# Predicting the outcome of a shot in the NBA using Logistic Regression and Random Forest

Tomáš Belák

January 6, 2025

**Abstract** In this project I tried to predict the binary outcome of shots in the NBA. I used a dataset that contains every shot taken in the NBA 2024/15 season which has naturally occuring class imbalance. I trained 2 classifying models for comparison - Linear regression model and Random forest model. Both models showed properties of being better than classifying everything as the majority class or random guessing. As the model evaluating metrics showed, either model had different classifying strategies. However, compared to other works the achieved accuracies on the test set were average at around 61%.

## Contents

## 1 Introduction

Remember playing basketball outside with friends, shooting a shot, it looks like it is going in and then suddenly you feel a cold breeze across your face and watch your shot get offset by the wind ? Well, that never happens in the NBA (due to the invention of indoor basketball courts). NBA has its own variables that may or may not attempt the shot going in. Perhaps the most obvious ones would be the ability of the shooter himself, the distance from the basket, maybe also the distance of the nearest defender, the level of contest the defender puts up, maybe the importance of the moment, and I could go on.

In this project, I wanted to acknowledge these factors plus some more and try to predict the outcome of a shot in the NBA. For the project I will use a dataset of all shots taken throughout the NBA‘s 2014/15 season[2], on which I have done some feature engineering to extract features that I thought might help disecting the space and result in a more accurate model. Furthermore, I enriched the dataset by the players‘ overall ability rating obtained from the videogame NBA 2K15[12] that was released in October 2014.

I used the data to train a default Logistic regression and then fine-tuned a Logistic regression model and Random forest model to see If I could beat the results achieved with the baseline model that classifies everything as the majority class and random guessing model.

## 2 Data

For this project I used a dataset of valid shots in the NBA taken throughout the 2014/15 season, which is available on the Kaggle platform[2]. The dataset keeps record of 128069 shots taken by 281 unique players, where 54.8% of shots were missed and just 45.2% were scored which shows slight imbalance of class representation within the dataset. The dataset contains the following useful features and more:

- MATCHUP - information about the date and between which 2 teams the game took end,
- LOCATION - represents whether the shooter‘s team played at home or away,
- W - represents whether the shooter‘s team won or lost the game,
- FINAL_MARGIN - represents the difference of scores at the end of the game ( positive if the game was won, negative if lost),

- SHOT_NUMBER - order in which the shooter has taken the corresponding shot,
- PERIOD - value representing the period when the shot was taken. The regular number of periods is 4, anything more means that the game went into overtime,
- GAME_CLOCK - valid value between 0 and 12 minutes. Represent how much time was left in the quarter.
- SHOT_CLOCK - valid value between 0 and 24 seconds. Represents how much time was left on the shot clock.
- DRIBBLES - the number of dribbles a player had made before taking the shot
- TOUCH_TIME - how much time did the player who took the shot actually had held the ball prior to the shot. Valid values are the same as for SHOT_CLOCK because,
- SHOT_DIST - distance in feet between the shooter and the basket,
- PTS_TYPE - determines wether it was a 2 point shot or a 3 point shot,
- CLOSEST_DEFENDER - the name of the closest defender,
- CLOSE_DEF_DIST - distance between the shooter and the closest defender,
- PLAYER_NAME - the name of the shooter,
- FGM - binary value that represents whether the shot was made(1) or missed(0).

The author of the dataset states that the data were obtained via the NBA REST API.

## 2.1 Pre-processing

Just by looking at the data I noticed there was a significant amount of SHOT_CLOCK data missing. In the NBA games, when the time remaining on the game glock is less then 24 seconds (maximum of the shot clock) the shot clock is actually turned off[14], therefore for some missing values there was an easy fix with filling it with the time of the game clock. For the other missing values I filled it with the median SHOT_CLOCK value of the shooter‘s team in an effort to represent the teams tendencies on offense.

Via the describe method avaialable in Pandas library I validated the values of numerical data. From the min/max values I found out that TOUCH_TIME had around 320 corrupt values where some of them were negative and some greater then 24. I treated the shots with TOUCH_TIME greater then 24 as last hope attempts to shoot before the shot clock expired and therefore I rounded them to 24. If the TOUCH_TIME value was negative I tried to impute the values with random numbers from a representative distribution. If the player took 0 dribbles before shooting it indicates a so called 'catch & shoot' situation when the player shoots as soon as he catches the ball. The distribution of TOUCH_TIME in these situations was positively skewed, with most values being between 1-2 seconds (see figure n.1), therefore I used a fitted gamma distribution to impute. On figure n.2 is the distribution after imputation of missing values. For all other situations I used uniform distribution of range 0 to 24 - SHOT_CLOCK.

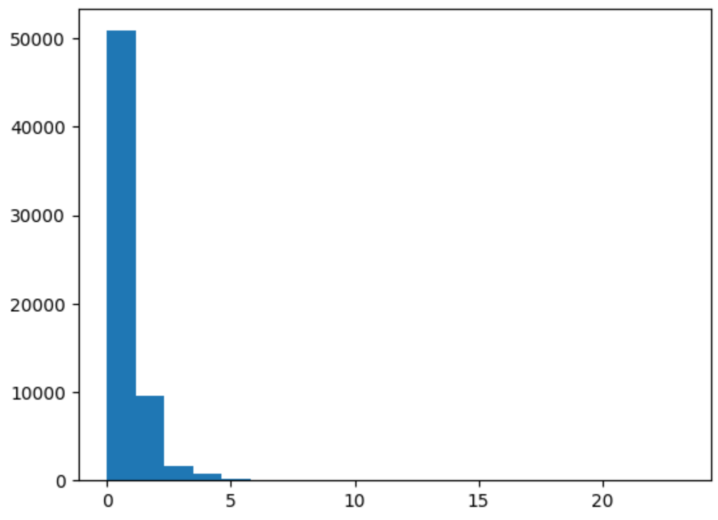

Figure 1: Distribution of touch time when zero dribbles before imputation

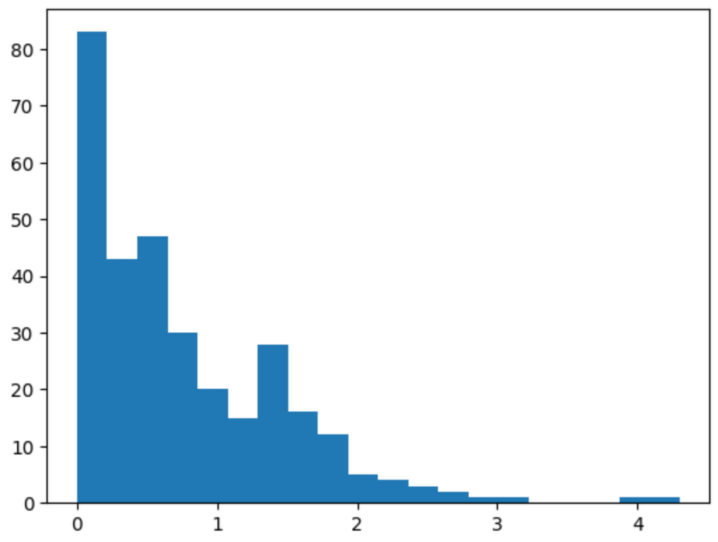

Figure 2: Distribution of imputed touch time when zero dribbles for the NA TOUCH_TIME values

After fixing missing data, I engineered more features that I hoped to be useful. Here they are:

- SHOOTER_FG_PERCENTAGE - # made shots / # total shots (taken by the player)
- DEFENDER_FG_PERCENTAGE - # missed shots / # total shots (defended by the player)
- CONTEST_LEVEL - a feature inspired again by the videogame NBA2K which categorically reperesents the defender's effort to defend the shot. We have to assume that the defender is always positioned in front of the shooter. It was calculated as following:
  - HEAVY CONTEST - if the nearest defender was within 2 ft of the shooter,
  - CONTEST - if the nearest defender was within 2-4 ft of the shooter,
  - SLIGHT CONTEST - if the nearest defender was within 4-6 ft of the shooter,
  - OPEN - if the nearest defender was further than 6 ft from the shooter

- CONTEST_LEVEL_FG_PERCENTAGE - # made shots / # total shots (grouped by CONTEST_LEVEL)
- PLAY_TYPE - feature inspired by a similar work done by students in Spain[13] which labels the shot with a type of play that lead to the shot. It follows these rules:
  - **no dribbles and shot distance within 5 ft** = CUT : player profile called Slasher would typically excel at doing these. It means running towards the basket, receiving a pass and finishing close to the rim.
  - **no dribbles and shot distance greater then 5 ft** = CATCH&SHOOT : some players are catch&shoot specialists and produce significantly better outcomes in these situations,
  - **more than 5 dribbles** = ISO : done typically by the best player on the team as the player in possession creates the shot for himself by outdribbling the opponent,
  - **atleast one dribble and shot distance within 5 ft** = DRIVE : Situation very similar to the CUT but involves dribbling. Also something a Slasher player profile would produce efficienlty from.
  - **atleast one dribble and shot distance further than 5 ft** = PULLUP : Very difficult shot as the shooter shoots off-balanced while basically still running. It often happens when the shot clock is about to exipre.
  - **everything else** = OTHER.
- PLAY_TYPE_FG_PERCENTAGE - # made shots / # total shots (grouped by PLAY_TYPE)
- IS_CLUTCH - According to wikipedia the quality of being 'Clutch' in sports refers to the phenomenon when athletes excel under pressure. I defined every shot as clutch that was taken in the last 8 minutes of the 4th quarter of a tight game (final margin no more than 8 points) or at any stage in overtime.
- SHOT_CLOCK_PRESS - feeling the pressure of soon to be expired shot clock. Every shot taken within the last 2 seconds of shot clock.
- CORNER_THREE - A 3-pt shot from a corner is the shortest 3-pt shot in the NBA. The so called Corner Specialists have a knack for shooting from the corners.

- IS_FASTBREAK - If a shot was taken within the first 7 seconds of the shot clock, I laballed it as it was on a fast break. Fast break would typically mean the defense is not positioned yet which could result in a open shot or the defender was chasing the shooter therefore positioned behind the shooter and possibly unable to contest the shot (I did not mirror this thought process in the data as it would mean recalculating the shot contest level based on too many assumptions).

I also enriched the data by representing players‘ general basketball ability in the 2014/15 season by their overall rating in the videogame NBA2K15. I obtained the ratings by scraping the HoopsHype website[12] using MS EXCEL‘s Get data via web feature. I joined the two datasets on the names of the players. Doing this I obtained a numerical feautre representing both the shooter‘s and defender‘s objective basketball ability.

## 2.2 Train and test sets

I split the preprocessed dataset into train and test datasets in 80-20 ratio. Meaning out of the total 128069 rows the train set now contained 102455 rows with classes represented in the same ratio as the original dataset meaning 54.8% were missed shots and 45.2% were made shots. Test set contained the remaining 25614 rows with the same class representation as the original dataset. This was achieved by using the train_test_split method available in sklearn library[9] and specifying the stratify parameter passed to the method. Validation set was not needed as the training during tuning will utilize cross-validation.

# 3 Context of my work

There has been quite a lot of work done on this dataset already. I have looked at one work by Toluwanimi Olorunnisola on Kaggle[10]. What I did not like about his work was the lack of feature engineering and thought it could improve the results. Another work that I have seen on github[13] which I already mentioned because it inspired me to engineer PLAY_TYPE feature, was not about predicting the outcome but rather anaylsis of the features and they used Random Forest to analyze the importance of individual features in the training process which I liked. I also decided to find out the importance of features to see if any of the engineered features, that I thought could be benefitial, actually held some importance in the training process or not.

## 4 Description and justification of methods used

In this section I describe the models I trained and the methods used to train them. Before training, categorical values were encoded using the get_dummies method from pandas library. All numerical features other than PERIOD and non-binary features were normalized for Logistic Regression to work. Even though Random forest do not require normalized data, it was also trained on normalized data. Every model was trained using tools from the scikit-learn library available in Python [6][8][5] and evaluated by metrics such as ROC AUC, precision, recall, log-loss whcih are available in the scikit-learn metric library[7].

### 4.1 Logistic Regression

Logistic regression is a linear model used for binary classification tasks. It predicts the probability of a binary outcome by modeling the relationship between the dependent variable (target) and independent variables (features) using the logistic function[11].

To optimize the logistic regression model, I used GridSearchCV, which performs an exhaustive search over a specified parameter grid. This method evaluates all possible combinations of hyperparameters using k-fold cross-validation to avoid overfitting. The scoring metric used was ROC AUC (described in the upcomming subsection 4.3), which measures the model's ability to distinguish between the two classes.

**Parameters Tuned**

- **C (Regularization Strength)**:
  - Represents the inverse of regularization strength.
  - Smaller values of C increase regularization, penalizing large coefficients to prevent overfitting.
  - Larger values of C reduce regularization, allowing the model to fit the training data more closely.
  - Values tested: [0.001, 0.01, 0.1, 1, 10, 100].
- **max_iter (Maximum Iterations)**:
  - Defines the maximum number of iterations for the solver to converge.

  - If the model does not converge within the specified iterations, it may underperform.
  - Values tested: [50, 100, 200, 500, 1000].

- **penalty** (Regularization Type):
  - Specifies the type of regularization to apply.
  - 'l1': Lasso regularization, which encourages sparsity by shrinking some coefficients to zero.
  - 'l2': Ridge regularization, which shrinks all coefficients proportionally.
  - None: No regularization is applied.
  - Values tested: ['l1', 'l2', None].

- **solver** (Optimization Algorithm):
  - Determines the algorithm used to optimize the logistic regression model.
  - 'lbfgs': A quasi-Newton method that is robust and efficient for small to medium-sized datasets.
  - 'liblinear': Supports both L1 and L2 regularization.
  - Values tested: ['lbfgs', 'liblinear'].

- **class_weight** (Class Weight):
  - Handles class imbalance by assigning different weights to classes.
  - None: All classes are treated equally.
  - 'balanced': Automatically adjusts weights inversely proportional to class frequencies.
  - Values tested: [None, 'balanced'].

- **tol** (Tolerance for Convergence):
  - Defines the tolerance for stopping criteria during optimization.
  - Smaller values require the model to achieve a more precise solution, potentially increasing training time.
  - Values tested: [1e-4, 1e-3, 1e-2].

**Justification**

- **C**: Controls the trade-off between model complexity and overfitting. A wide range of values ensures flexibility in finding the optimal regularization strength.
- **max_iter**: Ensures the solver has sufficient iterations to converge, especially for complex datasets or smaller regularization strengths.
- **penalty**: Allows exploration of different regularization types to determine the most effective approach for the dataset.
- **solver**: Different solvers are suited for different types of datasets and regularization methods. Testing multiple solvers ensures the best algorithm is selected.
- **class_weight**: Addresses class imbalance.
- **tol**: Balances precision and computational efficiency during optimization.

## 4.2 Random Forest

Random forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees. It is robust to overfitting, handles non-linear relationships well, and provides feature importance scores[3].

Similar to logistic regression, I used GridSearchCV to fine-tune the random forest model. The same scoring metric (ROC AUC, which is desribed in subsection 4.3) was used to evaluate performance, ensuring consistency in model comparison.

**Parameters Tuned**

- **n_estimators** (Number of Trees):
  - Represents the number of decision trees in the forest.
  - More trees generally improve performance but increase computational cost.
  - Values tested: [50, 100, 200].
- **max_depth** (Maximum Depth of Trees):

  - Controls the maximum depth of each tree.
  - Deeper trees can capture more complex patterns but may overfit.
  - None allows trees to grow until all leaves are pure or contain fewer than min_samples_split samples.
  - Values tested: [None, 10, 20, 30].

- **min_samples_split** (Minimum Samples to Split a Node):
  - Specifies the minimum number of samples required to split an internal node.
  - Higher values prevent overfitting by limiting tree growth.
  - Values tested: [2, 5, 10].

- **min_samples_leaf** (Minimum Samples at Leaf Nodes):
  - Defines the minimum number of samples required to be at a leaf node.
  - Higher values regularize the model by preventing the creation of leaves with few samples.
  - Values tested: [1, 2, 4].

**Justification**

- n_estimators balances model performance and computational efficiency[4].
- max_depth, min_samples_split, and min_samples_leaf control the complexity of individual trees, preventing overfitting and improving generalization[4].

## 4.3 ROC AUC as the evaluation metric

The ROC AUC metric measures the model's ability to distinguish between positive and negative classes. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings and calculates the area under this curve[1]. AUC is a value between 0 and 1. If a model produces AUC value above 0.5 it is deemed to be an improvement on a model that just guesses randomly[1].

## Justification

- ROC AUC is particularly useful for imbalanced datasets, as it focuses on the ranking of predictions rather than their absolute values[4].
- It provides a single metric to compare models, making it easier to select the best-performing configuration[4].

## 4.4 GridSearchCV and Cross-Validation

GridSearchCV is a hyperparameter tuning method that evaluates all possible combinations of hyperparameters in a predefined grid. It uses k-fold cross-validation (CV) to assess model performance, ensuring that the results are robust and not dependent on a specific train-test split[5].

- **cv=5**: The dataset is split into 5 folds, with each fold used once as a validation set while the remaining 4 folds are used for training.
- **scoring='roc_auc'**: The ROC AUC metric is used to evaluate model performance, as it is well-suited for imbalanced datasets and provides a comprehensive measure of classification performance.
- **n_jobs=-1**: Enables parallel processing to speed up the grid search by utilizing all available CPU cores.
- **verbose=2**: Provides detailed logs during the grid search process for monitoring progress.

## Justification

- GridSearchCV ensures that the best hyperparameters are selected systematically, improving model performance[4].
- Cross-validation reduces the risk of overfitting and provides a more reliable estimate of model performance on unseen data[4].

## 4.5 Description of technical issues

I had one issue when trying out various combinations of parameters when tuning the logistic regression. I have found out that some parameters are not compatible with each other like for example not every solver is able to do every regularization technique. It was not really an issue because the GridSearchCV tool enabled continuity of tuning process even though some fits failed due to the incompatibilies I described.

## 5 Experimental evaluation

In this section I discuss the results of my work. In this section I believe it is important to state the fact that a class represenation imbalance is present across the entire dataset. In both, the train set and the test set, the classes were represented identically 54.8% missed, and 45.2% made. Therefore if a model just predicted all shots to be missed it would yield an accuracy of 54.8% - I will call this the baseline model in this section.

In table n.1 we can see the metrics for the baseline logistic regression model which was initialized with default parameters, tuned logistic regression and tuned random forest models. From the table, and also as the figure n.3 shows, both tuned models had a ROC AUC score above 0.5 which means they are an improvement on random guessing. Looking at the accuracy metric in the metrics table it suggests that all of the models also improved on the baseline model.

Lets analyze each model‘s metric. Starting with the Random forest model, it had the highest ROC AUC score which could indicate better performance on not yet seen data. This appears to be true as it also scored the highest accuracy on the test set. The recall value is rather disappointing at around 0.34. This means that out of all the made shots, only around 34% get classified as made with 65% accuracy. With precision at around 0.65 and recall so low the Random forest model appears to have established a similar strategy as the baseline model but with an occassional guess that the shot was made which is also visible on the figure n.4. Also, by taking a look at the confusion matrix at figure n.5 we can see that the Random forest model is better at classifying the majority class - missed shots. I think these facts makes the accuracy metric misleading, because if this model was deployed on a NBA match with above 50% success (which is rare), the model would perform poorly. It relies too much on the class imbalance. However, the imbalance is caused by the nature of the NBA and reflects the reality therefore I am not sure if this should be considered to be a negative aspect of the model.

The Default Linear Regression model has similar behavior as the Random forest model. The tuned Logistic Regression model has a more balanced philosophy as the similar precision and recall values suggest. With the precision value at around 0.56 and recall value at around 0.53 it appears to be slightly better than random guessing on the imbalanced dataset where we would expect precision and recall to be the same as the proportion of positives in dataset - 0.452. Finally, all models have a very similar Log-loss metric at around 0.65. A model guessing at random on a dataset with the same class representation imbalance would be around 0.69. This can be calculated using the log-loss formula. Since the models have lower log-loss, it means they picked up on some patterns and are really slightly better then random guessing.

| **Metric** | **Default LR** | **Logistic Regression** | **Random Forest** |
|---|---|---|---|
| ROC AUC | 0.637570 | 0.637609 | 0.642657 |
| Accuracy | 0.613922 | 0.602288 | 0.619583 |
| Precision | 0.605776 | 0.563491 | 0.651343 |
| Recall | 0.418358 | 0.534151 | 0.341335 |
| F1-Score | 0.494918 | 0.548429 | 0.447932 |
| Log-Loss | 0.654361 | 0.658479 | 0.649295 |

Table 1: Performance metrics for default Logistic Regression, tuned Logistic Regression, and tuned Random Forest models.

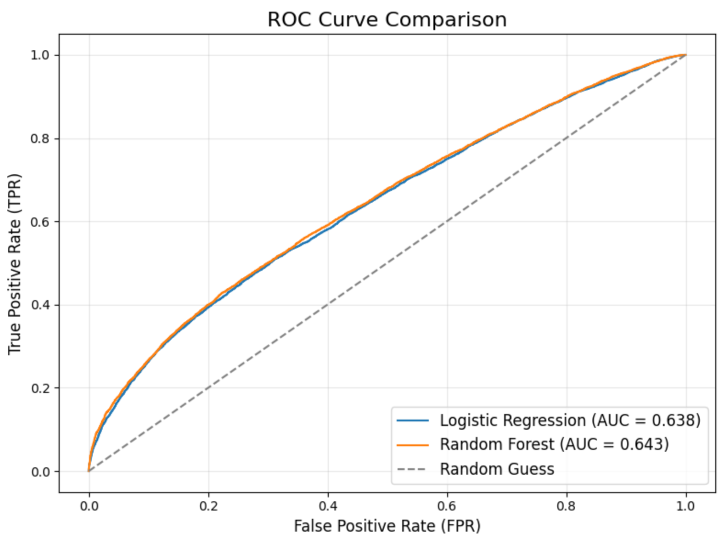

Figure 3: ROC curve for tuned LR and tuned RF models

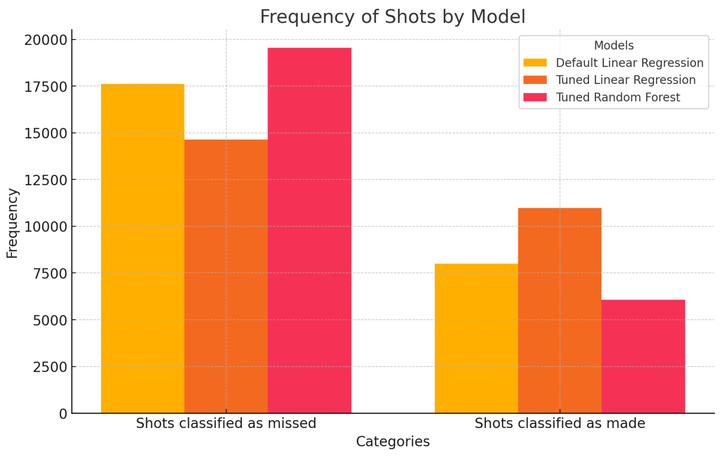

Figure 4: Model predictions frequencies on the test set

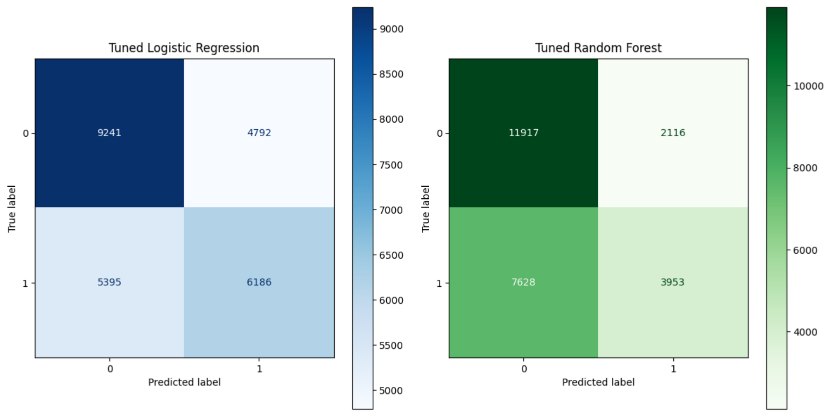

Figure 5: Confusion matrix of the tuned LR and RF models

As I mentioned earlier, I wanted to see the importance of the engineered features in the Random Forest model. In the figure n.6 we can see that some engineered features were quite useful for the model but some not so much. We can see that the PLAY_TYPE_FG_PERCENTAGE engineered feature was actually the second most important as it described the probability of a shot being made if it came from a specific type of play. The fourth most important feature was the PLAY_TYPE_CUT because shots with this attribute were made on 66% of occasions and therefore correlated with the target variable. To my surprise the feature RATING_DIFF was not as important as it described the difference in overal basketball ability of the shooter and defender. However, the model chose to give the SHOOTER_FG_PERCENTAGE and DEFENDER_FG_PERCENTAGE features higher importance, because they basically describe the ability of the players in the offensive and defensive role respectively and therefore are higher correlated with the target variable than RATING_DIFF. The features with lower importance were probably redundant meaning they were highly correlated with other more important features, like for example the IS_CLUTCH or IS_CORNER_THREE features.

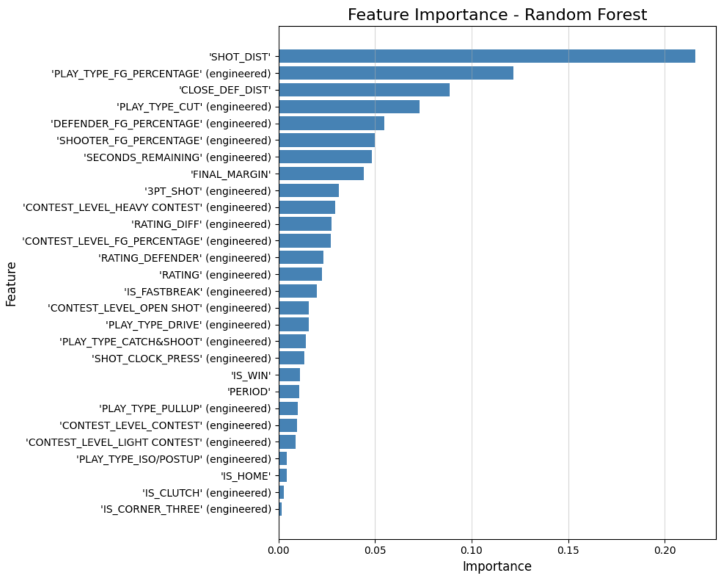

Figure 6: Feature importance in the Random forest model

## 6 Conclusion

In this project I managed to find models that are that are more accurate than random guessing or majority class classifying. Given the complex nature and uncertainty when it comes to the success of a shot taken in the NBA I would say the project was quite successful because of that. I got to tryout training Logistic regression and Random forest models, evaluate their performances and compare them. From what I saw in other works the accuracies I achieved were average. In this perspective the project was not really ground breaking. If I was to do the project again I would allocate time differently. I remembered the words of doctor V.Broza during one lecture where he said that in practice we gain more by paying attention to our data and cleaning it rather than tuning models. However, I believe I spent too much time ineffectively on feature engineering and trying to be smart with it or maybe too little time on features which could have made a difference. First I tried to engineer an angle based shot contest level but I decided to ditch the idea when I saw the contest level labels it assigned and realized it was faulty when the shots were taken from longer distances. Then I created features highly correlated with other features. If I did it again I would try different approaches and engineer different ’deeper’ features.

## References

[1] Evidently AI. *Explain ROC Curve — Evidently AI*. Accessed: 2025-01-05. 2025. URL: `https://www.evidentlyai.com/classification-metrics/explain-roc-curve`.

[2] Dan Becker. *NBA Shot Logs*. Accessed: 2024-12-20. 2015. URL: `https://www.kaggle.com/datasets/dansbecker/nba-shot-logs/data`.

[3] Applied AI Course. *Random Forest Algorithm in Machine Learning*. Accessed: 2025-01-05. 2025. URL: `https://www.appliedaicourse.com/blog/random-forest-algorithm-in-machine-learning/`.

[4] AI DeepSeek. *DeepSeek-V3*. `https://www.deepseek.com`. 2024.

[5] Scikit-learn Developers. *GridSearchCV — Scikit-learn Documentation*. Accessed: 2025-01-05. 2025. URL: `https://scikit-learn.org/1.5/modules/generated/sklearn.model_selection.GridSearchCV.html`.

[6] Scikit-learn Developers. *LogisticRegression — Scikit-learn Documentation*. Accessed: 2025-01-05. 2025. URL: `https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html`.

[7] Scikit-learn Developers. *Metrics API — Scikit-learn Documentation*. Accessed: 2025-01-05. 2025. URL: `https://scikit-learn.org/1.5/api/sklearn.metrics.html`.

[8] Scikit-learn Developers. *RandomForestClassifier — Scikit-learn Documentation*. Accessed: 2025-01-05. 2025. URL: `https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html`.

[9] Scikit-learn Developers. *train_test_split — Scikit-learn Documentation*. Accessed: 2025-01-05. 2025. URL: `https://scikit-learn.org/1.5/modules/generated/sklearn.model_selection.train_test_split.html`.

[10] Tolulope Gboyega. *NBA Dataset Modelling*. Accessed: 2024-12-20. 2023. URL: `https://www.kaggle.com/code/tolugboyega/nba-dataset-modelling`.

[11] GeeksforGeeks. *Understanding Logistic Regression.* Accessed: 2025-01-05. 2025. URL: https://www.geeksforgeeks.org/understanding-logistic-regression/.

[12] HoopsHype. *NBA 2K Ratings for the 2014-2015 Season.* Accessed: 2025-01-02. 2015. URL: https://hoopshype.com/nba2k/2014-2015/.

[13] Sergio Llana. *NBA Shot Analysis.* Accessed: 2024-12-30. 2023. URL: https://github.com/SergioLlana/nba-shot-analysis.

[14] Wikipedia contributors. *Shot Clock — Wikipedia, The Free Encyclopedia.* Accessed: 2025-01-05. 2025. URL: https://en.wikipedia.org/wiki/Shot_clock.

## Appendix

Here is a link to my github repository with all the files needed: https://github.com/tomasbelak24/matfyzuk-ml-project