Predicting Mixed Martial Arts Results using Machine Learning techniques

Emmanouil Lykos

February 2022

1. Introduction

In the recent years, Mixed Martial Arts gained a massive amount of fans around the world. This happened through some organizations that established their own brand in the sport, such as Ultimate Fighting Championship(UFC), or through certain people who know how to do a great marketing for themselves in order to make people curious about their next fight, such as Conor McGregor, thus, attracting them to buy tickets or Pay Per Views(PPVs) to be able to watch it online. Like every other popular sport, many fans see the possibility of getting money through betting[1] or just leveling up in Verdict app.

In the research community, Machine Learning also got massive popularity among other areas. Machine Learning, is a sub-field of Artificial Intelligence that deals with the study of computer algorithms that can improve automatically through experience and by the use of data, in a given task. Because of the current computational efficiency and the rapid interest in artificial intelligence, many companies also saw this as a solution method in their problems and many people as a method of solving a variety of problems. One application of Machine Learning can also be the prediction of the result of various sports events, such as Mixed Martial Arts(MMA) fights. In order to be able to do that we need mostly to create models that excel into the classification task, i.e. we want to classify each fight to some result.

In this report I will present my approach on a predictor who will predict the fighter that will win in an MMA fight, through Machine Learning. In this project I dealt with the whole Machine Learning project cycle, i.e from writing crawlers to acquire data, until the evaluation of the possible models that we can create, and ultimately the communication of our results. The structure of the rest report is the following. In Section 2, I will present my methodology from configuring my dataset until training and evaluating candidate models in order to ultimately choose the best one. In Section 3, I will present the experimental results that are shown from my methodology and then explain why they are these ones. Finally, I will write my conclusion and also present directions for further work.

2. Methodology

2.1. Feature Engineering & Exploratory Data Analysis

Initially, because I did not had a dataset available and also I did not found a comprehensive one on Kaggle, I took the decision to scrape my data. Initially, I scraped the past fights data from ufcstats web page, but only the total statistics and not the ones that contains the split among the significant strikes(example page). Afterwards, I took the current statistics of each

[1] **Disclaimer:** I do not promote or encourage any betting activity, especially using the models presented in this paper. This application was built for educational purposes in order to see the capability of Machine Learning into this particular task.

fighter(example page). Then, in order to be able to train my models, I needed to find the values of the features present on fighters page, but on the time of the fight. In order to do that, I just wrote a Python script that generates these features. Thus, each dataset's instance has the following, common for each fighters, features:

1. Fight ID
2. Fight Date
3. Gender: could be male or female.
4. Weight Class
5. Title Fight: Whether this fight was a title fight.
6. Result: The result according to the fighter that is first written in the vector. Could be win, lose or draw.
7. Method: The ending method, e.g. Decision, K.O. etc.
8. Round: How many rounds the fight lasted.
9. Time: At what time the end of the fight happened.
10. Fight Time Format: could be 3Rnd(5-5-5) or 5Rnd(5-5-5-5-5), because in MMA the fights last 3 or 5, 5-minute rounds.

and then there are the features of each fighter. Thus, the following features are present to the data twice(one time for each fighter). Note, that the following features have pretty representative names:

1. Fighter's ID
2. Fighter's Name
3. Fighter's Nickname
4. Fighter's Age(at time of the fight)
5. Fighter's Wins(at time of the fight)
6. Fighter's Loses(at time of the fight)
7. Fighter's Draws(at time of the fight)
8. Fighter's Average Time in minutes.
9. Fighter's Height: in feet, but it is converted to centimeters
10. Fighter's Reach: in inches, but it is converted to centimeters
11. Fighter's Stance
12. Fighter's Significant Strikes Landed per Minute.
13. Fighter's Striking Accuracy: it is a percentage, so its a number in [0,100].
14. Fighter's Significant Strikes Absorbed per Minute
15. Fighter's Defense: it is a percentage, so its a number in [0,100].

16. Fighter's Takedown Average per 15 Minutes
17. Fighter's Takedown Accuracy: it is a percentage, so its a number in [0,100].
18. Fighter's Takedown Defense: it is a percentage, so its a number in [0,100].
19. Fighter's Submission Average per 15 Minutes

Also, note that Result, Method, Round and Time features can be used only as label features. However, in our case we keep only the Result feature and drop the others, because we want for now for our model to predict the winner of a given fight.

Because the above features, seem to be pretty descriptive and useful, I thought that it will be a pretty good start if I just keep the aforementioned features except the "label" ones and the Fight ID, Fight Date, Fighter ID, Fighter Name and Nickname. Although we do not combine or create new features, we can do the following transformations in the dataset, before dropping the aforementioned features:

1. Double the instances of the dataset(aka **Double Dataset**): This is done by inserting rows that derive by taking each row's features, swap the position of fighters features and finally reversing the result(for example, if result is win then new row's result is lose). Note that, this transformation should not be done before Train-Test split, because there is no point to do that in fights that we want to predict. The reasoning behind this is to firstly mitigate the consequences of having an imbalanced dataset, especially between win and loses, because it will get balanced and afterwards is shown to be pretty useful to pass the information that if some instance gets its features reversed, then it has the "opposite" result.

2. Take the difference of the attributes between the fighters that participate to an instance(aka **Difference Dataset**): This is done by keeping the common features and fighters' categorical features(such as ID, Name, Nickname and Stance) and taking the difference of the rest of the features. This can be done to reduce the dimensionality of the original dataset, which might result in models that are faster or be better in terms of predicting performance.

3. Applying the aforementioned transformations(firstly the doubling, then the difference)(aka **Double Difference Dataset**).

Now, in terms of Exploratory Data Analysis(EDA), I apply tSNE to the original and the transformed datasets. Before reducing, I did ordinal encoding to categorical features, scaled properly the rest and dropped the rest. tSNE's results are the following:

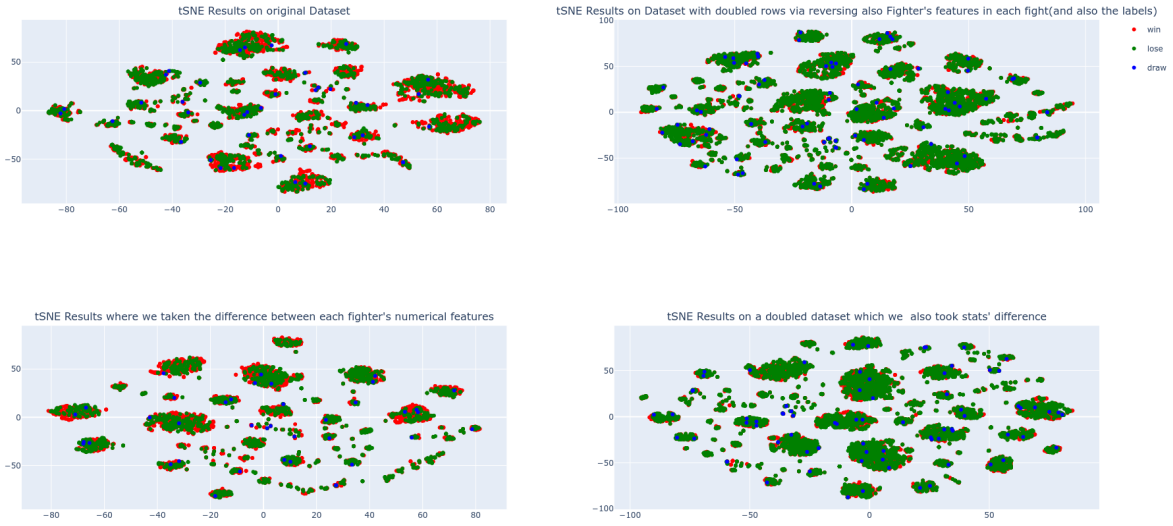


Figure 1: tSNE Results in Different Types of datasets

From the above figures, we can see that there are no obvious clusters between each label's instance and instances seem to be mixed in each cluster. This is normal because the matches in MMA are done through match making, instead of something like a draw, thus, in most fights the fighters are nearly skilled, hence the fight can go either way, and this is why fights near each other have different result. Another reason might be that the features are not sufficient or not be capable to capture sufficiently the fighter by himself. Afterwards, we will check if we have any correlation in our features. The correlation matrix of each dataset is the following:

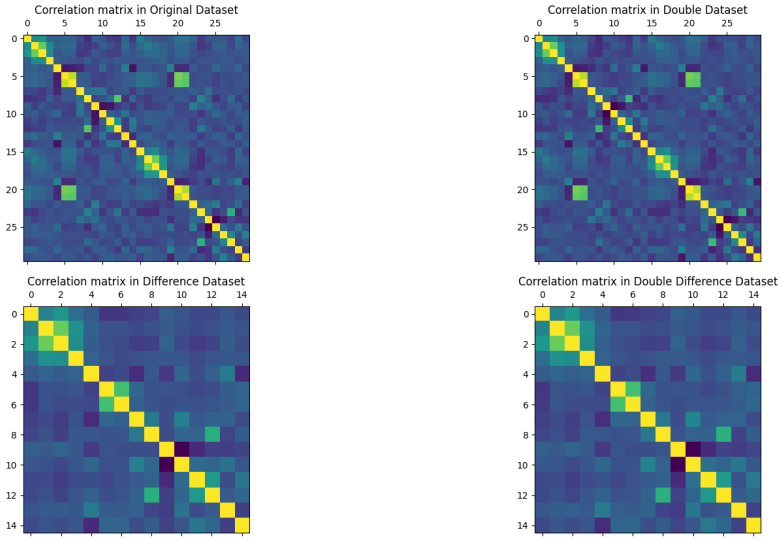


Figure 2: Correlation Matrix in Different Types of datasets

From the above figure, we can see that only Fighter's record features are highly correlated with each other, and also the Height and the reach. Thus, in the rest of our features we can see that we do not have any highly correlated features.

2.2. Training Preprocessing

After EDA, we should proceed to training my models. But, before that, we need to do some preprocessing on our data. The preprocessing that I do on the data is the following:

1. Removing irrelevant columns, such as dates, IDs etc.
2. Converting categorical features to numbers like an ordinal encoder.
3. Scaling properly numerical or percentage features with *MinMaxScaler*.

Note that, that imputing NaN values, lower-casing string features and converting height and reach to centimeters was done the time I read the datasets.

2.3. Training & Evaluation Methodology

In terms of Training and Evaluation, our task is to find the best classifier both in terms of predicting and efficiency performance, that excels in the given task. Initially, the metrics that I'll use for prediction performance will be the accuracy and F_1 score. I chose these two metrics, because accuracy can give us an initial overview of the efficiency of our models and micro-F_1 score, because it is more reliable than accuracy because it takes into account precision and recall metrics instead of a percentage of how many correct predictions the classifier did. Afterwards, given a specific classifier, we evaluate him using Repeated-K Fold Cross Validation and we determine ultimately the mean and standard deviation of accuracy and F_1-Score using the scores that each fold gave.

In order to find the best classifiers we get through two phases for each type of datasets that we showed in Feature Engineering section. Firstly, for each classifier we get with brute force through its given hyperparameters combinations in order to find the mean and standard deviation of accuracy and F_1 scores through the different folds. Secondly, we take each classifier with its best hyperparameters found from the previous step and then we compare those, not only in terms of accuracy and F_1, but also in terms of time performance by showing the training and prediction time per sample. Lastly, for those classifiers we also show their confusion matrix in order to have a better overview of the predictions done, because even with F_1 score, there is the case that it will be high but predict all the time only one class.

Now that we specified our methodology we are ready to proceed on showing our experimental results through that.

3. Experimental Evaluation

As mentioned before, the data that will be used for training and testing, were extracted by a crawler made by me with the use of Scrapy Framework. The initial data was each fight's statistic and general info(such as weight class, gender, is a title fights etc.). However, afterwards I converted it to a match-up form, i.e. each vector contained the statistics of a fighter prior to the actual fight. In terms of evaluating different classifiers-and finding the best one eventually- I evaluate each classifier separately in order to find with what hyperparameters they perform better and in what dataset. I tested the following classifiers:

1. Logistic Regression with different values of regularization strength C.
2. k-Nearest Neighbors with different values of k.

3. SVM with different values of regularization strength C.

4. Decision Tree with different values of maximum length.

5. Random Forest with different number of estimators.

6. AdaBoost with different number of estimators.

The results after doing Repeated k-Fold Cross Validation for 2 repeats in 6 Folds are the following(note that I did not did Time split validation because I do not have any feature that denote "time" or their timely manner is mandatory by another feature):

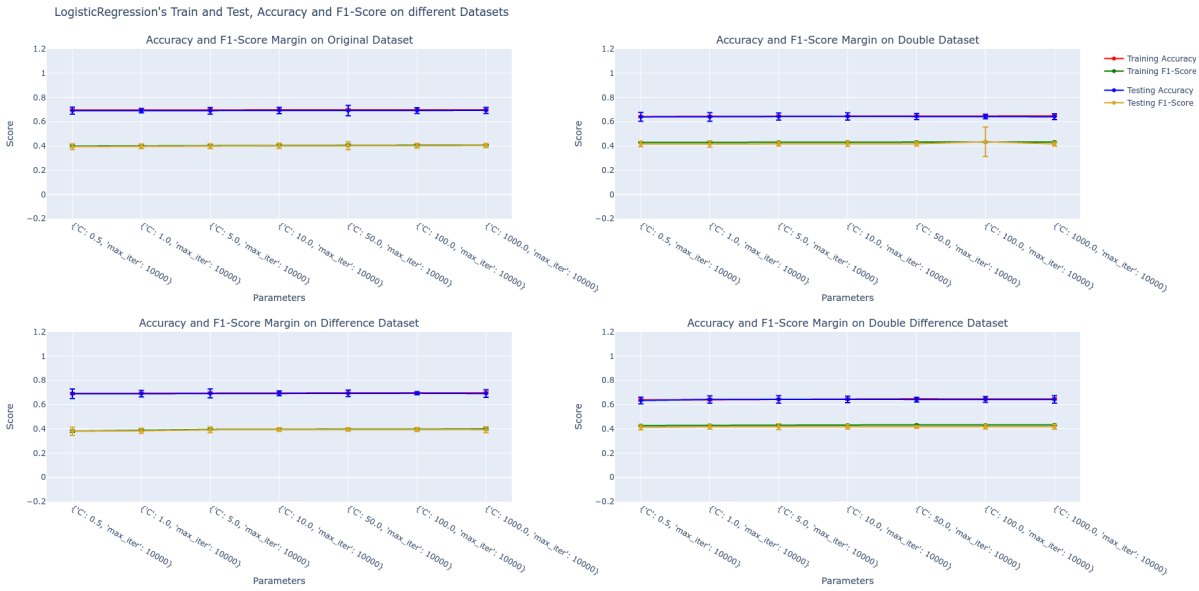


Figure 3: Logistic Regression's Mean Accuracy and F_1-Score through different hyperparameters with also 95% Confidence Intervals.

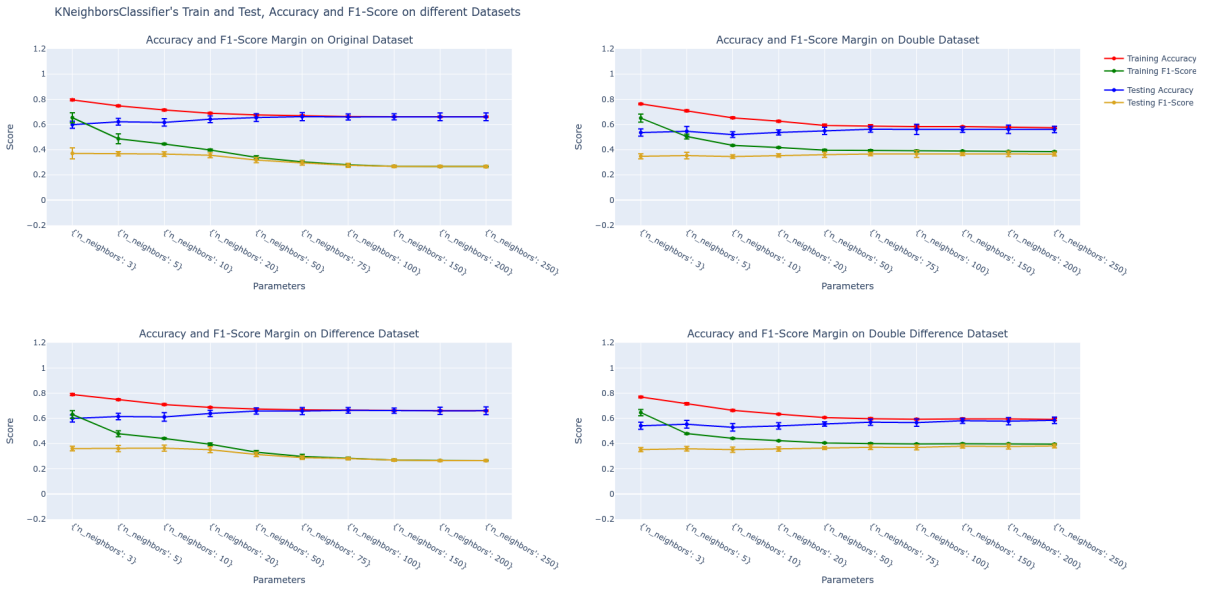


Figure 4: k-Nearest Neighbor's Mean Accuracy and F_1-Score through different hyperparameters with also 95% Confidence Intervals.

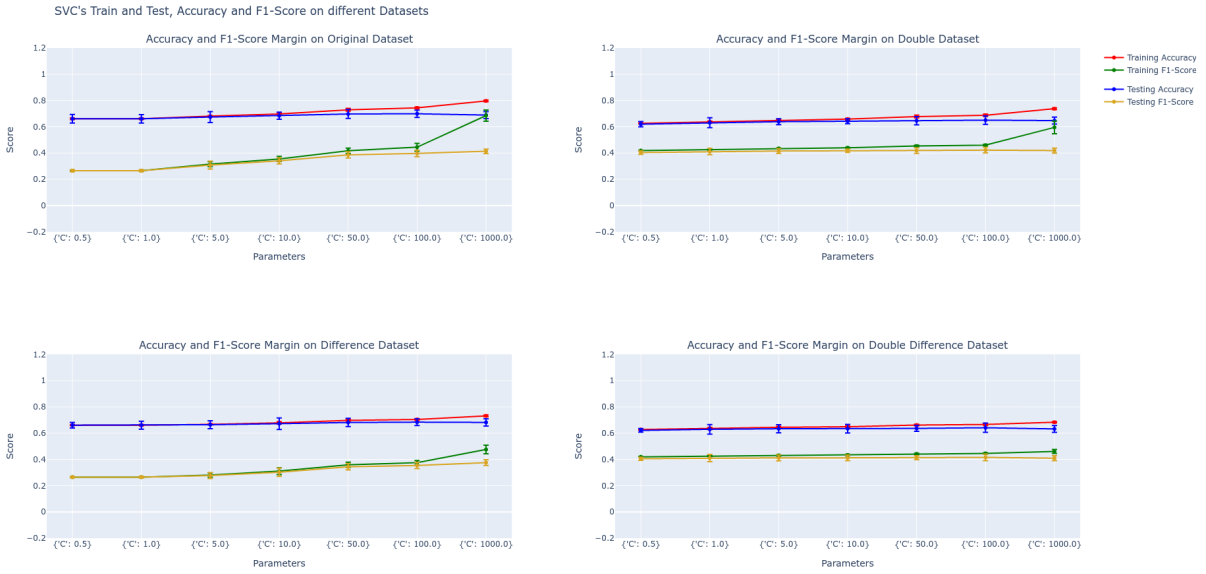


Figure 5: SVM's Mean Accuracy and F_1-Score through different hyperparameters with also 95% Confidence Intervals.

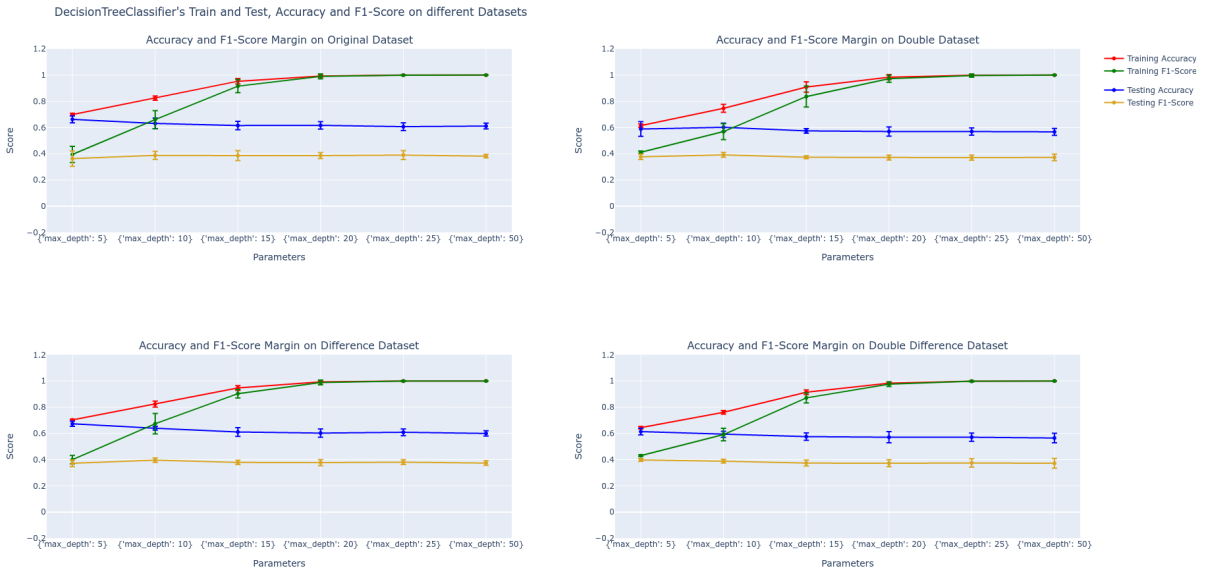


Figure 6: Decision Tree's Mean Accuracy and F_1-Score through different hyperparameters with also 95% Confidence Intervals.

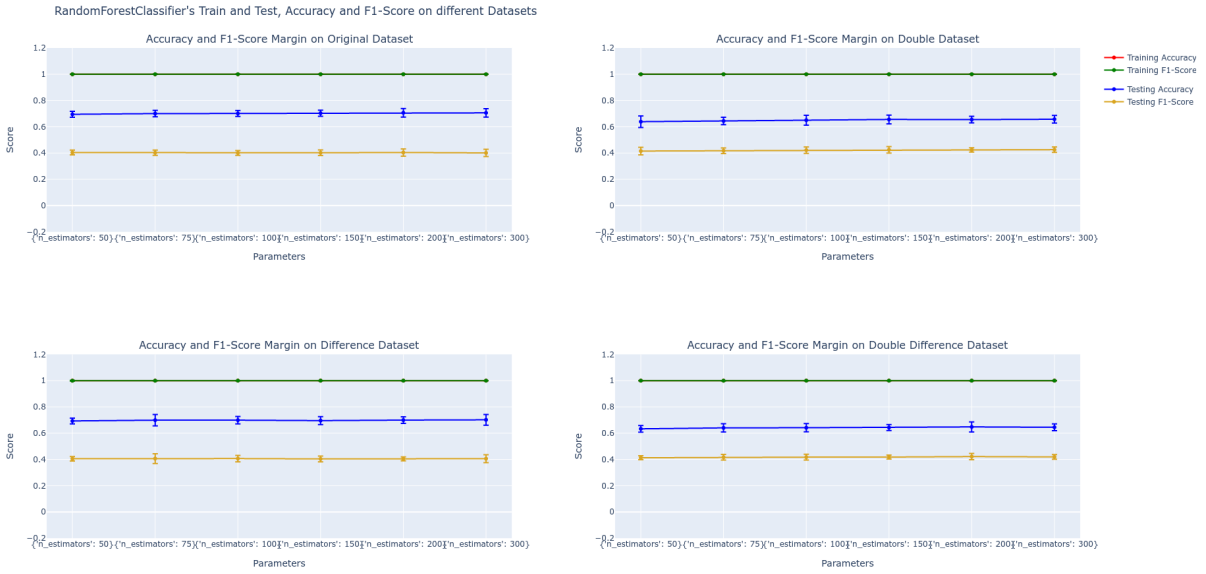


Figure 7: Random Forest's Mean Accuracy and F_1-Score through different hyperparameters with also 95% Confidence Intervals.

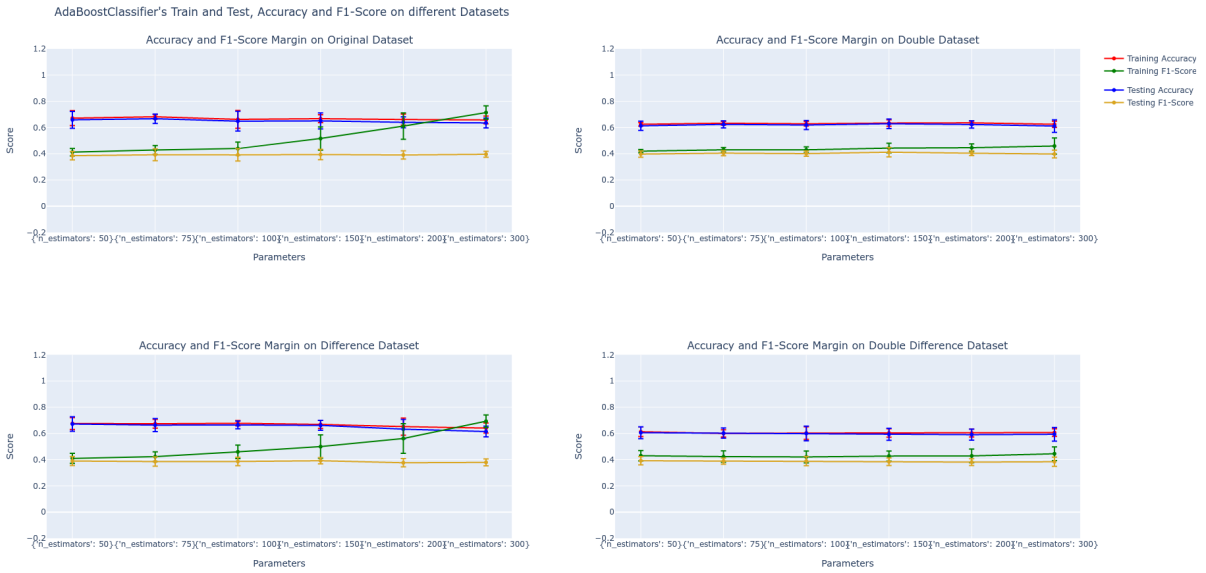


Figure 8: AdaBoost's Mean Accuracy and F_1-Score through different hyperparameters with also 95% Confidence Intervals.

From the above plots we can see that every classifier except logistic regression and k-Nearest Neighbors, after some value they overfit. Also, combining the 95% confidence intervals with mean accuracy and F_1 score, the best hyperparameters of each classifiers are the following:

1. Logistic Regression seem to have better overall performance with regularization strength $C = 1.0$ because even if with the other values the algorithm does not overfit with the specific one the intervals are stricter.

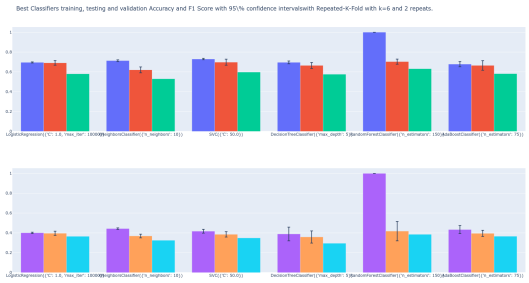
2. k-Nearest Neighbors seems to perform better with $k = 10$, because with $k = 20$ the algorithm overfits and the intervals are greater.

3. SVM seems to perform better with $C = 50.0$ because this is the point where his testing performance is significantly improved and does not overfit much.

4. Decision Tree seems to be better with max depth equal to 5 because at greater values we do not have any improvement in testing performance and the classifier overfits more.

5. Random forest performs better with 150 estimators in general because they have more narrow confidence intervals, so a more consistence performance.

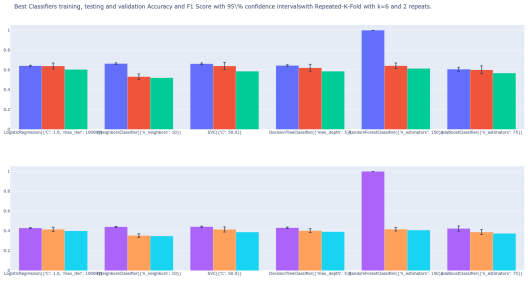
Afterwards, we will evaluate each classifier with their best hyperparameters, in terms of prediction performance, confusion matrix and efficiency, in each version of the dataset. Firstly, in terms of predictive performance, we will check the average training and testing accuracy and F_1 score with Repeated k-Fold cross validation with 6 folds and 2 repeats, but also the validation accuracy and F_1 score on a set of samples that our model never saw called validation set. Note that, our validation set are the last 500 instances of our dataset, i.e the 500 most recent fights according to our dataset. The results are the following:



(a) Accuracy and F1-Score of best classifiers in Original Dataset

(b) Accuracy and F1-Score of best classifiers in Double Dataset

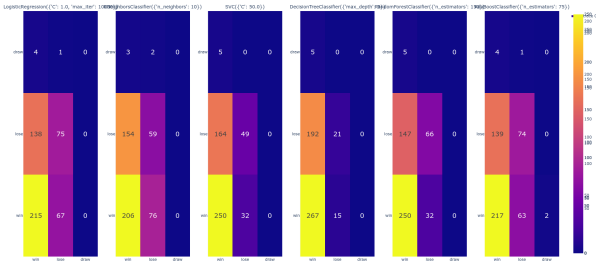
(c) Accuracy and F1-Score of best classifiers in Difference Dataset



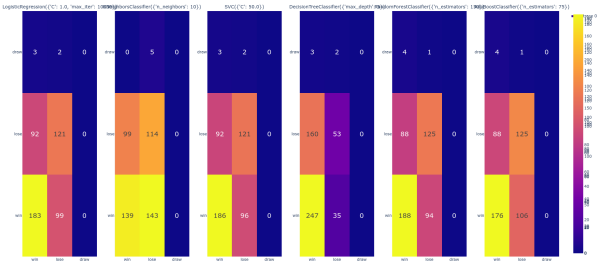
(d) Accuracy and F1-Score of best classifiers in Double Difference Dataset

Figure 9: Average and Standard Deviation of Accuracy and F1-Score of best classifiers in various datasets through Repeated k-Fold Cross Validation for 6 folds and 2 repeats.

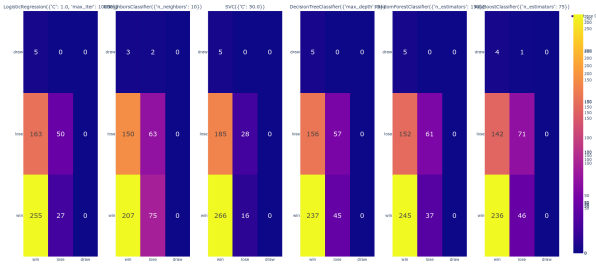
From the above figures we can see that, through the datasets, Logistic Regression, Random Forest and Random Forest yield the best performance with 60 and 40 per cent, accuracy and F_1 score, respectively. In terms of consistency, we can see that the classifiers are more consistent in Difference and Double Difference dataset because the 95% confidence intervals are more narrow on these datasets. Then, because there might be a case that our classifier might always just predict one label blindly and have a same or better predictive performance than a classifier that can predict every class, we should show the confusion matrices of each classifier, trained on each dataset version. The results on validation set, are the following:



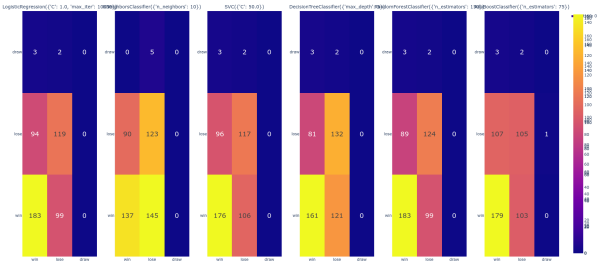
(a) Prediction Confusion Matrix after training with Original Dataset



(b) Prediction Confusion Matrix after training with Double Dataset



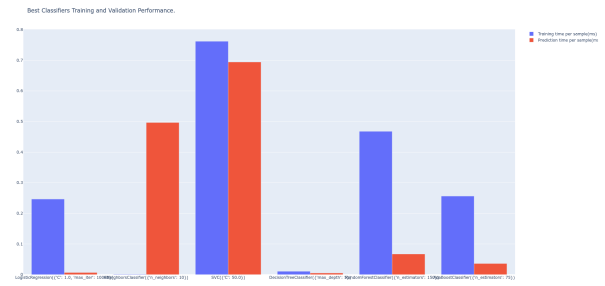
(c) Prediction Confusion Matrix after training with Difference Dataset



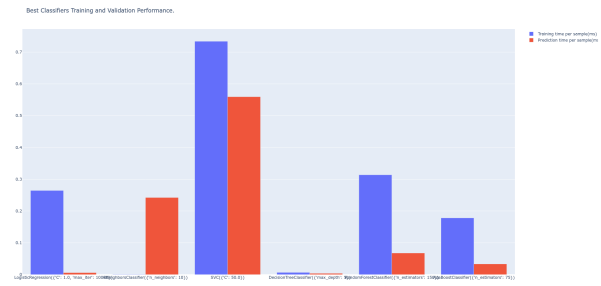
(d) Prediction Confusion Matrix after training with Double Difference Dataset

Figure 10: Confusion Matrices on validation datasets after training each best classifier in various datasets.

From the above graph, we can see that in Original and Difference Dataset all the classifiers are not the same capable at predicting the "lose" class than Double or Double Difference Dataset, which predict lose class with 50% accuracy, and less better the "win" class. This happens because in Double versions of the original dataset, the dataset is balanced in "win" and "lose" instances and also we pass the information that if we swap the features of each fighter, then we have the opposite result. Also, in terms of best classifiers we can see that Logistic Regression, Random Forest and Ada Boost have the best performance. Also, note that best performance is yielded on Double and Double Difference dataset. Finally, we will evaluate the classifier in terms of efficiency, but only on Double and Double Difference dataset, because only in these datasets, the classifiers seem to have a decent predictive performance by looking at the confusion matrix. Classifiers' performance is displayed in the below figure:



(a) Training and Prediction Performance in Double Dataset



(b) Training and Prediction Performance in Double Difference Dataset

Figure 11: Training and Testing efficiency per sample in milliseconds for best classifiers.

As it was expected because Double Difference dataset has less dimensionality, the classifiers were faster on Double Difference dataset, both in terms of training and prediction. Given that in Confusion Matrices these classifiers do not have much difference in predictive performance, I conclude that it is the best to train our model by transforming the Original dataset to a Double Difference one. Seeing each classifier's performance in confusion matrix, I conclude that the best classifier is the Random Forest with 150 estimators, because it seems to be slightly better at predict loses even if it is a bit slower, because time actually is not very crucial because we want to predict a small number of fights each time, and also we do not have a big dataset, in order to worry about training times.

In conclusion, I determined that the best classifier is Random Forest with 150 estimators and the original dataset to be transformed into Double Difference one. Although, this classifier achieves approximately 60% accuracy and 40% F_1 score meaning that this classifier does not seem to perform well, there are many explanations on why we cannot achieve accuracy like 85-90%, while the best model-found on web-achieved 75% accuracy. The main reason that we cannot achieve high accuracy scores is that the match-ups in MMA are not like other sports where a good team will face sometime a bad team. Instead, in MMA the fights are determined by a matchmaker who wants for the fights to provide good spectacle for the world, thus he chooses two fighters that are actually balanced in terms of skills, meaning that the fight can go either way. This means that in our dataset we do not have many fights that are sure wins or loses, in terms of statistics, and this is shown on tSNE plot, where we saw two fights that are near to each other and have a different result. Thus, the data cannot be clearly separated. Also, independently of how many features we will add and especially our current features(that are not much), we won't cover the fighter "as a whole" in order to pass what the people see in each fighter, such as special skills, cardio etc. which also is a hinder on having a better performance because some fighters might have some characteristics, except their numbers, that will give a better picture in order to predict the final result.

4. Conclusion & Further Work

In this report, I tried to use Machine Learning methods in order to predict Mixed Martial Arts fights. In order to achieve that, I got through the whole pipeline of a Machine Learning Project. Initially, I wrote the crawlers and the further configurations of the crawled data in order to create my dataset that I will train my models. Then, I did some feature engineering by proposing some transformations of my original dataset, and also preprocessing the different attributes in order to be ready for training. Afterwards, I train each of the chosen classifiers with a brute-force approach through different hyperparameters and datasets, in order to find with what hyperparameters those classifiers perform better. Finally, I evaluated each classifier in terms of predictive performance confusion matrix and F_1 score. Ultimately, I determined that the best classifier is Random Forest with 150 estimators in Double Difference dataset, but still has relatively low performance. However, low predictive performance can be justified because of the nature of each fight that can go either way and for the features, because are unable to cover each fighter as a whole.

In terms of further work I can go through different directions. In order to make my models stronger I can find more features by combining existing or create new ones, such as streak on last 5 fights, that could represent better a fighter. Another thing, was to try one hot encoding for categorical features. Also, I could extend my crawlers, or create new ones in order to extract more fights and more features. Finally, we could try more classifiers or even XGBoost in order to see if they can yield better results with the features I already have. Another direction of further work, is to create a model that can also predict the method of the ending of a fight and the round that this will happen. If that succeeds, then we will have a more powerful predictor

that has a better overview of the fight, and ultimately is capable to do full predictions in betting or in Verdict App.