NHL Shot Outcome Prediction Model  
  
Team Members:  
Aidan Lee, Max Weinstein  
  
10/21/24

## **1. Project Goal**

Hockey, especially played in the National Hockey League, is known for being a very fast-paced sport, where every shot on goal is crucial. In each game of hockey, each team takes an average of 30 shots on goal, with the more shots taken, the higher likelihood of a goal being scored. This project aims to analyze a dataset of hockey shots taken in the National Hockey League (NHL), during the 2021-2022 season. We want to focus on the outcome of these shots: resulting in a goal, being blocked, or not being on target at all.   
  
We will utilize a dataset of 160,573 instances and 21 attributes, data on every shot from every NHL game during that season. Our attributes are based on the following categories:

1. Shot Data:   
    The distance from the goal, and the angle to the goal
2. Game Context:   
    Current score, current period, power play

Based on these attributes, plus a few additional ones provided by the dataset, we want to develop a multi-classification model that will accurately predict the shot outcomes. The model we will create would have several practical applications:

1. Training Tools:   
    Coaches can create simulations to demonstrate shot opportunities based on several factors, in order to teach the players what to look for
2. In-Game Analysis:  
    Based on information from the model, the coaches of a team can adjust the strategy to be more or less aggressive on offense, based on factors such as them taking shots at bad moments, or not taking viable shots.
3. Post-Game Analysis:   
    A combination of the two prior applications, coaches can use this model when reviewing the game film with the players post-game. By showing the shots, and the models opinion on them, while looking at the game from a birds-eye view, players can learn from their mistakes.

Throughout the project, we hope to find the most influential attributes towards the shot outcomes, and identify the most effective model for predicting shot results.

## **2. Dataset Description** We sourced the dataset from this project from the SCORE Network Sports Data Repository, and like we mentioned earlier, it accounts for every shot taken during the 2021-2022 NHL season. The data includes detailed shot information, such as the score of the game, the home and away teams, which period the shot was taken in, and whether the shot was on an empty net. Like most sports, the context of the game is crucial for the analysis of it, and with the factors of several players moving around the rink rapidly in hockey, the more contextual information we can get, the better. The dataset provided a nominal class variable of shot\_outcome, which made the classification approach easier. The variable had 4 outcomes:

* **SHOT:** A shot on goal, blocked by the goalie
* **MISSED\_SHOT:** A shot that missed the goal, without being blocked
* **BLOCKED\_SHOT:** A shot on goal, blocked by a non-goalie player
* **GOAL:** A shot on goal, that resulted in a goal

## **3. Pre-Processing**

3.1 - Removing Unnecessary Attributes

We started by removing some of the attributes that had no significance to us, and would just mess with the model’s accuracy when making a prediction, with a large amount of these being names. Here are the attributes we removed:

* game\_id: The ID for the game, likely to be used in the NHL API  
   *Ex: 2021020001*
* description: A description of the shot, with player names and shot outcome.  
   *Ex: “Steven Stamkos Wrist Shot saved by Tristan Jarry”*
* home\_name & away\_name: The names of the home and away teams in the game  
   *Ex: “Tampa Bay Lightning”*
* event\_team: Which team the shooter was on  
   *Ex: “Tampa Bay Lightning”*
* event\_player\_1\_name: The name of the shooter  
   *Ex: “Steven.Stamkos”*
* event\_player\_1\_type:   
   The type of player. Depends on shot outcome, which we can use instead.  
   *Ex: “Shooter” or “Scorer”*
* event\_goalie\_name & event\_player\_2\_name: The name of the goalie being shot on  
   *Ex: “Tristan.Jarry”*
* event\_player\_2\_type:   
   The type of the second player involved, depends on shot outcome  
   *Ex: “Goalie”, “Blocker”*

3.2 - Removing Null Data

Before transforming the data, we wanted to remove null data, because it would affect the accuracy of the model, because of its reliance on having the context around the shot.

Starting with the variables that seemed to have the largest effect on the context of the shot, shot\_distance & shot\_angle had 38,224 instances where they were missing, so we dropped all of them, leaving the dataset with 120,556 instances. This correlated with the BLOCKED\_SHOT category in the class variable, which completely disappeared after dropping these rows, leaving us with 3 outcomes: SHOT, GOAL, MISSED\_SHOT.  
  
Next, strength\_code had 303 instances where it was missing, and with such a low number of instances for what seemed like a major variable, we also dropped those instances from the dataset, leaving us with 120,253 instances.  
  
Then we came to a trickier part, where the only other variable with missing values was empty\_net, having 111,742 instances where it was missing, which represented 93% of the total dataset. With it being a decently important attribute, we decided to try and keep it by filling in the missing values. We settled on Random Sampling Based on Proportions, where we kept the distribution of the filled in values, and randomly assigned missing values to True/False, while keeping the existing proportion, going from [517, 7,994] true/false to [7,272, 112,981] true/false, but we also saved a dataset with the missing values dropped if we needed to use it for some reason later. A random seed of ‘20241009’ was used when randomly assigning the values, for reproducibility.

3.3 - Transforming Data

We then transformed some variables, so that the model would have an easier time interpreting them. Transformations were performed using Python, by loading our dataset with Pandas, then manipulating the DataFrame object.

* empty\_net: “TRUE” / “FALSE” -> True/False  
   Converted to boolean
* game\_seconds\_remaining (Numeric):  
   We did several things with this variable. Starting off, there were negative values, which shouldn’t exist for a variable calculating how much time is left in the game, and with less time left, teams become more aggressive about scoring, so incorrect values could mess with the model. There are some instances with 0 as the value, but we will ignore those for now, assuming that they were probably close enough to the end of the game that it wouldn’t matter much. As a result, we chose to drop the rows where the variable was negative, removing 2,163 instances.  
    
   Secondly, the range of the variable is pretty large, being 3597 seconds. We decided to normalize the data using min-max normalization, which scaled it to values between 0 and 1.
* period\_seconds\_remaining (Numeric):   
   Similar to game\_seconds\_remaining, but didn’t have any negative values. Normalized with min-max normalization.
* x\_fixed & y\_fixed (Numeric):   
   With ranges of [-99, 99] and [-42, 42], we normalized this with Z-score normalization, so that it would keep the aspect of it being centered around 0, after normalization.
* shot\_distance (Numeric):  
   With a large range, but mostly skewed to being closer to the goal, we normalized it with Z-score normalization.

3.4 - Splitting to Training/Validation/Testing

Then using Pandas in Python, we did a 70/15/15 split on the cleaned dataset. 70% of the total dataset will go into a training dataset, 15% will be a validation dataset, and the final 15% will be the testing dataset. The same random seed of “20241002” was used for the splitting. This resulted in: 84,177 instances in the training dataset, 18,038 instances in the validation dataset, and 18,038 in the training dataset. Stratify was used to preserve the distribution of the class variable in each dataset, so they would all end up with basically the same distribution: ~65.9681% SHOT, ~26.8541% MISSED\_SHOT, ~7.1793% GOAL.

## **4. Attribute Selection and Model Classifiers**

After the pre-processing stage of our project, our dataset was left with 11 attributes:

1. period
2. period\_seconds\_remaining
3. game\_seconds\_remaining
4. home\_score
5. away\_score
6. empty\_net
7. strength\_code
8. x\_fixed
9. y\_fixed
10. shot\_distance
11. shot\_angle

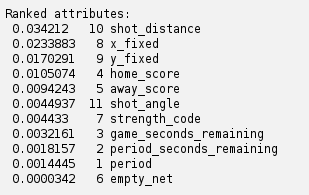
Class Attribute: shot\_outcome

4.1 - Attribute Selection Algorithms

1. **CfsSubsetEval:**We used WEKA for this selection algorithm. We used BestFirst as the search method. After running it, it selected the following 6 variables as being the most related with the class variable:

* home\_score
* away\_score
* strength\_code
* x\_fixed
* y\_fixed
* shot\_distance

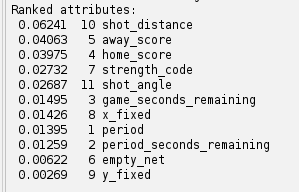
1. **InfoGainAttributeEval:**Using WEKA, we ran InfoGainAttributeEval with the Ranker search method. Here is what was outputted:

  
Based on this, setting a threshold at 0.003, we will only keep 8 attributes, ranked highest to lowest correlation with the class variable:

* shot\_distance
* x\_fixed
* y\_fixed
* home\_score
* away\_score
* shot\_angle
* strength\_code
* game\_seconds\_remaining

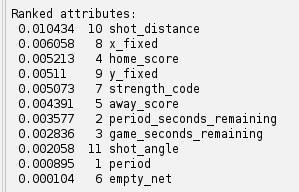
1. **CorrelationAttributeEval:**

Using WEKA, we ran CorrelationAttributeEval with the Ranker search method. Here was the output:

  
Based on this, with a selected threshold of 0.015, we will keep the following attributes:

* shot\_distance
* away\_score
* home\_score
* strength\_code
* shot\_angle

1. **GainRatioAttributeEval**

Using WEKA, we ran GainRatioAttributeEval with the Ranker search method. Here is the output of that:  
  
  
With a selected threshold of 0.003, we will keep the following attributes:

* shot\_distance
* x\_fixed
* home\_score
* y\_fixed
* strength\_code
* away\_score
* period\_seconds\_reamining

1. **Mutual Information:**We chose to use the mutual information algorithm, because it works well with both categorical and continuous data, which is good for our dataset, containing continuous data like shot\_distance, and categorical data like strength\_code. In addition to that, it is also able to capture linear and non-linear relationships, meaning that it can interpret more complex relationships to the class variable if needed. We ran using Python3 with the scikit-learn library, which contains a mutual\_info\_classif function in its feature\_selection library.

Here is the essential part of our script, where df is our loaded dataset:



X = df.drop('shot\_outcome', axis=1)

y = df['shot\_outcome']

X = pd.get\_dummies(X, drop\_first=True)

scaler = StandardScaler()

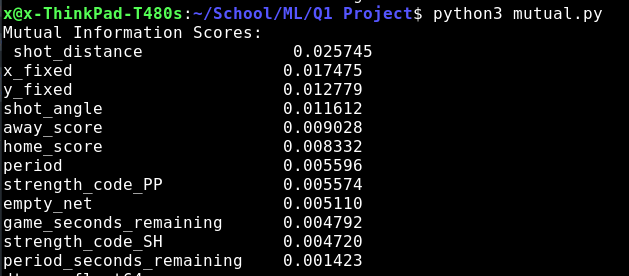
X\_scaled = scaler.fit\_transform(X)

mi\_scores = mutual\_info\_classif(X\_scaled, y, discrete\_features='auto')

mi\_scores = pd.Series(mi\_scores, index=X.columns)

mi\_scores = mi\_scores.sort\_values(ascending=False)



Here is what the script outputted:  
 

Setting a threshold value of 0.01, based on this algorithm, we should keep the following attributes:

* shot\_distance
* x\_fixed
* y\_fixed
* shot\_angle

4.2 - Classifier Models

1. **trees.J48**

J48 is an open-source implementation of the popular C4.5 algorithm, which builds a decision tree that is used for classification. This model works by recursively partitioning the dataset into different subsets that are based on the attribute that provides the largest information gain at each tree split. Once the tree is built, each node represents a class label, and the decisions are made by tracing paths from the root of the tree to the nodes.

1. **bayes.NaiveBayes**

NaiveBayes is a classification algorithm that is based on Bayes’ Theorem with the assumption that all features are independent. Despite this assumption, the model often performs well, especially in spaces with high-dimensions. The model calculates the conditional probability of each class given the feature values, then classifies a new instance by selecting the class with the highest posterior probability. It is suitable for situations where speed is important, as it is computationally quick.

1. **rules.OneR**

OneR is a simple but powerful rule-based classifier. It generates rules based on each separate attribute to predict the class. The algorithm goes one by one through each feature and selects the one that gives the fewest error rate, hence the name “OneR (one rule)”. For each value of the attribute, it assigns the value that has the lowest error rate. Although OneR may seem too simple, it often performs well, serving as a strong baseline model.

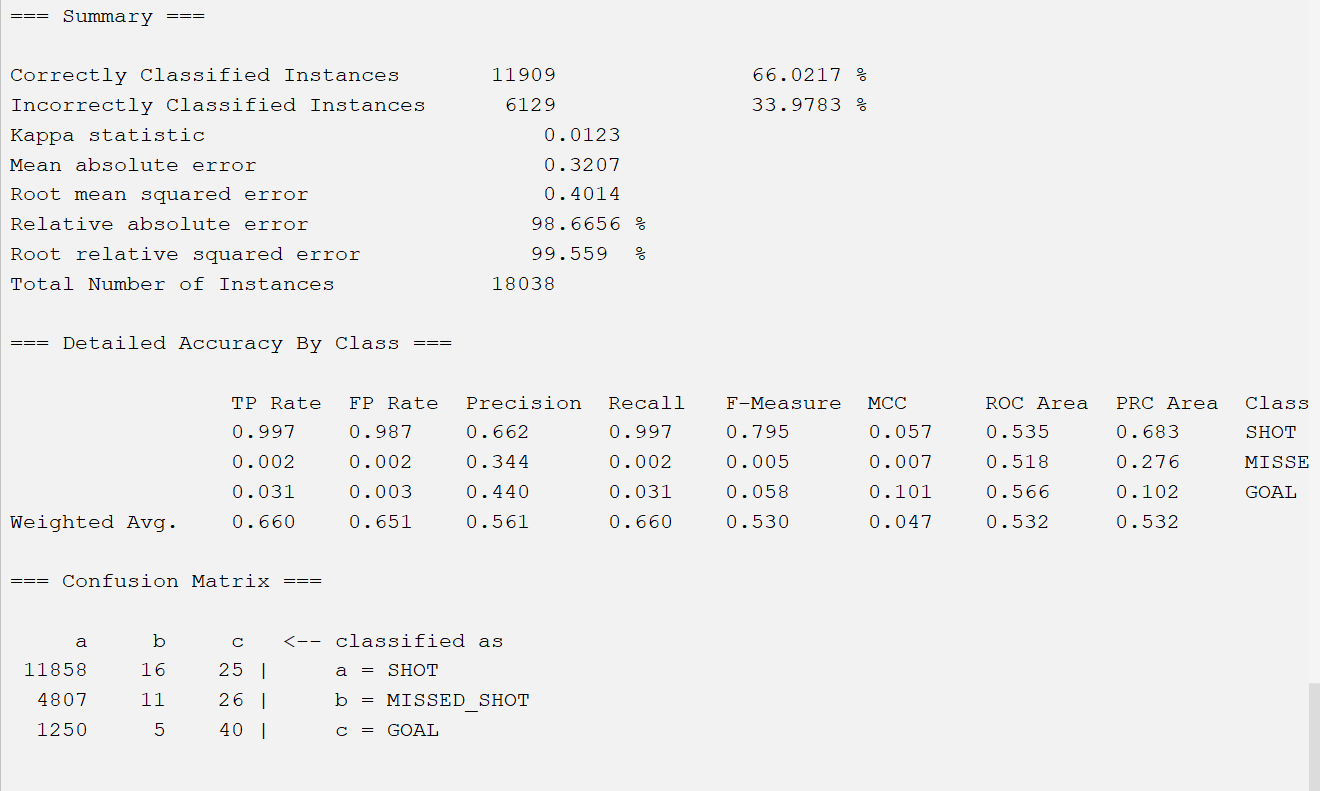
1. **trees.RandomTree**

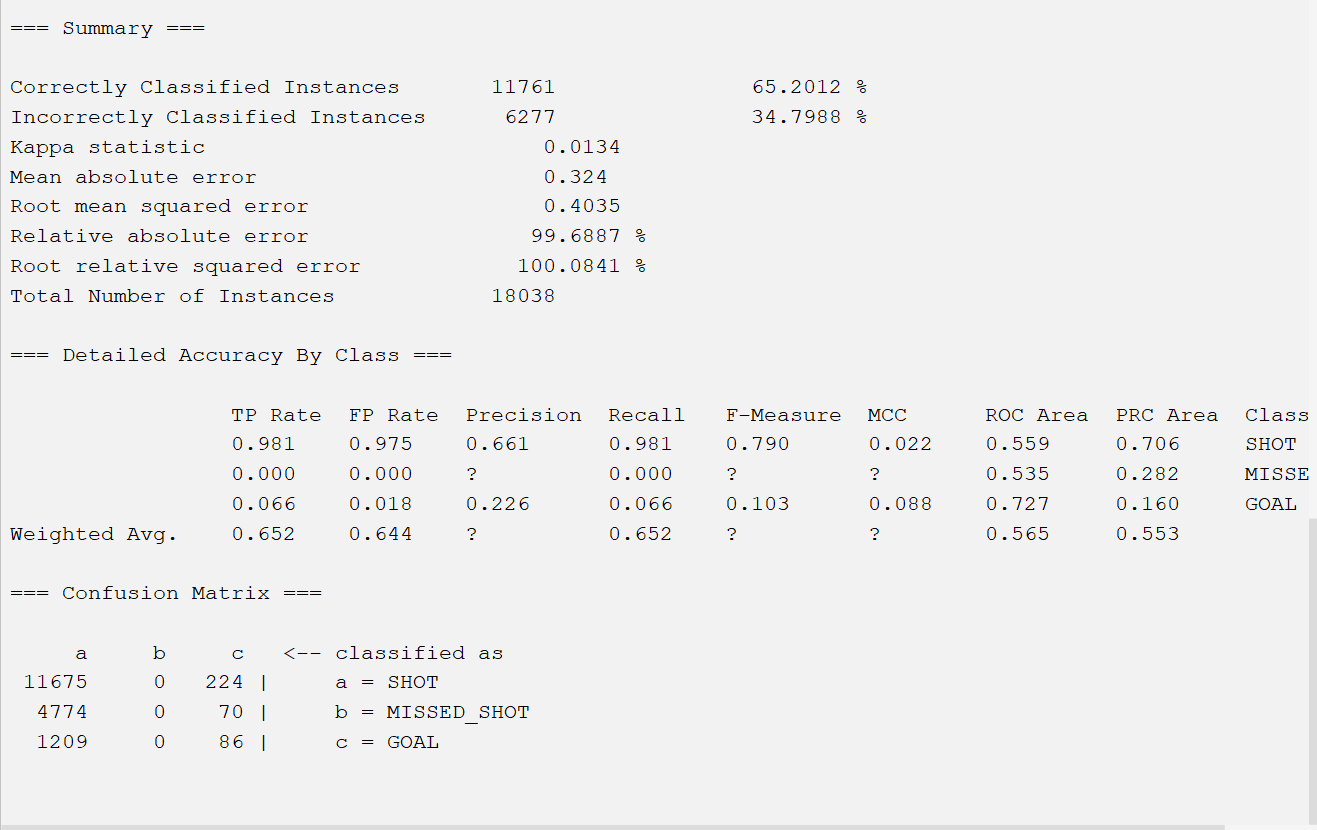
This classifier is a randomized decision tree algorithm. Unlike traditional decision trees such as J48, RandomTree selects a random subset of attributes, ensuring that there is greater diversity in the tree. RandomTree is a base classifier that is used in methods like RandomForests which combine multiple random trees to improve the performance of the classifier.

## **5. Results and Analysis**

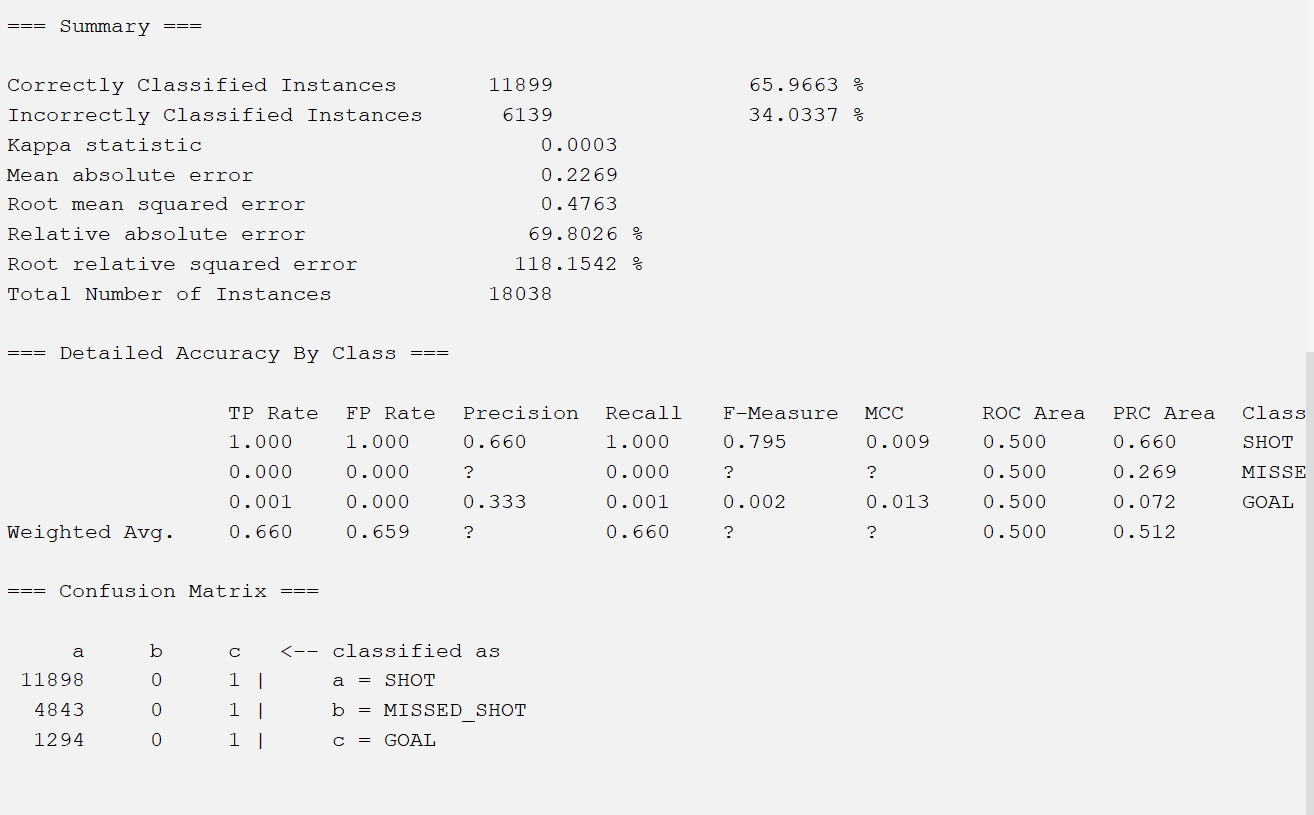
5.1 - Results

**CfsSubsetEval with J48**

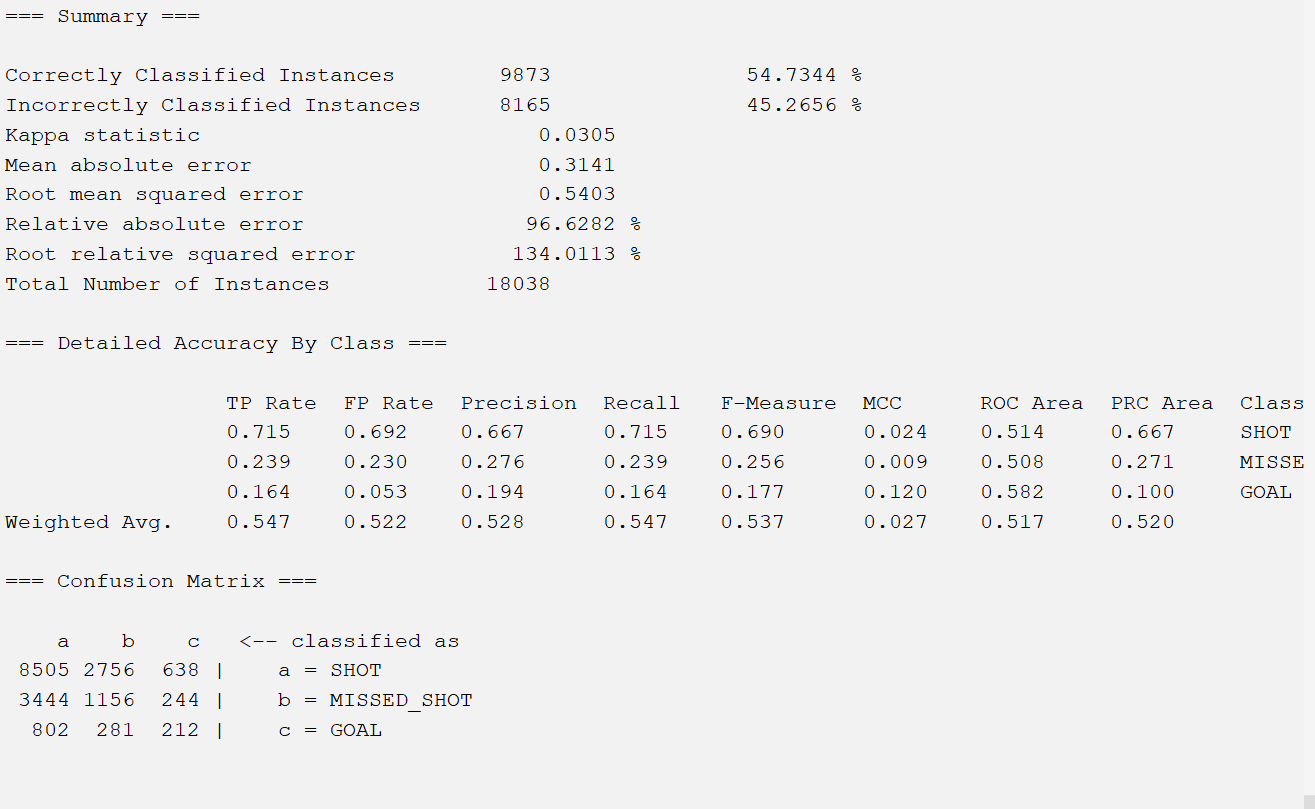


**CfsSubsetEval with NaiveBayes**

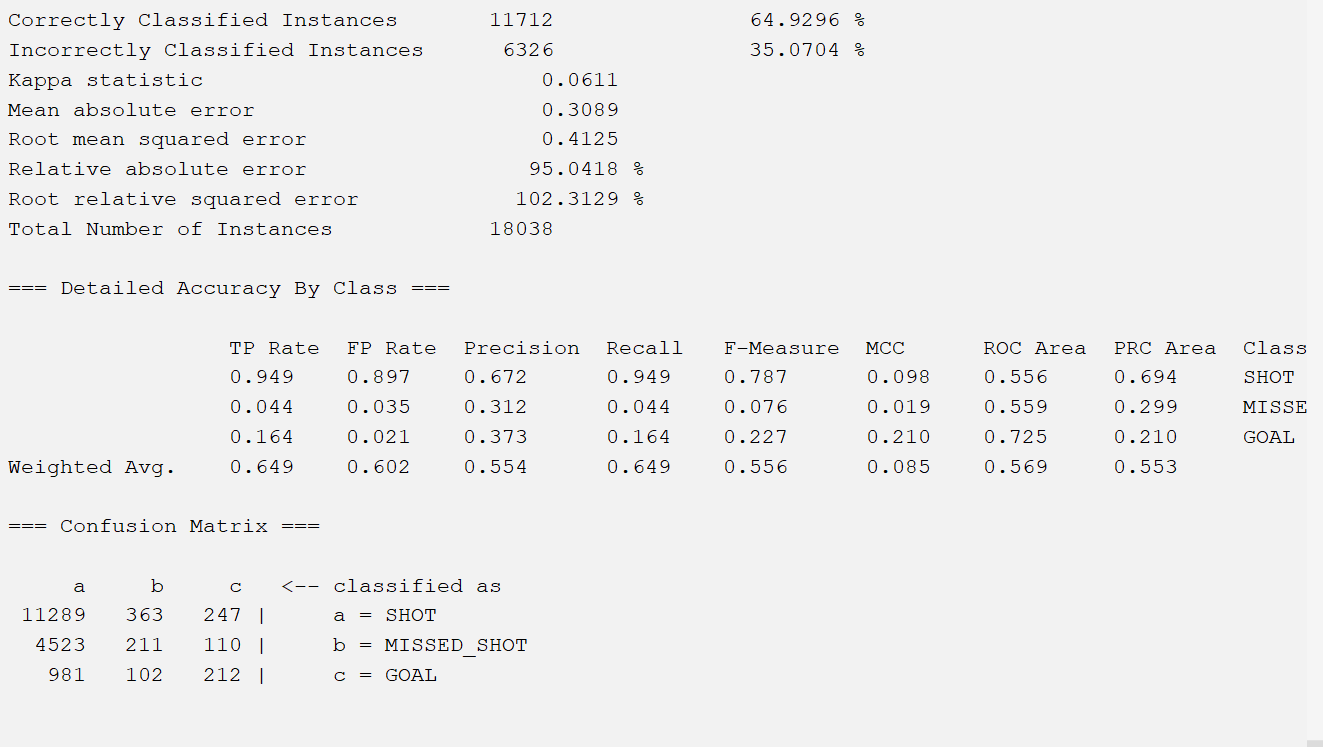
**CfsSubsetEval with OneR**



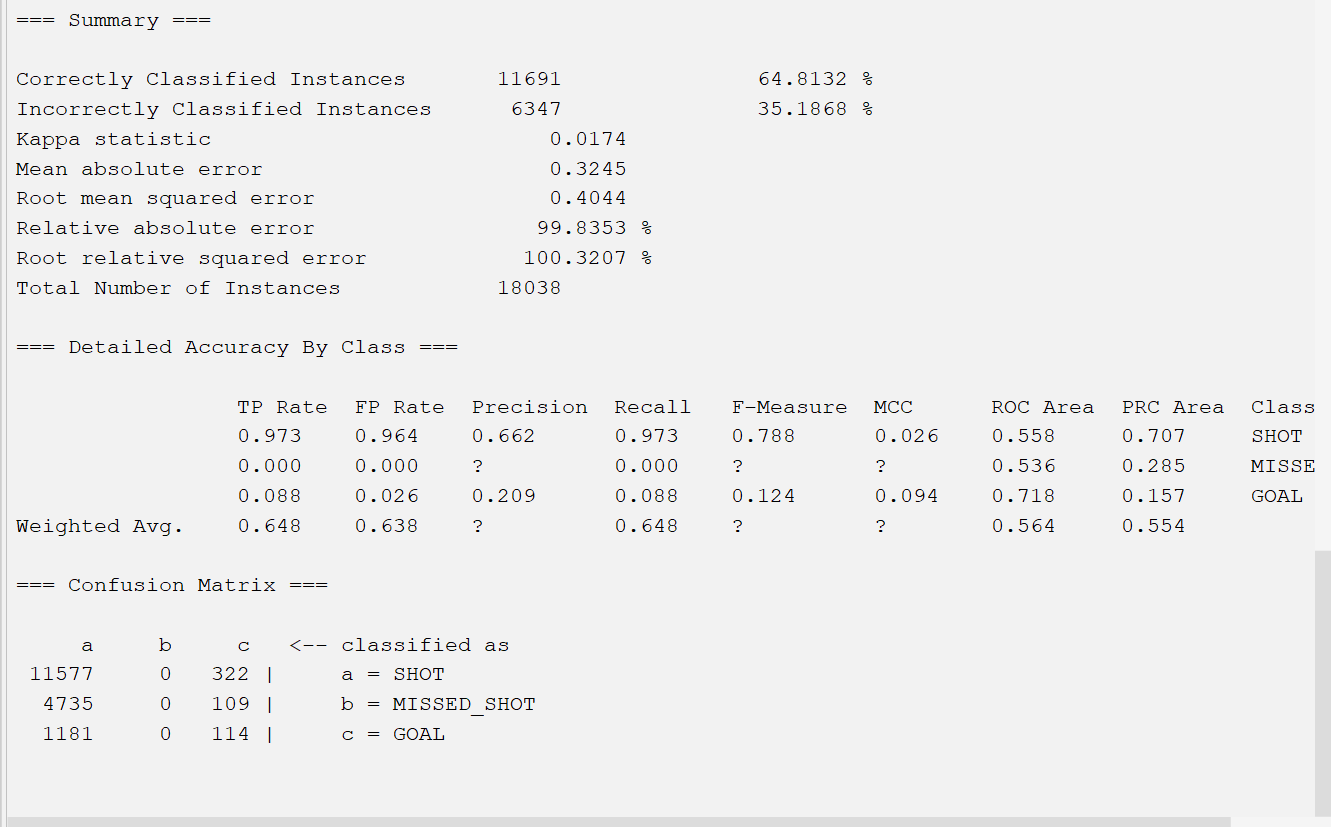
**CfsSubsetEval with RandomTree**



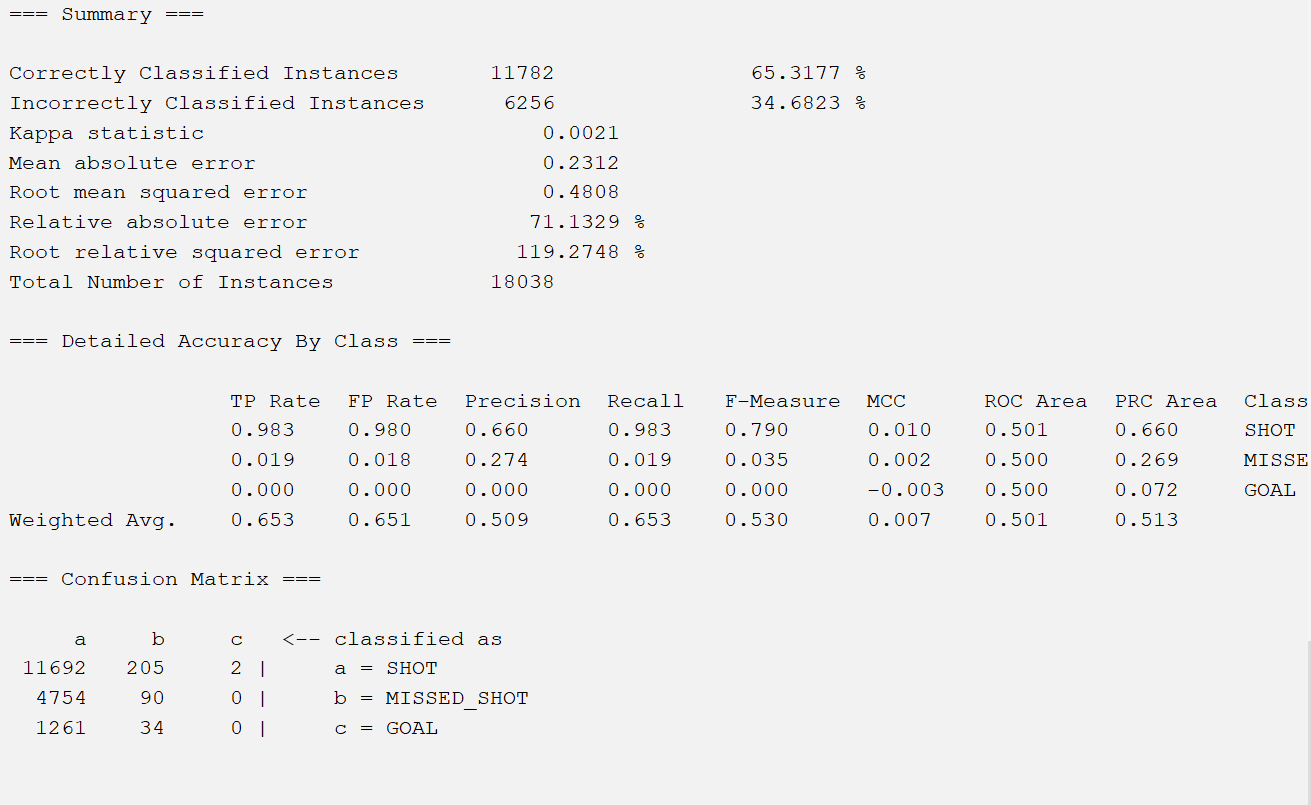
**InfoGainAttributeEval with J48**



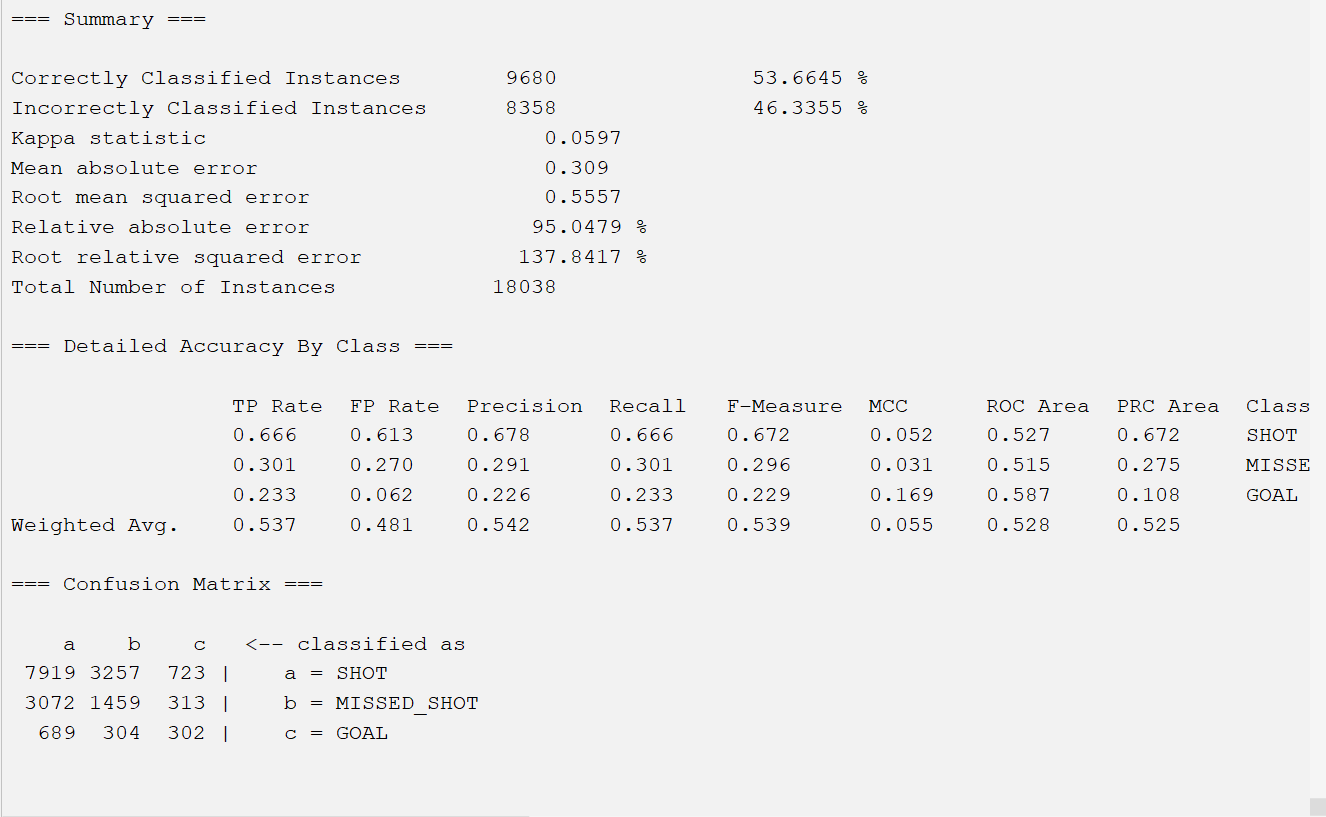
**InfoGainAttributeEval with NaiveBayes**



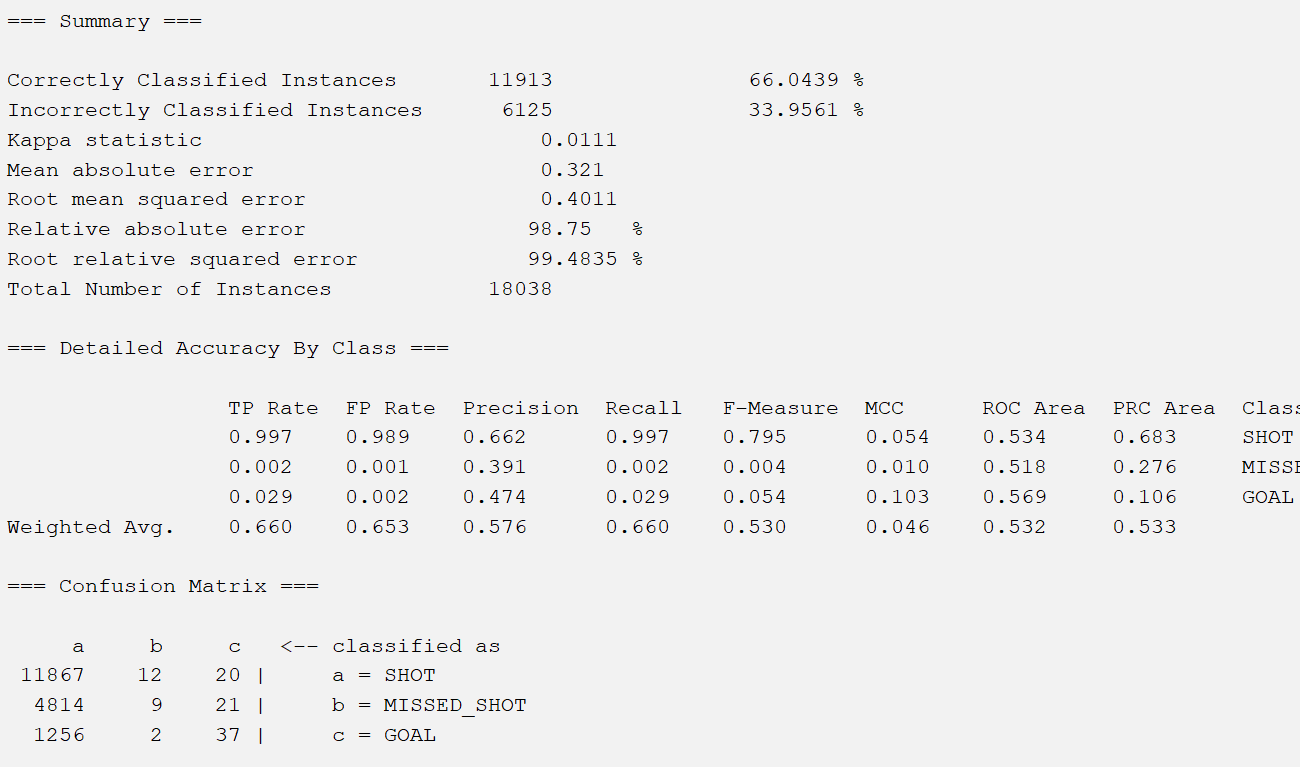
**InfoGainAttributeEval with OneR**



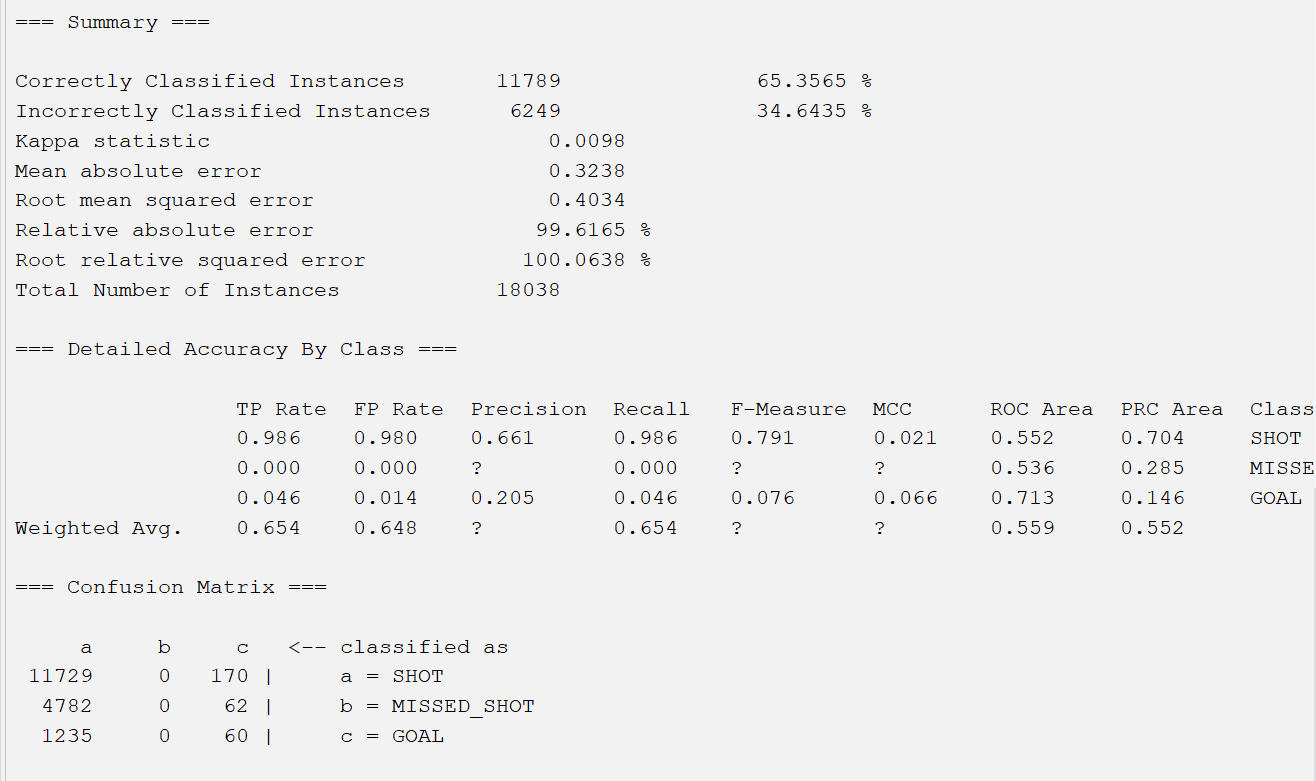
**InfoGainAttributeEval with RandomTree**



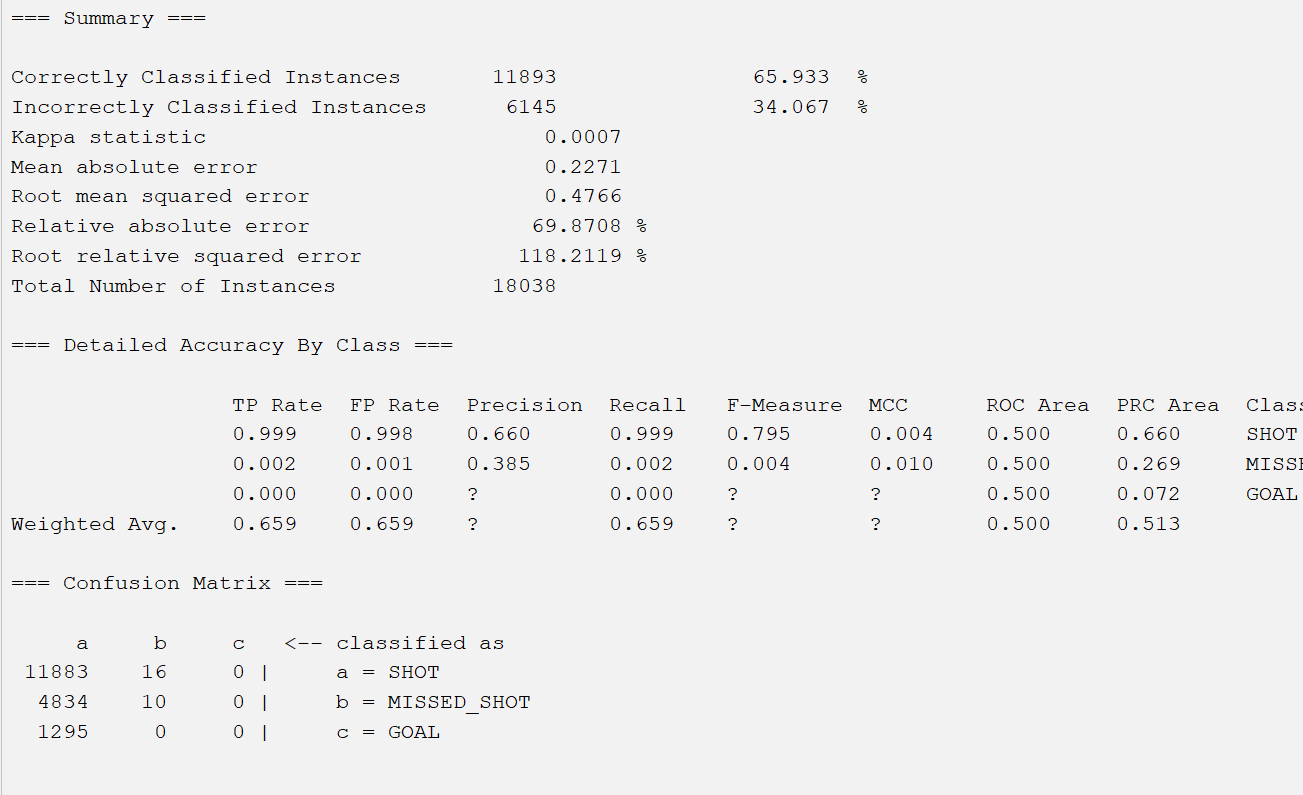
**CorrelationAttributeEval with J48**



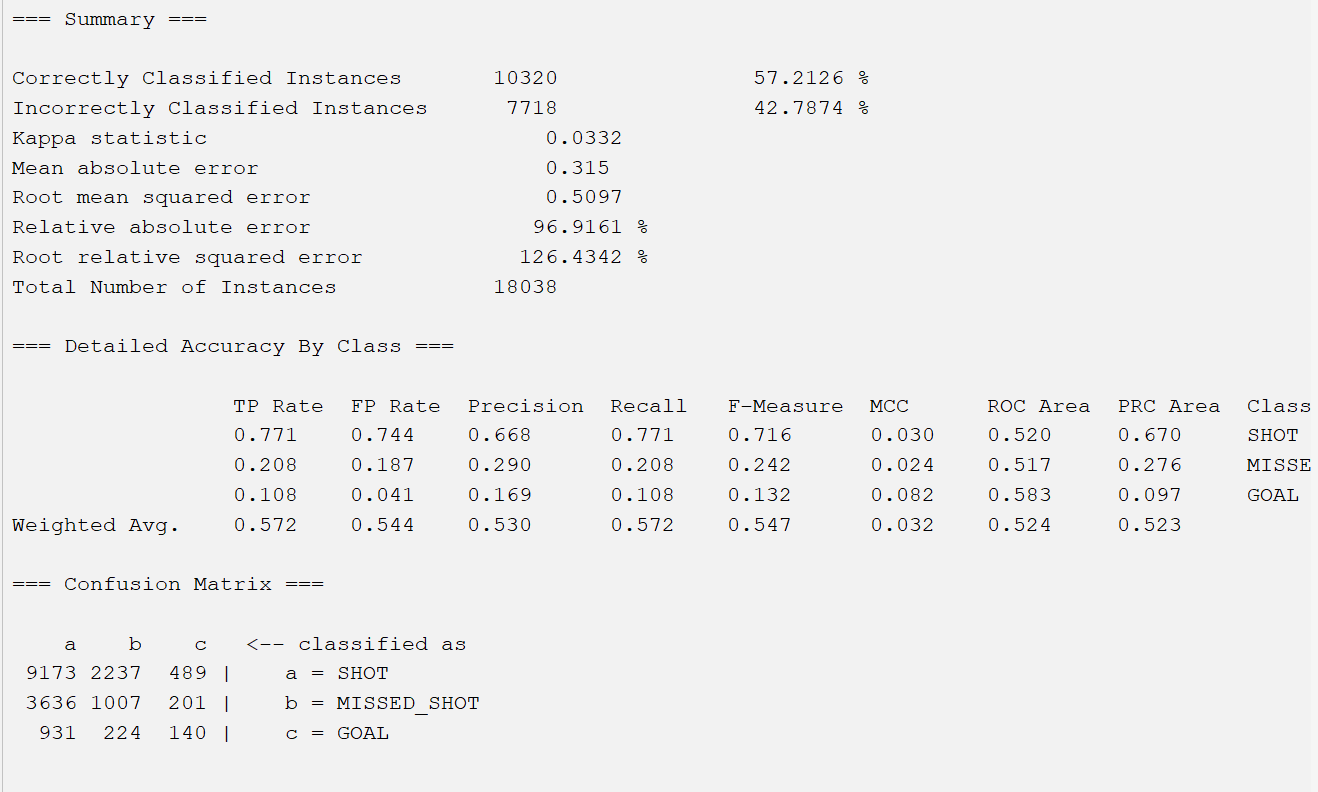
**CorrelationAttributeEval with NaiveBayes**



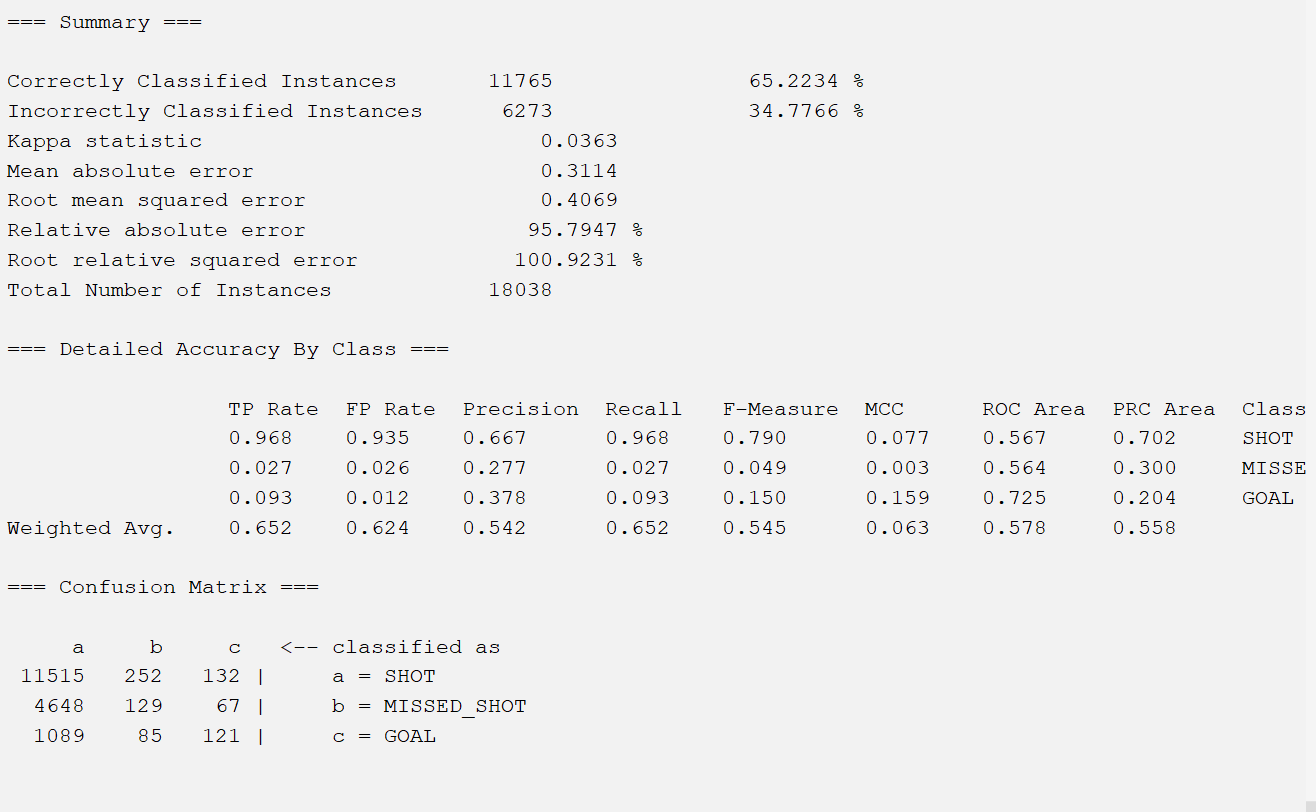
**CorrelationAttributeEval with OneR**



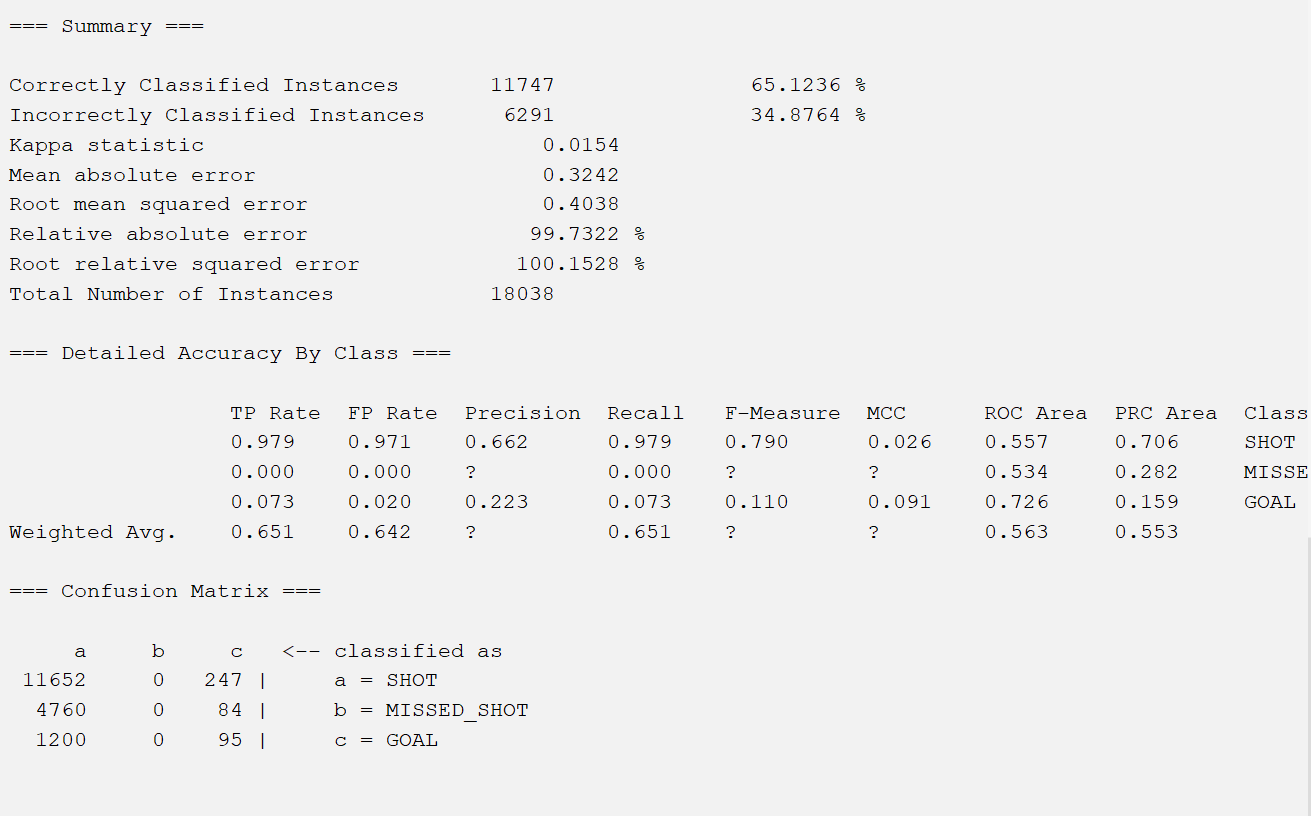
**CorrelationAttributeEval with RandomTree**



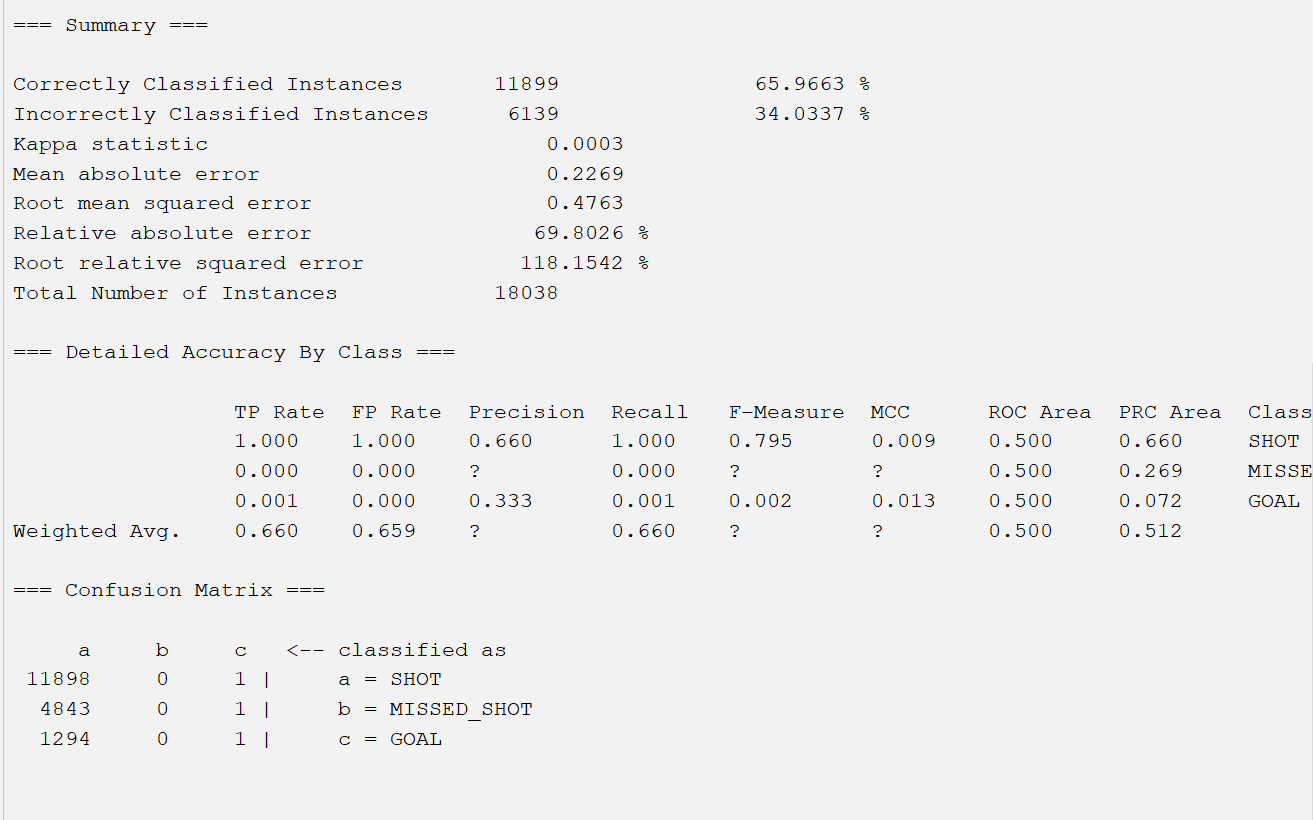
**GainRatioAttributeEval with J48**



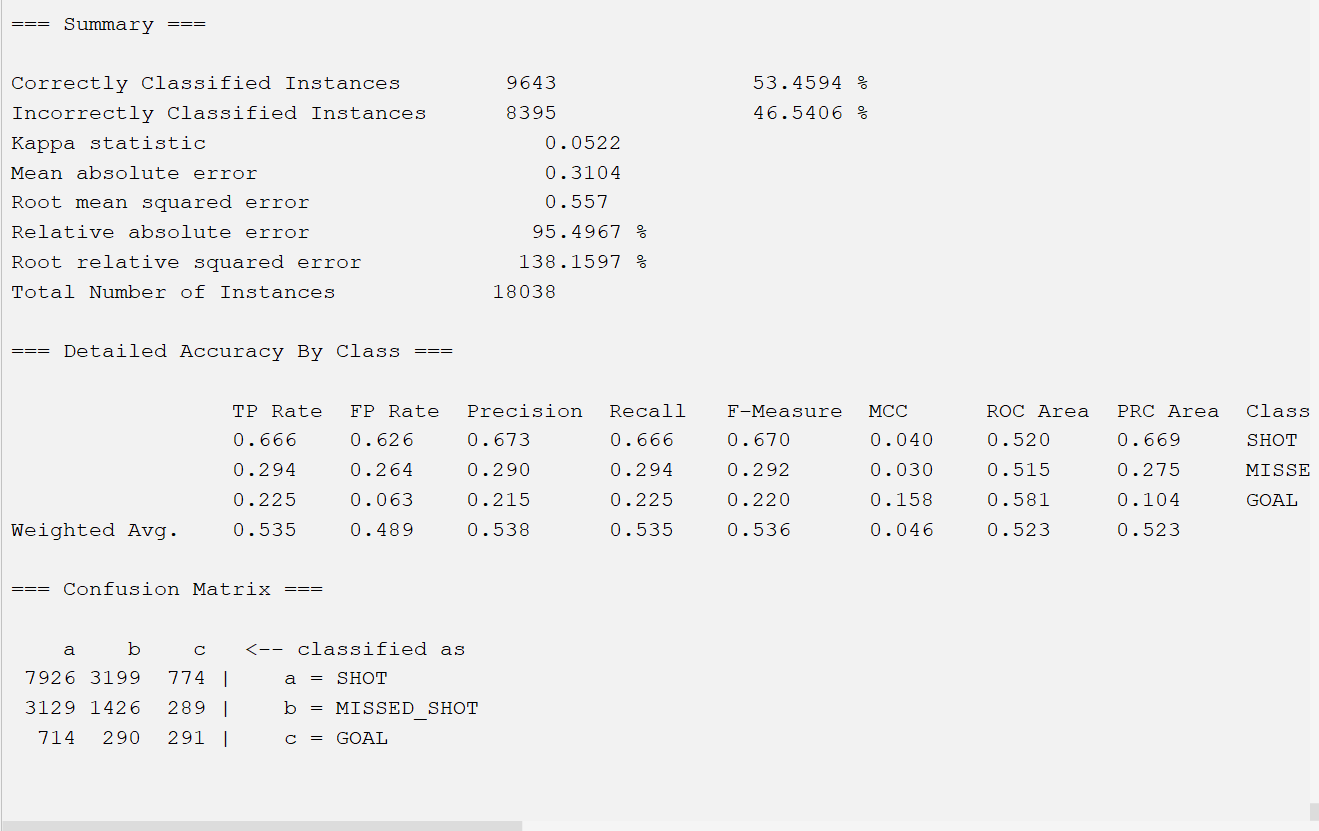
**GainRatioAttributeEval with NaiveBayes**



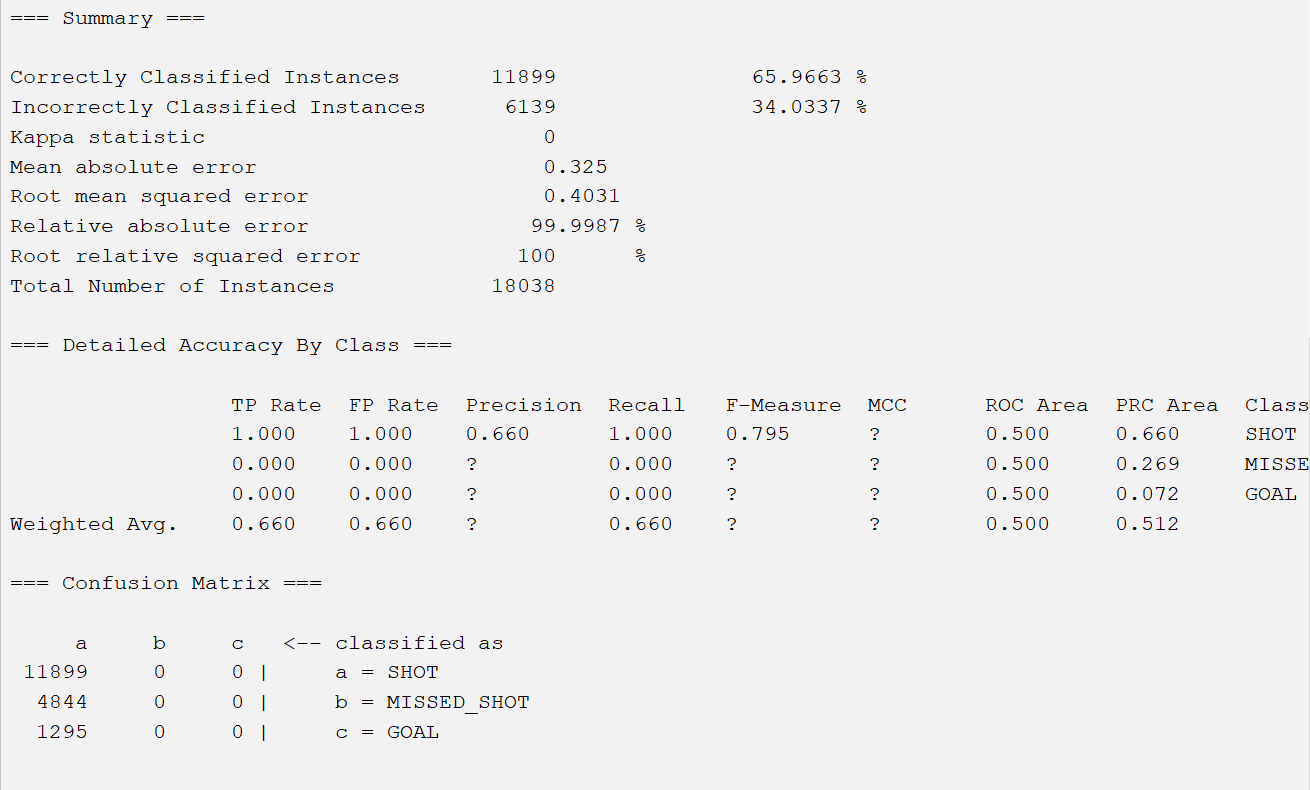
**GainRatioAttributeEval with OneR**



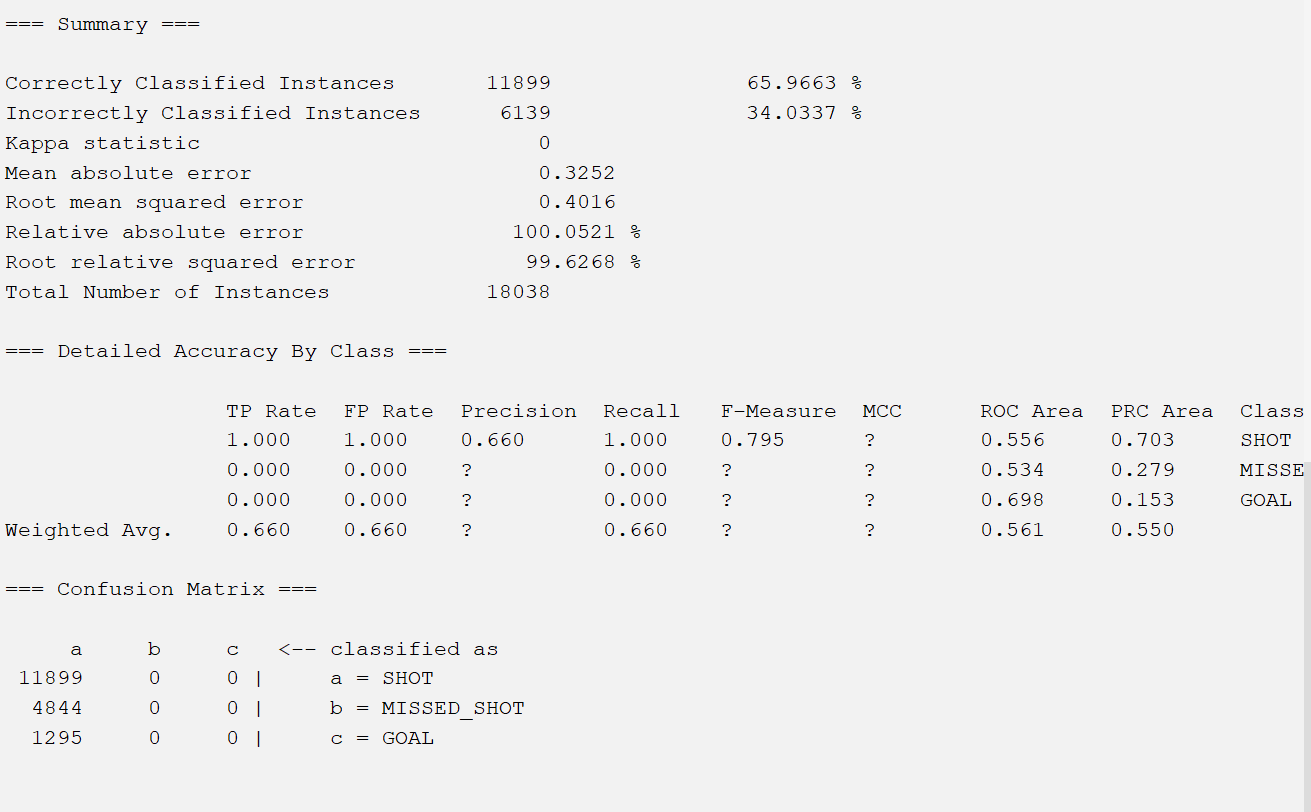
**GainRatioAttributeEval with RandomTree**



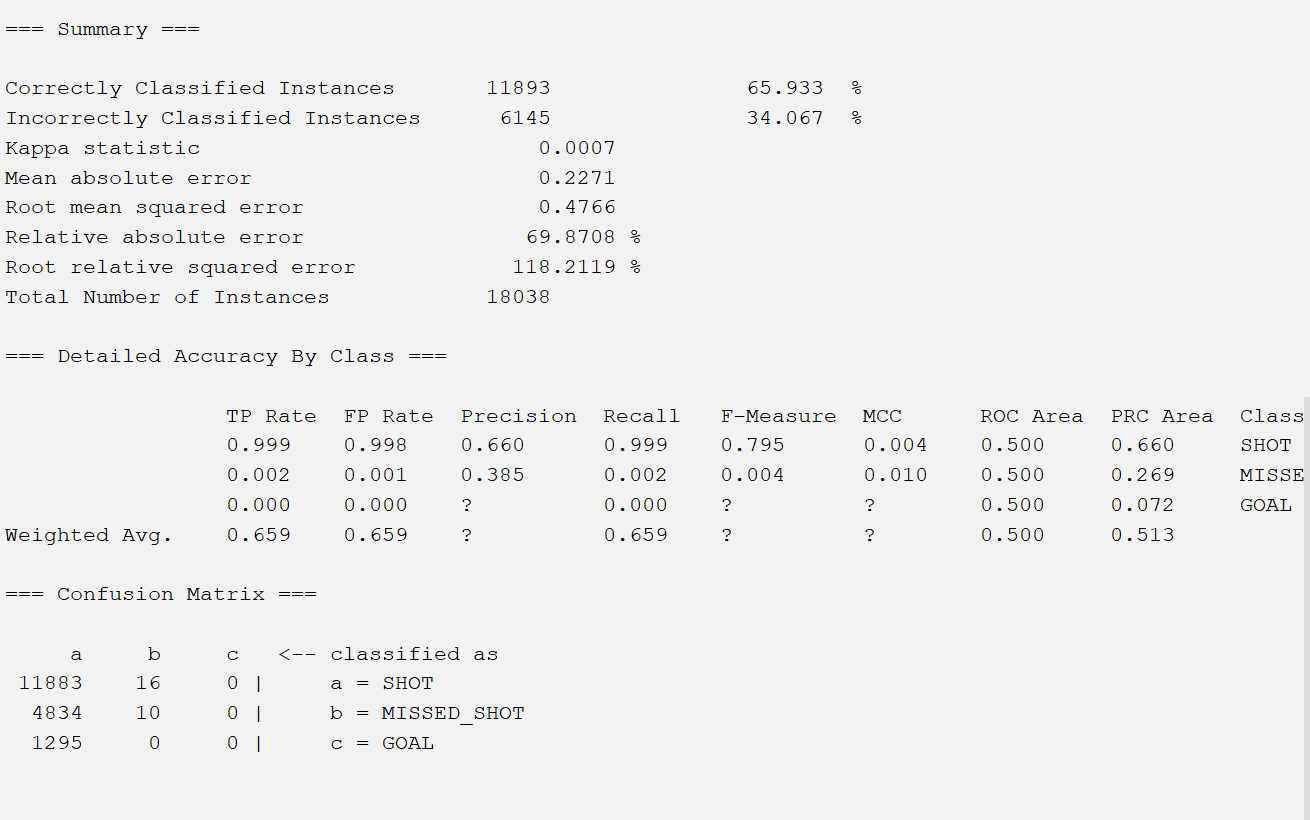
**Mutual Information with J48**



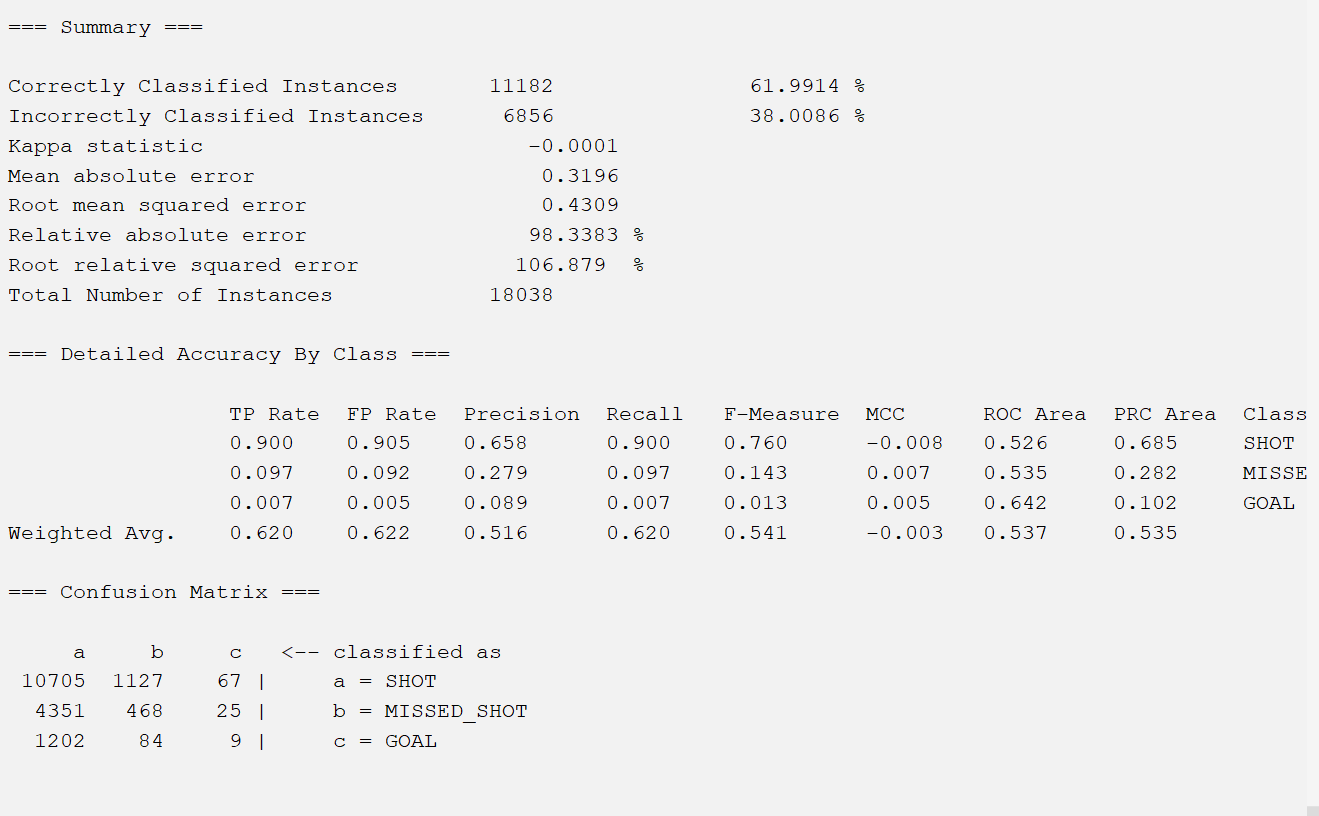
**Mutual Information with NaiveBayes**



**Mutual Information with OneR**



**Mutual Information with Random Tree**



5.2 - Analysis

We ran four different classifier models, which were J48, NaiveBayes, OneR, and Random Tree, on five different datasets that were created by using the attribute selection algorithms CfsSubsetEval, InfoGainAttributeEval, CorrelationAttributeEval, GainRatioAttributeEval, and Mutual Information. After running these classifier models and attribute selection algorithms, we were able to achieve the highest correctly classified instances with **CorrelationAttributeEval** **with J48**. This attribute selection and classifier model combination gave us 11913 correctly classified instances, which is an accuracy of 66.0439%.

We can also look at the errors that are provided by the model results. Looking at the four error results provided—mean absolute error, root mean squared error, relative absolute error, and root relative squared error—there is not much of a difference across all of the outputs. So, the errors can be almost ignored, and the model with the highest accuracy can be chosen.

Although the accuracy for the classifier models seem to be low for our datasets, this can be expected because of several key aspects of our initial dataset. First, instead of two classes, we have three, blocked shot, missed shot, and goal. Additionally, sports have inherent unpredictability. Players can miss shots that they easily should have made and some shots can have a lucky bounce or the goalie could make an error, resulting in a goal. These aspects of the dataset make it harder for the model to be accurate.

## **6. Conclusion and How to Reproduce Our Model**

Our chosen model is CorrelationAttributeEval with J48. This pairing had the best results out of all of our 20 different pairings. We created a model that provided us with decent results, although we feel like they could have been better. There are ways that we could improve our model and attribute selection. Future work could include combined attributes that may provide us with better results.

**How to reproduce our model**

1. Open WEKA and load the explorer.
2. In the explorer, open the file train\_set.csv.
3. This file was already processed, so you don’t need to change anything about the data
4. Go to the “Select attributes” tab
5. In the class selection dropdown (defaults as “No class”), choose our class variable: “(Nom) shot\_outcome”
6. Choose “CorrelationAttributeEval” from the selection in the Attribute Evaluator dropdown, and it will automatically pick “Ranker” as the search method
7. Click Start
8. Write down which attributes pass the threshold of 0.015, then go over to the Preprocess tab, and remove every attribute that doesn’t, by selecting them and clicking remove. **Make sure to keep the class variable.** For these sets, we end up with the following attributes: home\_score, away\_score, strength\_code, shot\_distance, shot\_angle, shot\_outcome
9. Save this dataset as an arff file.
10. Open the file test\_set.csv
11. Remove attributes to attain the same features as the training dataset, and save the dataset as an arff file
12. Open the Explorer and load the training dataset arff file
13. Go to the “Classify” tab, under options click “Supplied test set”, and load the test dataset arff file. Make sure to select the correct class variable (“shot\_outcome”)
14. Make sure to have the correct class on the class dropdown (defaults to “No class”)
15. Under the Classifier dropdown, go Trees -> J48
16. Click start, and if it asks to wrap the classifier in a “InputMappedClassifier”, select “Yes”

**Arff files and Model are saved under the directory “best\_model” in the project folder**

## **7. Team Members and Tasks Performed**

**Finding the Data & Building Proposal: Aidan and Max**

**Preprocessing Initial Attempt:** **Max**

**Preprocessing & Project Update: Max and Aidan**

**Non-Weka Attribute Selection Algorithm: Max**

**Attribute Selection Algorithms and Classifiers:** **Aidan**

**Results Output: Aidan**

**Results Analysis:** **Aidan**

**Building Final Report: Max and Aidan**

## **8. References and Sources**

The website that we downloaded the dataset from is: <https://data.scorenetwork.org/hockey/nhl-shots.html>.

Additional files:

* nhl-shots-2021.csv — The raw dataset before altering anything.
* shots\_final.csv — The dataset after preprocessing. All the unnecessary attributes and missing values are removed. Additionally, some data is normalized.
* test\_set.csv, train\_set.csv, val\_set.csv — The training, testing, and validation sets before the attribute selection
* best\_model (dir) - files used by the best model