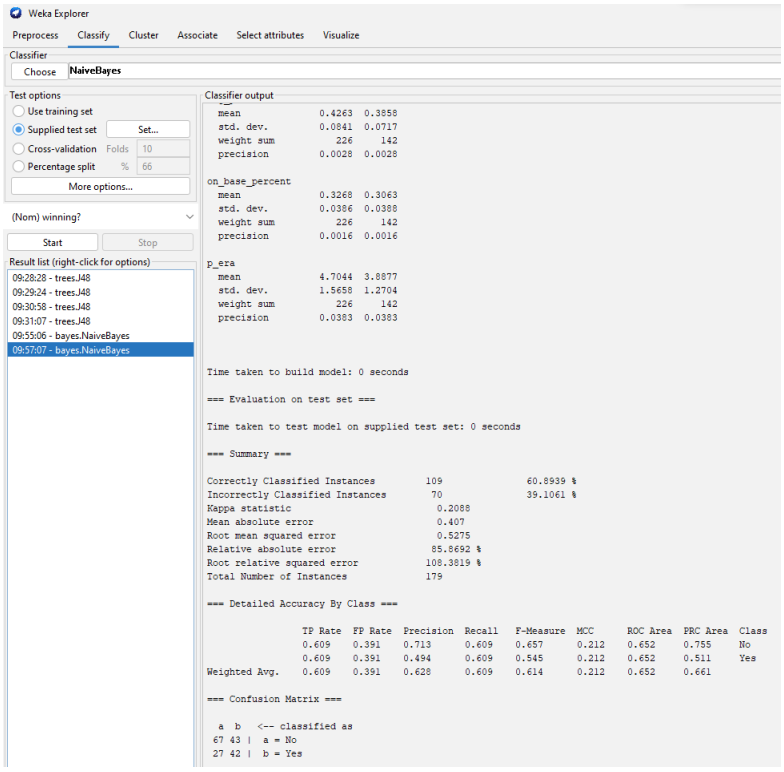


Figure 50: WEKA NaiveBayes Summary Results on CorrelationAttributeEval Attributes

4.7.5 OneR - CorrelationAttributeEval

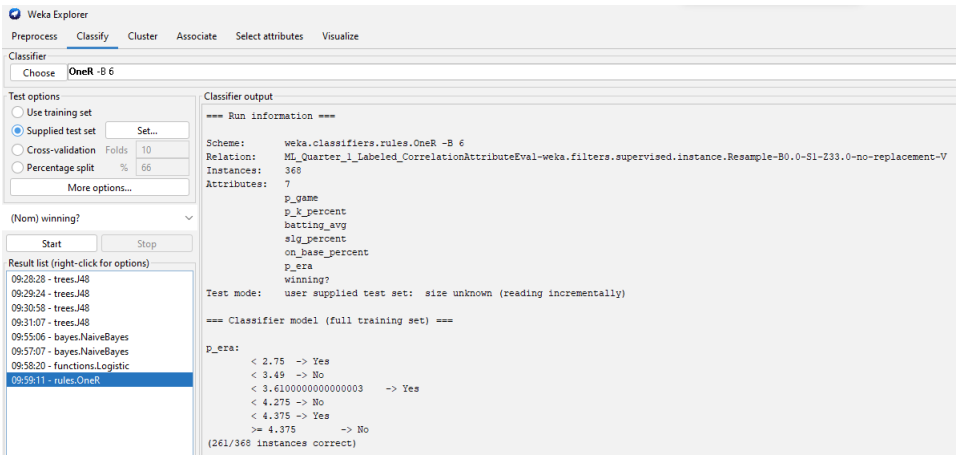


Figure 51: WEKA OneR Model on CorrelationAttributeEval Attributes

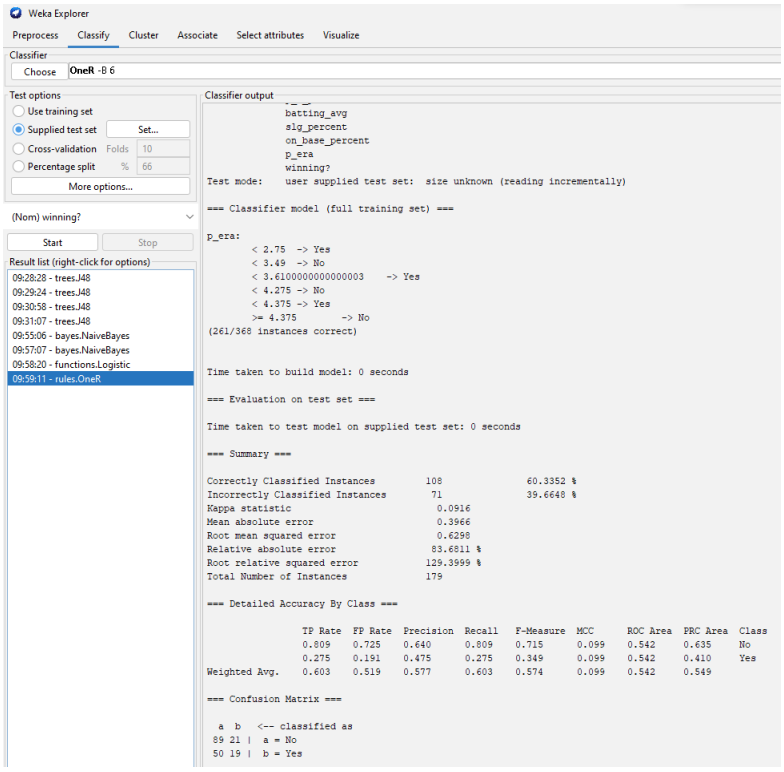


Figure 52: WEKA OneR Summary Results on CorrelationAttributeEval Attributes

4.7.6 Analysis

J48 performed the best with this attribute selection method, with **LogisticFunction** being a close second. **J48** was a lot better at categorizing true positives while **LogisticFunction** was a lot better at categorizing true negatives. This time, **LogisticFunction** performed a bit worse than normal compared to **J48**.

5 Conclusion

Looking at the data given to us by WEKA, we can see that the best combination of attribute selection and classification method is **WrapperSubsetEval** and **LogisticFunction**. This method classified with an accuracy of 67.0391%. However, with just one more instance misclassified, **WraperSubsetEval** and **J48** was a close second. Overall, the best performing models used the **WrapperSubsetEval** attribute selection method. This is probably due to the fact that it is one of the more complex attribute selection methods, and therefore is more accurate.

There are many reasons why **WrapperSubsetEval** and **LogisticFunction** are better in our case over **J48**. The former had the highest true positive rate (0.670), the lowest false positive rate (0.423), and the highest F-measure (0.655). These three facts combined make **WrapperSubsetEval** and **LogisticFunction** the undeniable best attribute selection method and classification method combination.

Overall, the accuracy of the classifier models was low, with the highest at just over 67%. This was surprising because we expected these statistics to be better predictors of performance, but the results make sense given the random nature of baseball and other external factors, like scouting reports and weather that could skew the statistics in some way.

6 Discussion

Throughout the course of this project, we learned a lot about data preprocessing and classification. We learned that well-chosen attributes are essential to having an accurate model, as evidenced by the fact that the attributes that we selected because we thought were the most predictive of pitcher performance were actually not the best attributes, and that greatly decreased the accuracy of any classifier algorithm, and even was the worst performing attribute selection method based on the results of the classification methods. This is a classic case of garbage in, garbage out.

We also learned that sometimes you cannot just compare the accuracy to determine which of two models is better. There needs to be some sort of justification. In our case, the difference between **LogisticFunction** and **J48** for the **WrapperSubsetEval** was almost negligible, but several factors pushed us to determine that **LogisticFunction** was better.

Finally, we learned that more is better. It is always better to test out more combinations of classifier models and attribute selection methods. While some are there to provide a baseline estimate, like **OneR** and **NaiveBayes**, they help provide us with insights if the more complex models go wrong and perform worse than the lightweight ones. Through testing of more combinations, we are more likely to find better performing models. While

J48 performed the best on most attribute selection methods, it did not perform the best overall, and it would be wrong for us to assume so. If we had not tried out **LogisticFunction** and **WrapperSubsetEval**, we may not have gotten the same level of performance.
The work was split evenly between both members during this project, with each person doing half of each step. We collaborated on the best way to handle missing values, and ended up deciding on just deleting rows we found were not needed. We each did three attribute reduction methods, collaborating on the **SelfSelected** method. We then ran classifier models on the two attribute reduction methods that we each did, and took screenshots of our work. Progress was updated through a git repository, so we always had access to each other's work. Finally, we collaborated on the writeup and the presentation.

7 Sources

We got our data from https://www.baseballsavant.mlb.com and used WEKA to preprocess and do classification work.
Datasets for this project can be found at https://drive.google.com/drive/folders/1sgNZJn71GUput_BZugSGtm8ybw6_6p0P?usp=sharing. This link includes both the original datasets as well as the modified datasets.
Research on models was done on both the official WEKA sourceforge site and through https://www.medium.com. The links to sources used, as well as the original dataset, are in the Bibliography on the next page.
WEKA (Waikato Environment for Knowledge Analysis), is a collection of machine learning and data mining algorithms and tasks. It was developed at the University of Waikato, New Zealand.

8 Bibliography

https://baseballsavant.mlb.com/leaderboard/custom?year=2021&type=pitcher&filter=&sort=2&sortDir=asc&min=100&selections=p_game,p_total_hits,p_home_run,p_strikeout,p_walk,p_k_percent,p_bb_percent,batting_avg,slg_percent,on_base_percent,on_base_plus_slg,p_earned_run,p_win,p_loss,p_era,p_rbi,p_called_strike,p_unearned_run,exit_velocity_avg,launch_angle_avg,sweet_spot_percent,barrel_batted_rate,hard_hit_percent,meatball_percent,pitch_hand,n_fastball_formatted,fastball_avg_speed,n_offspeed_formatted,offspeed_avg_speed,&chart=false&x=p_total_hits&y=p_total_hits&r=no&chartType=beeswarm
https://weka.sourceforge.io/doc.dev/weka/attributeSelection/WrapperSubsetEval.html
https://weka.sourceforge.io/doc.dev/weka/attributeSelection/InfoGainAttributeEval.html
https://weka.sourceforge.io/doc.dev/weka/attributeSelection/GainRatioAttributeEval.html
https://weka.sourceforge.io/doc.dev/weka/attributeSelection/CorrelationAttributeEval.html
https://weka.sourceforge.io/doc.dev/weka/filters/supervised/instance/Resample.html
https://medium.com/@nilimakhanna1/j48-classification-c4-5-algorithm-in-a-nutshell-24c50d20658e
https://medium.com/swlh/lets-talk-logistic-regression-4b2072ad7b4e
https://medium.com/x8-the-ai-community/a-simple-introduction-to-naive-bayes-23538a0395a
https://medium.com/squad-engineering/one-class-classification-for-images-with-deep-features-69182fb4c9