Predicting Mixed Martial Arts Fights Outcomes Using Machine Learning: a Quantitative Betting Perspective

Maxime Rochkoulets
University of Bordeaux
maxime.rochkoulets@etu.u-bordeaux.fr
Dublin City University
maxime.rochkoulets2@mail.dcu.ie

Ryan Dempster
Dublin City University
ryan.dempster2@mail.dcu.ie

***Abstract*—This study explores the use of machine learning algorithms in predicting Mixed Martial Arts fights outcomes, as well as their potential application in betting strategies. Various models were trained on historical fight data. Although predicting the winner of fights has already been discussed in the literature, we demonstrate interesting results in predicting the round and manner in which an MMA match will conclude.**

1. Introduction

1.1. Motivation

Our goal is to predict with relative confidence the outcome of an MMA (UFC) fight based on the $k = 2^1$ previous fights of fighters *A* and *B*. To do this, we will train machine learning classification models. Using the probabilities given by the models, we want to see if they can be used for betting. Assume a model can give us $P_A = \mathbb{P}(\text{"fighter } A \text{ wins"})$, then, computing a positive

$$\mathbb{E}[X_A] = P_A(O_A - 1) - (1 - P_A)$$

with X_A the random variable that associates the gain of the bet with odds[2] O_A should give us the signal to take this bet.

1.2. Dataset

The biggest organisation promoting MMA is currently the UFC, it is also the one for which the most data is available online. We used a dataset available on Kaggle[3] containing data that was collected from the website ufcstats.com.

We did not use use any "physical" characteristic of fighters, such as the reach, the height or the age, only in-fight statistics. Therefore, this study should also answer the question: Can we predict the outcome of a fight with accuracy without using any fighter's physical characteristic ?[4]

1.2.1. Preprocessing. The following data preprocessing steps were required before being able to start training the models:

1. Handle missing or undefined values.
2. Remove non 3-round fights. Our model will therefore work only for predicting 3-round fights.
3. Remove female fights and fights in weight categories with very few fights.
4. Randomize the position of the winner, the UFC tends to put the fighter that is the favorite in the fighter 1 position.
5. Divide continuous features by the duration of the fight (in seconds) when required.
6. Replace each in-fight statistic by the average of this feature in the last k fights.
7. Choose the proper encoding for categorical variables.
8. (For training NNs) Standardize the continuous features.

At the end of this process we were left with only 2869 samples (fights) in our dataset, compared to more than 7000 at first. The reason for this massive loss is that we can only keep fights where both fighters have at least already 2 fights in the UFC.

1.2.2. Features. The following is a description of the different features.

- `weight_class` - The weight class of the fighters, ordinal-encoded (0 for Flyweight, 7 for Heavyweight).
- `knockdowns(A|B)` - Average number of knockdowns inflicted to opponent in the last k fights.
- `total_strikes_att(A|B)` - Average number of strikes attempted in the last k fights.

[1] The reason for such a low k is mainly because a lot of fighters get an initial contract for only 3 fights when joining the UFC. Taking $k > 2$ could yield more accurate models, but we would not have enough data to train them.

[2] We assume odds in the European/Decimal format. This is also why we need to subtract 1 in the EV formula.

[3] The name of the dataset is "MMA Dataset 2023 (UFC)".

[4] For example, the age of the fighters has been shown to be one of the most important features used by Gradient Boosting models when trying to predict the winner. [1]

- `total_strikes_succ(A|B)` - Average number of strikes attempted and succeeded (landed) in the last k fights.
- `sig_strikes_att(A|B)` - Average number of significative strikes attempted in the last k fights.
- `sig_strikes_succ(A|B)` - Average number of significative strikes attempted and succeeded in the last k fights.
- `takedown_att(A|B)` - Average number of takedowns attempted in the last k fights.
- `takedown_succ(A|B)` - Average number of takedowns attempted and succeeded in the last k fights.
- `submission_att(A|B)` - Average number of submissions attempted in the last k fights.
- `reversals(A|B)` - Average number of reversals[5] in the last k fights.
- `ctrl_time(A|B)` - Average control time[6] in the last k fights.
- `wins (dec|sub|ko)(A|B)` - Number of wins of the corresponding category in the last k fights (0 everywhere means that the fighter lost the last k fights).

1.2.3. Dependant Variables. There are 3 different outcomes that we can try to predict given our data - the winner of the fight - the way the fight will end - the round at which the fight will end.

We assume that draws are impossible. Therefore, after encoding them numerically, the first possible target yields to a binary classification problem, the two others to a 3-class[7] classification problem.

1.2.4. Training/Testing Split. Finally, we set the last 500 fights to be our testing set (X_{test}, Y_{test}), and the rest to be our training data.

2. Classification Models

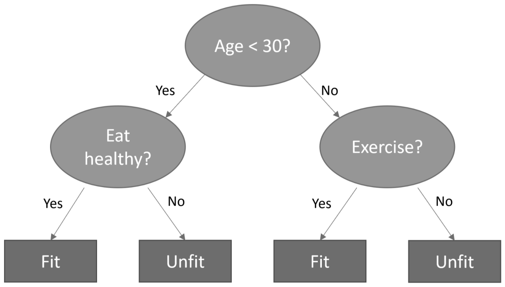
We chose to work with the three following types of machine learning algorithms: Decision Tree, Random Forest and Neural Network (Multilayer Perceptron). We chose these models for their propensity to "take risks", that we define by the larger variance in their output probabilities, compared to linear models[8]. Gradient Boosting also seemed promising and could have also been used, although it requires a lot of hyper-parameters tuning.

Our approach was to train the models in a way that offered a good compromise between accuracy (on unseen data) and their score in metrics that take into account probabilities, such as the Cross-Entropy-Loss or the Brier-Score[9].

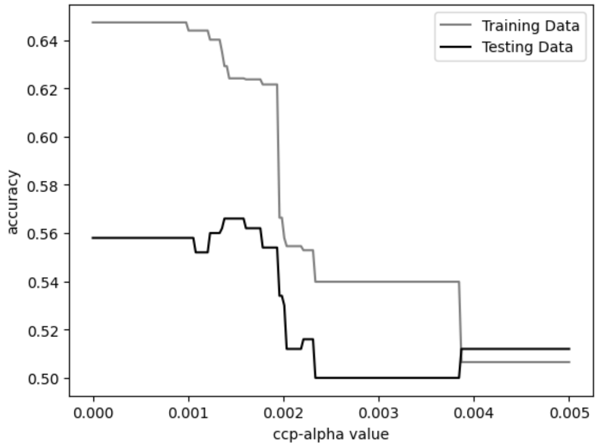
Since we want to try to predict three different dependant variables, we had to train three models per type of algorithm.

2.1. Decision Tree

This algorithm works by recursively partitioning the input space into smaller sub-spaces based on feature values, creating a tree where each internal node represents a feature test, each branch represents an outcome of the test, and each leaf node represents a class label.



For each tree, we had to identify the optimal ccp-α parameter. Finding the optimal value for this hyper-parameter helped us prevent over-fitting[10] and increase the accuracy of the model.



We also constrained the tree to have a depth of at most $\log_2(|X_{\text{train}}|)$ and a minimum number of samples per leaf to 25~30, otherwise, the model will likely over-fit by creating almost one leaf per sample.

2.2. Random Forest

Random Forest is a machine learning algorithm that belongs to the family of ensemble methods. It works by

[5] Reversal is a general term for going from a bottom position that is not a "guard" to a top position.

[6] Control time is the time spent by the fighter in the dominant position on the ground or in the clinch.

[7] The outcome of a fight is Decision, Submission or Knockout. And, as explained previously we train our models exclusively on 3-round fights.

[8] Such as Logistic Regression or Support Vector Machines, among others.

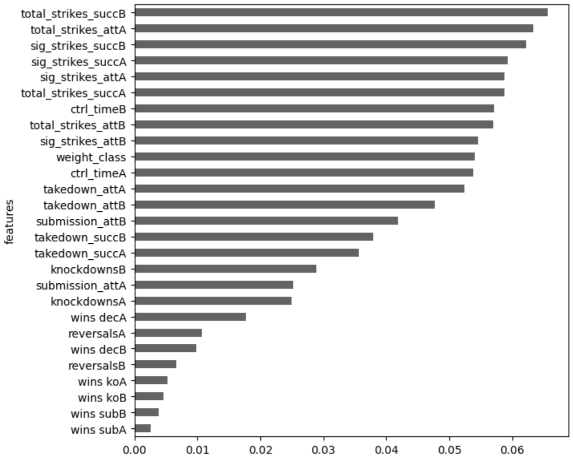
[9] These metrics are talked in details in section **3. Evaluation Metrics**.

[10] At first we had a perfect accuracy on training data, and almost 500 leaves.

creating multiple decision trees and merging them together to get a more accurate and stable prediction.

One advantage of using Random Forest over a single Decision Tree is that it is less sensible to change in its hyper-parameters. However, when running the scikit-learn `RandomForestClassifier` with default parameters, we were likely over-fitting on the training data[11]. As with the Decision Tree algorithms, we increased the minimum number of samples per leaves, set a maximum depth and a high number of estimators (number of trees used to make the prediction). As a result, our accuracy on unseen data increased and our Out-of-bag (OOB) error[12] decreased.

After fitting this algorithm, we can plot the importance of each feature, that is how much each feature contributes at reducing the uncertainty in the target variable. Doing this can give us an idea of what are the most important features in the dataset.



Features importance in the Random Forest model predicting the round. Interestingly, none of our 3 Random Forest models used much the different number of wins over the last k fights for its predictions.

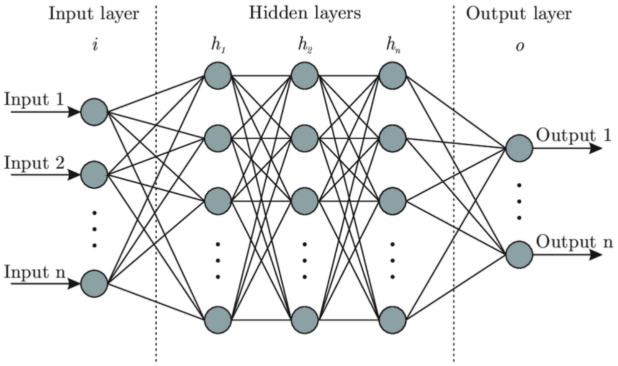
2.3. Neural Network (Multilayer Perceptron)

A Neural Network is a type of machine learning model consisting of layers of interconnected nodes, called neurons, that process information and make predictions based on that information.

More precisely, we use a Multilayer Perceptron, a kind of NN where each layer of neurons is fully connected to the next one, and the nodes in each layer use a non-linear activation function to transform the input signal.

The three models that we trained had in common the number of hidden layers, that is one, and the activation functions used, namely the Rectified Linear Unit $\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$ for the hidden layer, and Softmax for returning probabilities as our final output layer, defined as (with K classes)

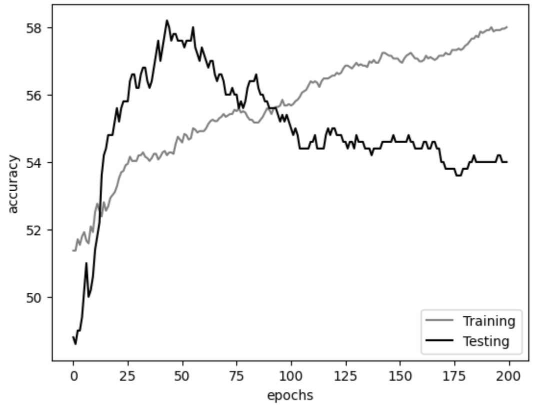
$$\text{Softmax}(\mathbf{x}) = \Big[\frac{e^{x_1}}{\sum_{j=1}^{K} e^{x_j}},\ \frac{e^{x_2}}{\sum_{j=1}^{K} e^{x_j}},\ \ldots,\ \frac{e^{x_n}}{\sum_{j=1}^{K} e^{x_j}}\Big]$$



In our case we have 27 inputs and either 2 or 3 outputs. For predicting the winner, that is a binary classification problem, we could have had a single output if we used the Sigmoid (also called Logistic) function $\frac{1}{1+e^{-x}}$ as our final layer activation function.

We also used the Adam[13] algorithm for optimizing the network weights [2].

Among the different hyper-parameters that are not common to our three Neural Network models, the number of neurons in the hidden layer is 20, 32 and 100 respectively for the model predicting the winner, the result and the round[14]. We also did not train them for the same number of epochs[15], but respectively 16, 7 and 2. The reason for such a low number of training iterations for the last two is that the networks, because of the small size of our dataset, have a tendency to over-fit very easily.



The graph of the training of the NN for predicting the winner, we can see that after ∼50 epochs the accuracy on testing data stops improving whereas the accuracy on training data keeps getting better, a sign that the model is over-fitting.

[11] For the same reasons as for the Decision Tree.

[12] An estimate of the generalization error of the Random Forest model. It is calculated using the predictions made on the samples not used in the bootstrapping for each tree in the forest.

[13] A popular stochastic gradient descent (SGD) method, usually more suited for bigger datasets, but that worked well in our case.

[14] We say that they have a different network architecture.

[15] Training for an epoch means showing the whole training data to the model (since it was previously divided in smaller batches).

Finally, the last parameter that we modified was the weight decay (or L2 regularization), a greater weight decay will discourage the weights from becoming too large, and thus help prevent the model from over-fitting.

3. Evaluation Metrics

In this section we explain the different metrics we used to assess the performance of our models.

First and foremost, we have the accuracy, simply defined as the number of correct predictions over the total number of predictions. A simple metric that allow us to - at least - see if we are going in the right direction when training. Another metric widely used in classification problems is the F1-Score defined as

$$\mathrm{F1} = \frac{2 \times \mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

$$\mathrm{Precision} = \frac{\mathrm{True\ Positives}}{\mathrm{True\ Positives} + \mathrm{False\ Positives}}$$

$$\mathrm{Recall} = \frac{\mathrm{True\ Positives}}{\mathrm{True\ Positives} + \mathrm{False\ Negatives}}$$

When the problem is not a binary classification, we use the Weighted-F1 that computes a weighted average of the F1 score for each class. The weights are determined by the number of instances in each class. This is particularly useful when dealing with imbalanced datasets, where some classes may have significantly more instances than others[16].

Since the goal of this is to build models that can be potentially used for betting, and thus to model probabilities, we are also interested in knowing the quality of the probabilities that our models output. We used two metrics, the Brier-Score, defined as, with N samples, R classes, Y the targets and $\hat{Y}$ our predictions (Softmax probabilities)

$$BS = \frac{1}{N}\sum_{i=1}^{N}\sum_{r=1}^{R}\left(\hat{Y}_{i,r} - \begin{cases} 1 \text{ if } Y_i = r \\ 0 \text{ otherwise} \end{cases}\right)^2$$

this metric is in a way similar to the Mean Squared Error, widely used for regression tasks. We also used the Cross-Entropy-Loss, defined as

$$CE = \frac{-1}{N}\sum_{i=1}^{N}\sum_{r=1}^{R}\log(\hat{Y}_{i,r})\begin{cases} 1 \text{ if } Y_i = r \\ 0 \text{ otherwise} \end{cases}$$

it is also the function we used as our loss function for training the Neural Networks.

Finally, we are also interested in knowing the propensity of the models to ”take risks” in their predictions, so we also looked at the variance and the maximums in their output probabilities.

4. Results and Analysis

The following tables summarize the results we obtained with the different algorithms - all metrics were computed on data unseen by the models. The hyper-parameters for each model were detailed in the previous sections.

Winner	**Decision Tree**	**Random Forest**	**Neural Network**
Accuracy (%)	52.0	54.6	**56.5**
F1-Score	0.589	**0.537**	0.581
Log-Loss	0.719	**0.688**	0.721
Brier-Score	0.520	**0.495**	0.513
Variance (×100)	0.822	0.463	2.010
Max (A, B)	0.64, 0.75	0.67, 0.71	0.99, 0.86

Result	**Decision Tree**	**Random Forest**	**Neural Network**
Accuracy (%)	**44.6**	43.6	44.0
F1-Score (weighted)	**0.375**	0.326	0.305
Log-Loss	1.062	**1.041**	1.057
Brier-Score	0.642	**0.632**	0.641
Variance (×100)	5.159	2.193	3.668
Max (Dec, Sub, Ko)	0.92, 0.60, 0.73	0.64, 0.34, 0.48	0.97, 0.23, 0.54

Round	**Decision Tree**	**Random Forest**	**Neural Network**
Accuracy (%)	52.0	**52.8**	51.6
F1-Score (weighted)	0.397	**0.371**	0.392
Log-Loss	1.032	**1.030**	1.075
Brier-Score	0.617	**0.614**	0.647
Variance (×100)	5.083	3.193	1.155
Max (1, 2, 3)	0.68, 0.50, 0.97	0.43, 0.27, 0.72	0.43, 0.34, 0.99

All three algorithms seem to give interesting results. Random Forest, in particular, always gives the best probabilities even if it is not always the most accurate[17]. It also has a decent variance.

The Neural Network models also give similar results, however they might be sometimes too confident in their predictions, as we can see in the Max row of each table.

Interestingly, the Decision Tree model, despite being the most accurate for predicting the result, has a Log-Loss much worse than the others. Actually, the Decision Tree algorithm always gets the worst results in metrics that take into account probabilities.

5. Betting / Conclusion

These models must now be back-tested (for e.g. using historical odds data), and it is the next step for this project. We hope that our models might be able to predict potential upsets.

References

[1] McQuaide, M., 2019, *Applying Machine Learning Algorithms to Predict UFC Fight Outcomes*

[2] Kingma, D.P. and Ba, J., 2014. *Adam: A method for stochastic optimization.* arXiv preprint arXiv:1412.6980.

[3] Turgut, M., 2021, *Machine Learning approach to predicting Mixed Martial Arts matches*

[16] In our case, there are much more fights that end at the 3rd round by decision that by knockout or submission for example.

[17] For Random Forest, we know that with more features and data an accuracy of 58.98% can be reached at predicting the winner. [3]