A Puncher's Chance: MMA Classification Predictor

Context

Mixed martial arts (MMA) is the ultimate stage of informed gambling - a sport where individual known quantities come head to head to decide a definitive outcome. As a massive combat sports fan, I saw MMA as the perfect sport to try my hand at predicting the future. Specifically, I aimed to develop a machine learning model that could accurately predict the outcome of UFC fights based on a variety of features. While machine learning for sports betting is hardly new, MMA provides a special abundance of diverse features from which to develop an algorithm.

To achieve my goal, I began by developing a model that exhibits accuracy in a classical sense - by picking the correct binary target choice. However, as I learned more about betting, I realized that the non-binary nature of predictive probability was more relevant. Ultimately, I pivoted my goal towards finding the most favorable bets by focusing on the linear probabilities of victory for each party and tweaking my probability thresholds to skew more generously towards the precision end of my algorithm's precision-recall curve.

I owe a lot of thanks to the wonderful data set compiled by Kaggle user mdabbert. This data set includes all public UFC data sets compiled into a glorious 4986 x 118 dataframe, providing a wealth of features to train my model. In this writeup, I will provide an overview of the data set and discuss how I used it to train my model. I will also discuss precision and recall, two important metrics for evaluating the performance of my model.

Precision and Recall

To evaluate the performance of my model, I prioritized precision and recall over other metrics like accuracy and F1 score. I decided to prioritize accuracy less because I was more interested in the predictive probabilities of a choice (the chances a model picks one choice or the other) rather than the result. In addition, using the F1 score (a metric that combines precision and recall into a single score) as a decision making metric created too little of a threshold difference to accurately tell whether my model was tending towards one side or the other.

Precision represents the proportion of correct positive predictions among all predictive positives, whereas recall represents the fraction of correct positive predictions among all true positives (in this

case, a "positive" is simply an arbitrary level of the target to be chosen as the value 1 - to see how I encoded my target to follow the positive/negative meta, please see below).

In sports betting, precision is crucial as it focuses on false positives, or the number of positive predictions that are actually true. A high precision score suggests a model's ability to correctly predict profitable bets. On the other hand, a high recall score is ideal for algorithms that strive to identify all positive examples, even if it means incorrectly identifying some negative examples as positives. This is useful in situations where failure is not an option, such as medical diagnosis or fraud detection. However, in a betting context, a model tending towards recall can be especially volatile since a false negative can lower the overall expected value. When targeting my algorithm towards a higher recall value range, I found myself betting a lot of underdogs.

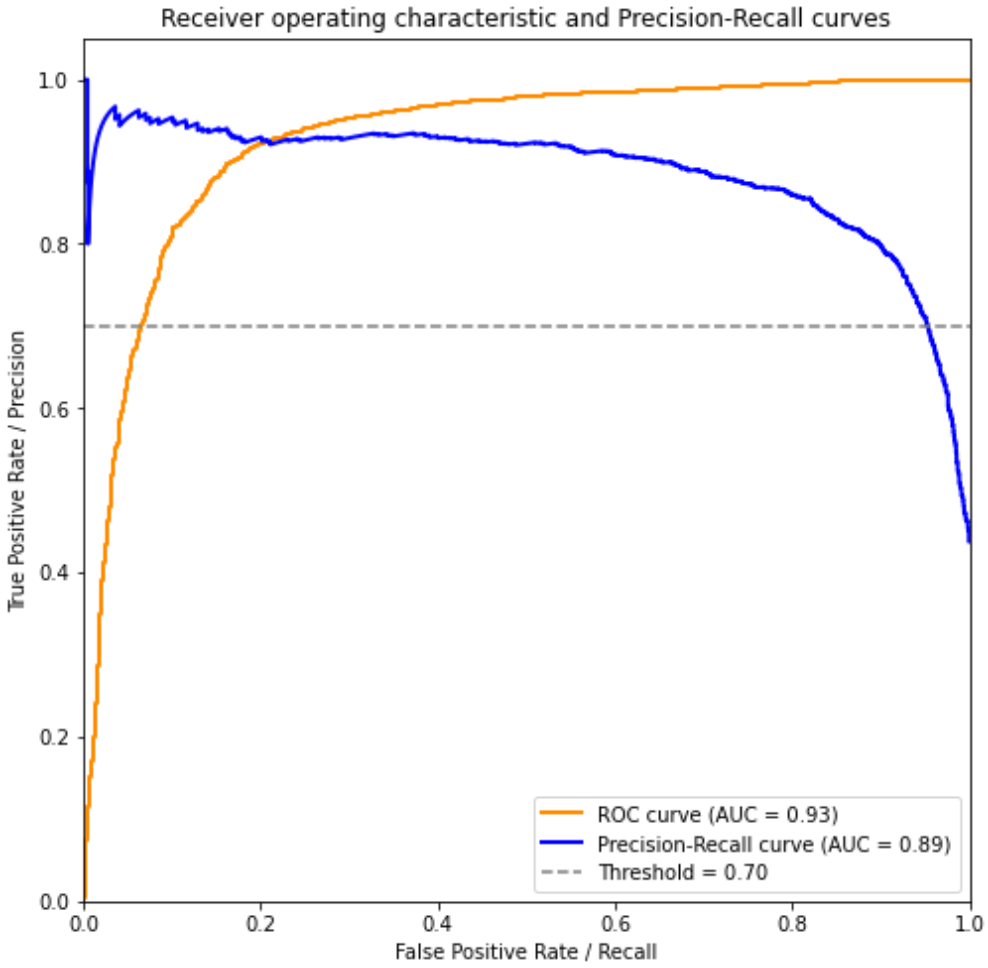
Rationale behind Feature Selection

The dataset contains a lot of nulls, incongruencies, and noise due to its origins from multiple sources. To address this, I removed most columns with more than half-nulls. However, I also realized that the ranking variables were problematic, as they showed no clear correlation with the target.

It's worth noting that one-hot encoding can help with classification-based models, but in some cases it can make linear models unreliable by creating multiple groups that need to be monitored and muddying an otherwise linear-tending correlation.

After experimenting with different variables, I found that including odds and expected value was crucial for creating a reliable predictive algorithm. Odds and expected value are mathematically tied to the probable rate of return on a bet, making them essential variables for identifying profitable bets.

Below is the combination of precision-recall and ROC curves that I settled on. As can be seen, with the threshold I selected (0.7), the plot splits into a high false-positive region with more activity than its counterpart below. This is exactly what I want - since the model is picking probabilities based on the threshold value (the model changes both regions on either side of the threshold to areas with equal probability of being selected), I'm eliminating ambiguity on the false-positive side which should ensure more profitable bets across the board. This is because my model tends to see the more ambiguous false-positive cases as positive as well, sequestering only the region dedicated to guessing a false-positive correctly.



Preprocessing and Feature Engineering

The main goals of my preprocessing and feature engineering flow were to increase the linearity of my features and reduce extra complexity in the data. Given the diverse information in the dataset, especially with numerical features on different scales, I aimed to curate a set of features that showed no collinearity with each other, had no crazy outliers, and exhibited a (relatively) strong correlation with the target.
To achieve this, I first calculated the square of the Pearson correlation coefficients for each feature, selecting only those whose squared coefficient exceeded a threshold of 0.05. Squaring the coefficients allowed me to evaluate both positive and negative correlations at the same time, and to focus on the magnitude of the correlation.

Before I did anything, I performed some basic imputation to get rid of nulls in my data, along with some rudimentary encoding, and ran a baseline model using a dummy classifier that only predicts one value. My baseline accuracy was about 0.54.

Additionally, I reduced complexity by creating difference columns wherever possible, subtracting one fighter's value from the other to eliminate individual columns for each fighter. For example, I created a "win difference" column by subtracting the red fighter's win column from that of the blue fighter. I dropped individual columns that had already been engineered this way and eliminated high-null columns to get a clearer view of which columns exhibited a stronger linear correlation with the target. For more detailed information and code, please refer to the final_process.ipynb file in my project repository on GitHub.

Model Selection and Hyperparameter Tuning

After trying multiple linear models, four of them stood out in terms of their performance. The first model, Gaussian Naive-Bayes, and the second model, Adaboost Classifier, achieved a perfect accuracy score of 1.0. This perfect score indicates that these models may be overly sensitive to the training data and prone to overfitting when exposed to new data.

In contrast, the third model, Logistic Regression, yielded a more reasonable accuracy score of 0.66. Logistic Regression is a type of regression analysis used for predicting binary outcomes by fitting the data to a logistic function. It is widely used in various fields due to its simplicity and interpretability. The fourth model, Support Vector Machines (SVM) Classifier, returned an accuracy score of 0.64. SVM is a powerful classification algorithm that aims to find the best hyperplane to separate data points of different classes. It works by maximizing the margin between the decision boundary and the closest points from each class.

Considering the performance and the desire to strike a balance between accuracy and avoiding overfitting, the focus was narrowed down to Logistic Regression and SVM Classifier as the most promising models for further analysis and evaluation.

I settled on logistic regression because other metrics were more or less similar but my logistic regression model's ROC had a steeper rise suggesting an affinity towards precision.

In the process of hyperparameter tuning, a simple parameter grid was established to optimize the performance of the model. Three specific hyperparameters were selected for tuning. Firstly, the penalty hyperparameter was set to "l2." This hyperparameter determines the type of regularization applied to the model's weights during training. The "l2" penalty, also known as Ridge regularization, adds the squared magnitude of the weights to the loss function, effectively constraining them and preventing overfitting.

Secondly, the max iterations hyperparameter was set to 135. This hyperparameter determines the maximum number of iterations allowed for the model to converge during training. A higher value allows for more optimization steps, potentially improving the model's performance. Lastly, the C

hyperparameter was set to 0.01. C represents the inverse of the regularization strength, and a smaller value like 0.01 imposes stronger regularization, discouraging the model from fitting the training data too closely. By selecting these specific hyperparameters and fine-tuning their values, the goal was to enhance the model's generalization ability and strike a balance between bias and variance.

Model Evaluation

The evaluation of my classification model revealed promising results compared to the baseline model. My new model achieved an accuracy of approximately 0.84, demonstrating an improvement of nearly 30% over the baseline model's accuracy of around 0.54. This significant enhancement highlights the effectiveness of the approach. The ROC curve for my model exhibited a favorable position, tending towards the top left of the plot. This positioning suggests a preference for precision, which aligns with the objective of the model. Furthermore, the precision-recall curve displayed a steeper slope, indicating an increased focus on precision. These findings validate the model's ability to make predictions with a higher level of confidence. The model also correctly predicted the outcomes of UFC Fight Night: Thiago Santos vs Johnny Walker, a test dataset obtained from the same source as the main training data on Kaggle.

Analysis and Final Thoughts

In the context of sports betting, I am satisfied with the performance of my model, although I acknowledge its limitations in professional settings. The primary aim of the model was not only to predict fight outcomes but also to identify results that leaned towards more favorable betting opportunities. Achieving an accuracy improvement of approximately 30% demonstrates the model's potential in assisting with sports betting predictions. To further enhance my model, there are several areas for future improvement. Firstly, I would consider exploring more advanced calculations that relate ranking variables to reduce complexity while retaining valuable information. Although I had to exclude some one-hot encoded variables due to their impact on data linearity, I believe incorporating a neural network or a tree-based model could harness the power of these variables effectively. Additionally, I would work on establishing a more indicative baseline by conducting thorough research and data visualization to gain insights into the structure of the numerical data. Choosing an appropriate baseline model that better aligns with the data characteristics would provide a more reliable reference point for evaluating the model's performance.