Counter Strike solution explanation

Bartosz Lewandowski

January 2021

Contents

1 Introduction

This is my solution for **esportsLABgg Counter-Strike Data Challenge**.

2 Problem with Data

The dataset that was provided for the competition is a typical dataset that is unbalanced. There was a little problem I had to face of.

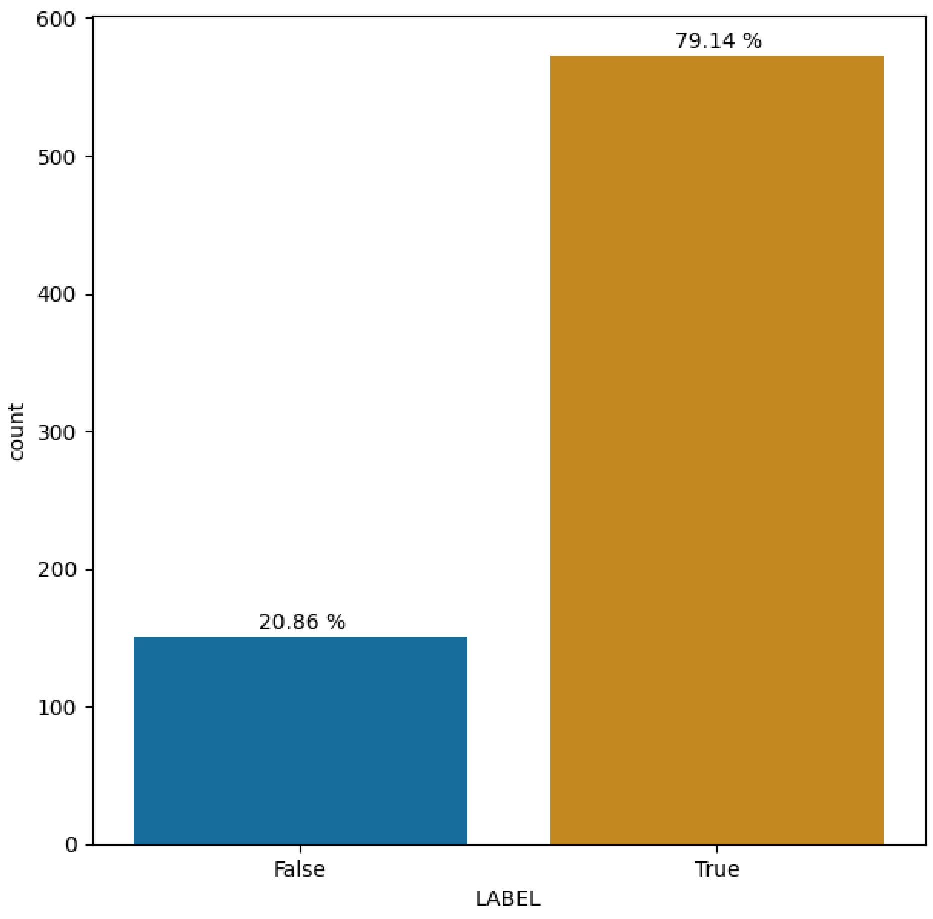


Figure 1: Proportion of classes in the dataset

My first two thoughts were two solutions: **oversampling** and **weights**. I tried this two solutions and compared them. As my evaluation metric I chose "roc_auc" score, because accuracy is always close to 80%, and it is not a valid metric in this case. I thought about f1 metric as well, but models often liked to increase recall, but precision was on really low level. In that case f1 score was high and I could treat that as good model, but the truth was this model had accuracy lower than randomly guessing. The roc_auc metric is the one that best represents the quality of the model.

3 Features selected

Dataset contained a lot of features, but not all variables contained valuable information. On the first place I wanted to give my model as much information as possible. I've made some feature engineering and created new features - duration time of thrown grenade in seconds, and change tick to time in seconds. I have done a feature importance with ExtraTreesClassifier and visualize dataset with PCA. This were the results:

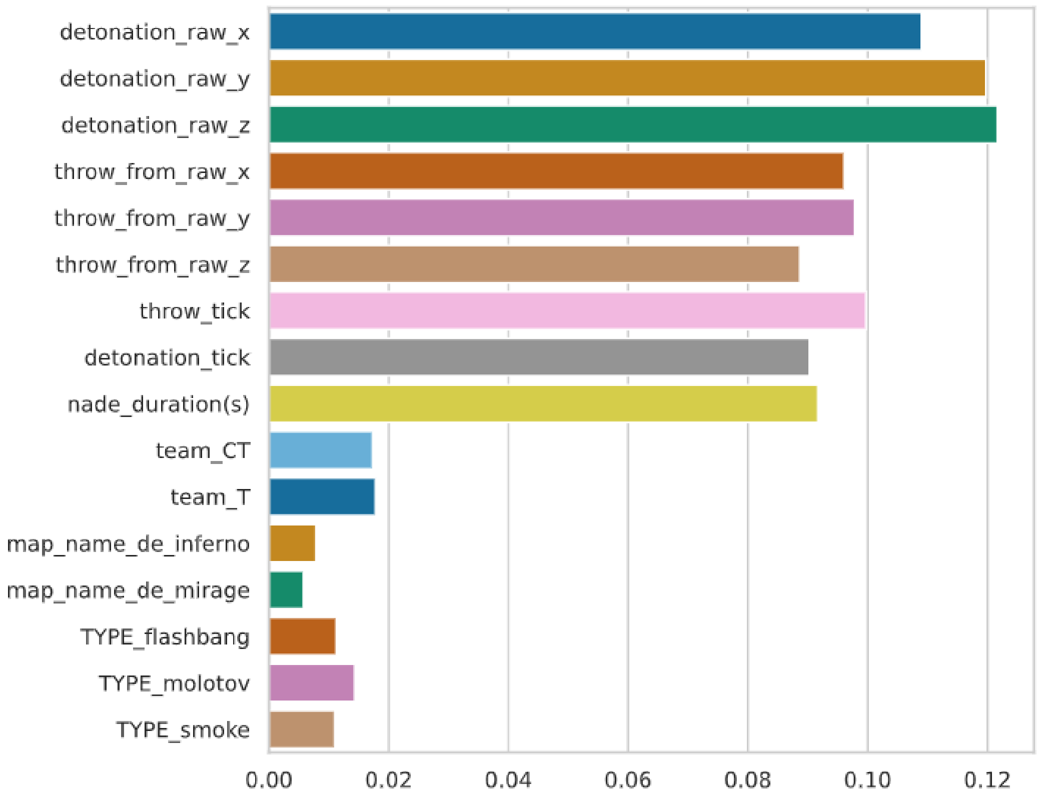


Figure 2: Features importance

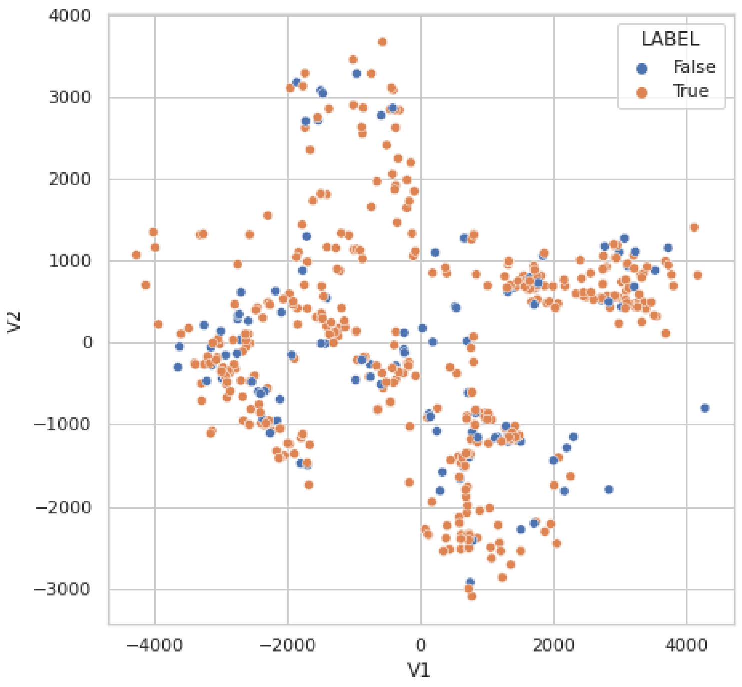


Figure 3: Dataset after PCA

I could see that for machine learning model it will be hard to separate this "points". It will be hard to classify grenades with this features. So I asked myself a question: "When grenade is thrown correctly?" The answer was simple - when it lands where it should be and at the exact time. So the most valuable information is the one with coordinates where the grenade have detonated and if it was at good time (e.g. when team is entering one site and player have to flash at exact time for the other one to enter). Ok, but what about categorical features? I checked them and as it turns out there is no difference between the maps and teams when it comes to the percentage of correctly thrown grenades (or at least the difference is negligible).

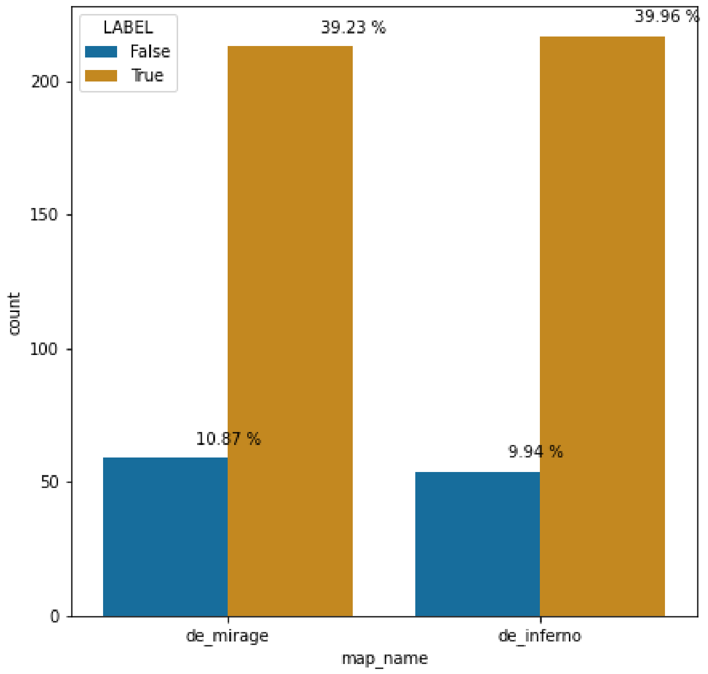


Figure 4: Thrown grenades on de_inferno and de_mirage

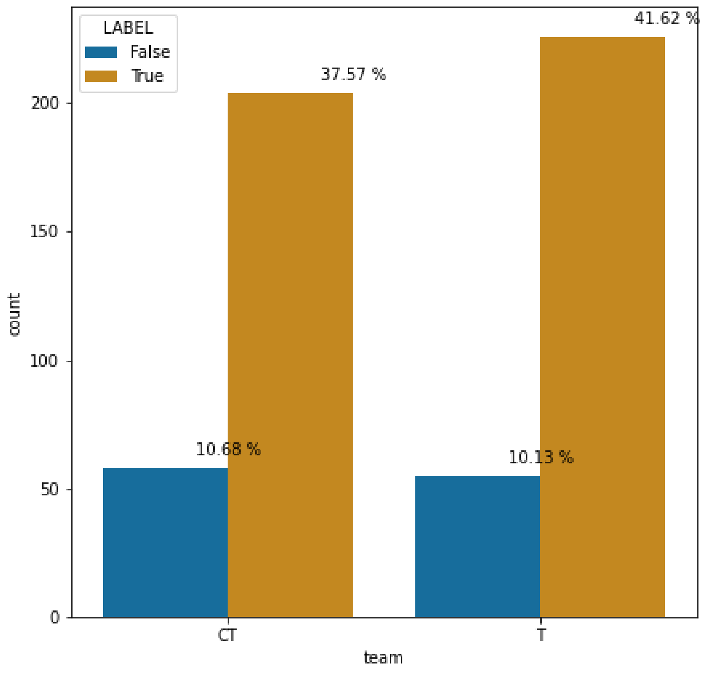


Figure 5: Thrown grenades in CT and TT

With the type of thrown grenade it was slightly difference between proportions of correctly to incorrectly thrown grenades.

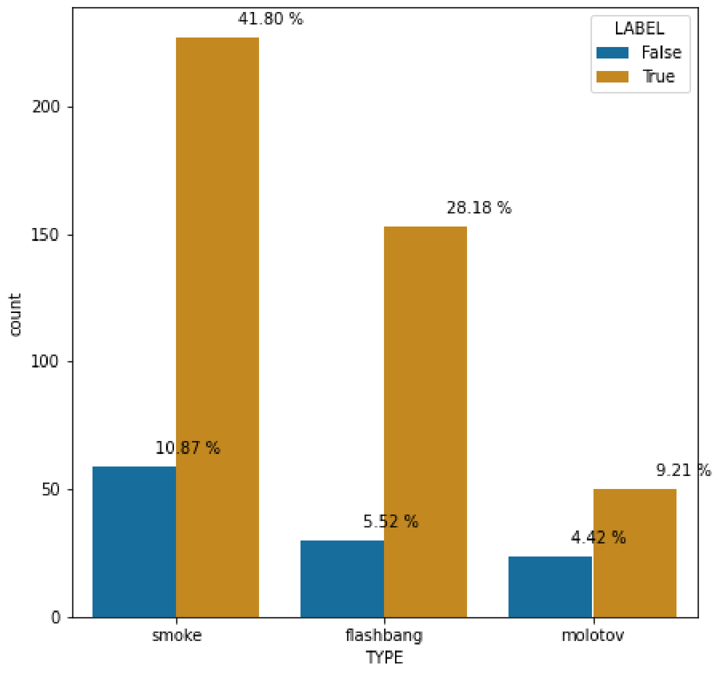


Figure 6: Types of thrown grenades

Ok so at this point I wanted to use variables like: coordinates, time and type of thrown grenade. I created earlier new features with time and I wanted to use them as well. Then I stopped myself, and rethought that. The round lasts 115 seconds, but as I know from experience, there can be a bomb planted at the last second. After the bomb is placed, players continue to throw grenades. I treat any round that lasts longer than 115 seconds as a round with a break. So it just confuses my model. After all, it came out that these variables reduce the accuracy of the model anyway. I also added information about what map it is, because this information increased model performance.

So at the end I picked only information about grenades detonation, type of grenade and on which map it was thrown. After that I performed PCA again and now I think we can separate correctly from incorrectly thrown grenades easier (but it is still a dataset that is hard to classify).

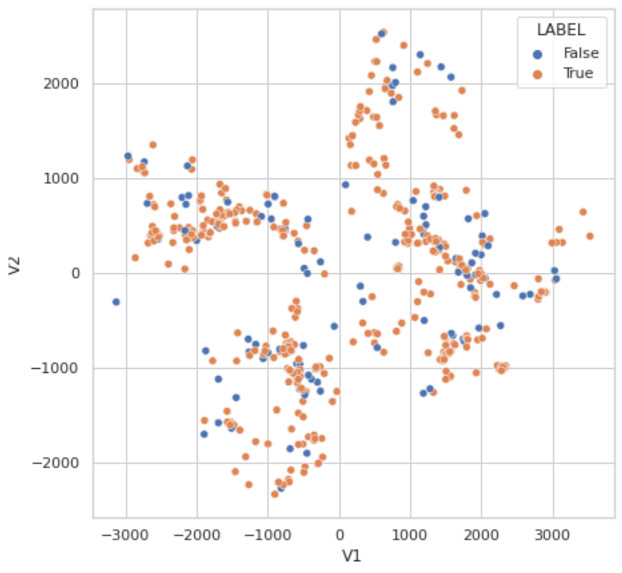


Figure 7: Features selected after PCA

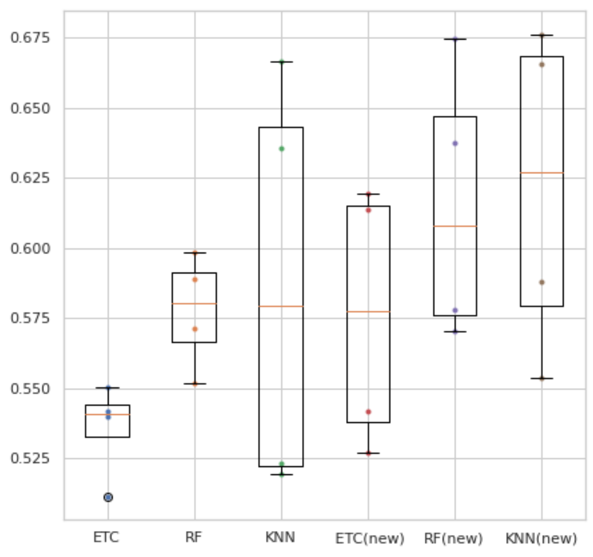
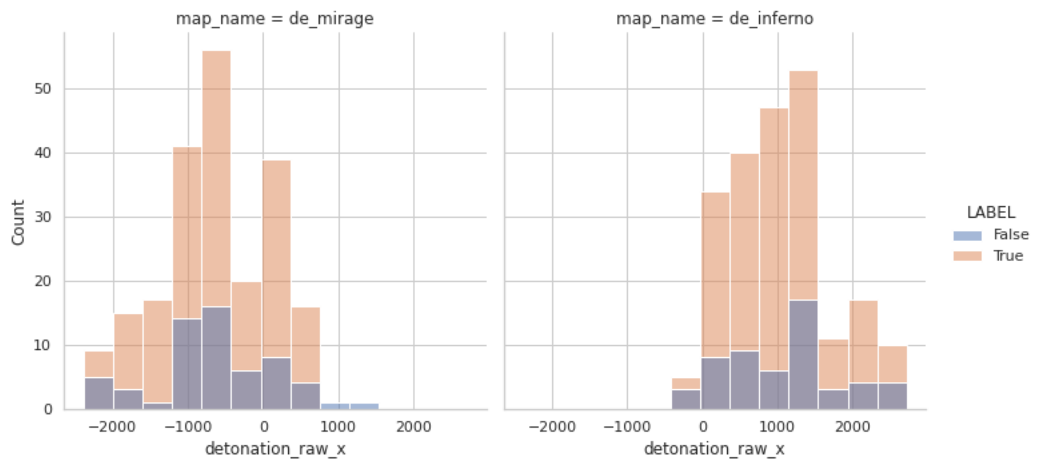
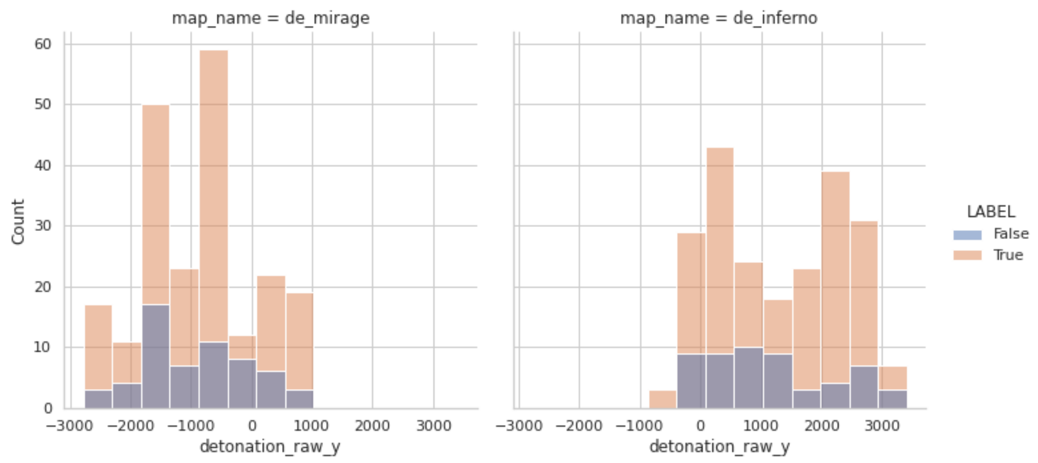


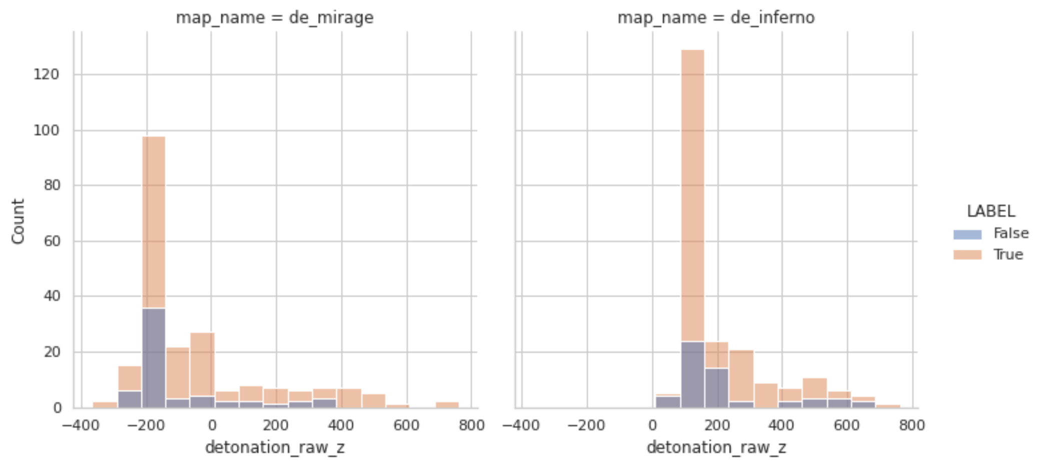
Figure 8: ROC AUC scores with different algorithms and features. On the right side models with fewer features.

4 Grenade detonation coordinates

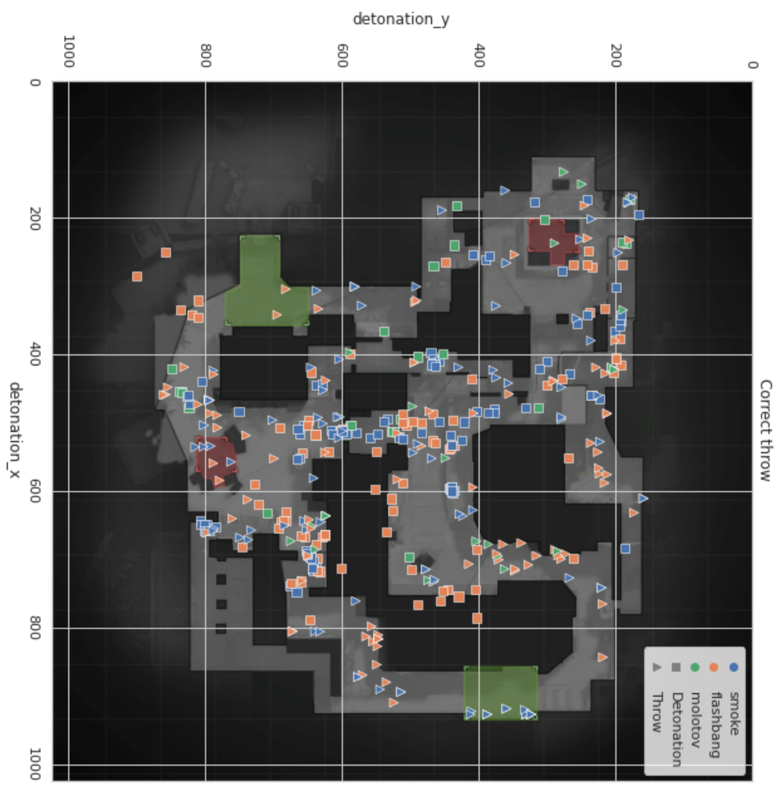
I looked at the coordinates a little more, since they were my most important variables. I first looked at their distribution and find out they overlap a little bit.







At this point I knew that it really won't be easy to classify this grenades.
I created a function to plot those grenades on map image (only x and y coordinates).



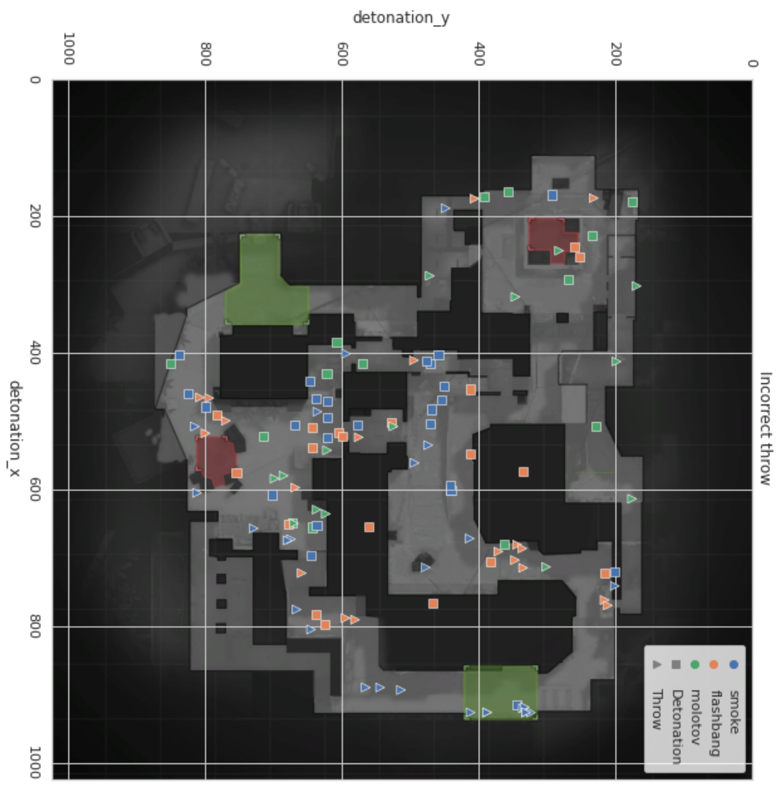


Figure 9: Map de_mirage with spots where grenade was thrown and where detonated.

I created this for de_inferno map as well. Also I created a function to plot just one grenade and information about it. This allowed me to take a closer look at grenades that were incorrectly classified.

5 Algorithms with weights

The dataset is small so I tried a lot of different ML algorithms. I have not tried neural networks, because I didn't want my model to overfit and I think neural networks are just too complicated for this problem (I guess they wouldn't got a better score).
So I have tried algorithms like:

- SVM(kernel poly and rbf)
- RandomForest
- XGBoost
- RandomForest
- K-nearest neighbors
- ExtraTreesClassifier
- LogisitcRegression

I've done hyper param tuning with grid search and these are the best five models I've got.

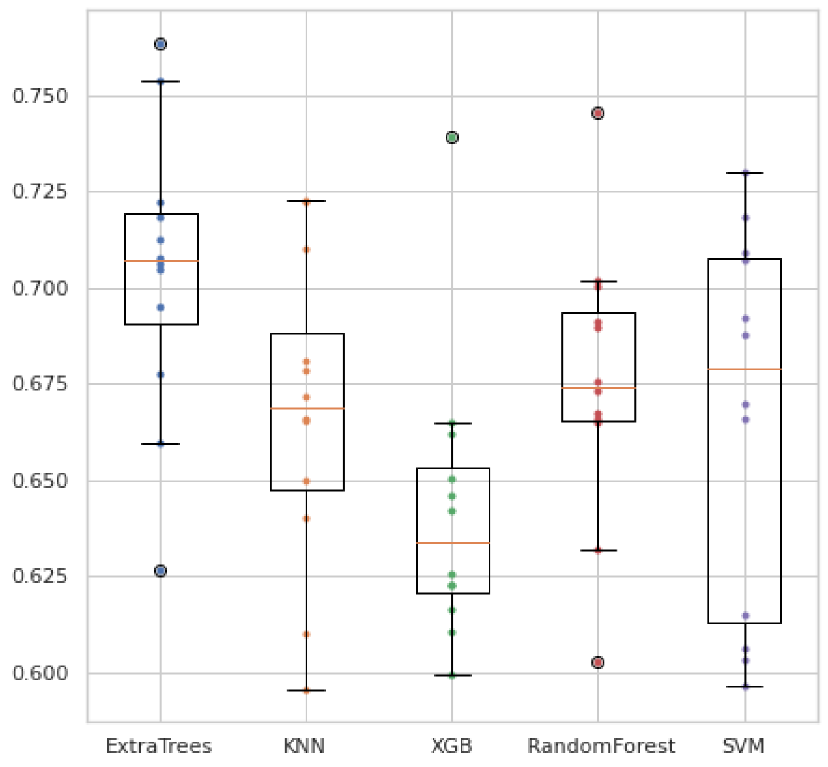


Figure 10: Best models with weights

6 SMOTE (oversampling)

Next I have tried those algorithms with SMOTE, which is oversampling method. I've done grid search again, but this time I've searched for best **k** and **sampling strategy** for SMOTE. K stands for how many neighbours algorithm have to consider to create new samples and sampling strategy define how much samples it have to create comparing to major label. This is what I've got:

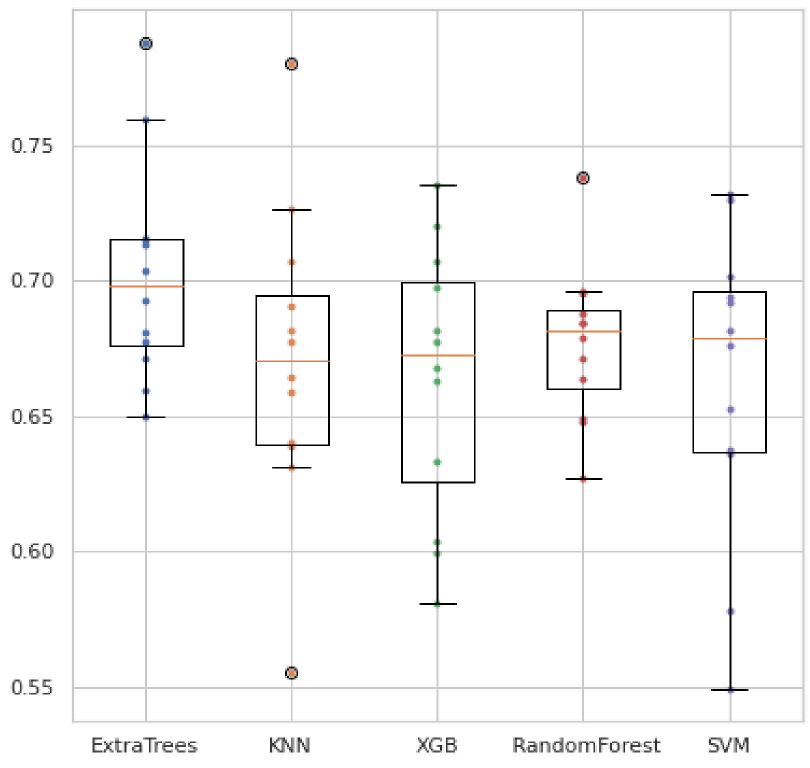


Figure 11: Best models with oversampling

7 Conclusions

I could not get higher roc_auc score than ExtraTreesClassifier.
Also I have tried AdaBoost, Bagging, Stacking and VotingClassfier, but it couldn't get higher performance than ExtraTrees as well. Summarize, I tried various methods with different hyperparameters, but none obtained higher values than ExtraTrees. I use this ML algorithm as my final to predict test dataset. When I wanted to increase the recall it caused a strong decrease in precision and vice versa. This is the type of case which is hard to classify. By collecting more data I could try to implement neural networks, and the model could get better performance. Another solution is to extract new information from matches, such as whether a grenade exploded next to a player.