Who are the high-frequency traders?

Guanyu Zhang, Hanady Gebran
https://github.com/JBGYZ/MAP569-datachallenge

***Abstract*—The purpose of this project is to classify traders in 3 categories: high-frequency traders, traders carrying out hybrid activity and non-high-frequency traders.**

I. Introduction

The scope of this project lies in supervised learning, in particular the classification of traders. Indeed we start from a dataset with an already established categorization, it is therefore more efficient to use this information to sort the traders. Nonetheless, we opted to begin with an unsupervised approach in order to see if the choice of three labels was justified, through the use of the k_means and DBSCAN algorithm. We then applied a number of standard classification algorithms found in the scikit library.

This report proceeds as follows. In the following section we will look at the theoretical aspects behind k_means, DBSCAN and the state of the art classification algorithm which are easily applyable through the scikit library. The third section focuses on the feature selection, the fourth section highlights the different performance of the various methods.

II. Background and Related Work

For our research project, we are interested in the following algorithms.

A. k-means clustering

When given a set of observations, k-means clustering is designed to partition the n observations into $k \leq n$ clusters such that the sum of the squared distances within a cluster is minimized, which is equivalent to minimizing the pairwise squared distances of the points in the same cluster. It proceeds as follows, given an initial set of k means, the algorithm alternates between two steps: assigning step: Allocate each observation to the cluster with the closest mean, i.e. the one with the smallest Euclidean distance squared, updating step: Recompute the means (centroids) of the observations in each cluster

B. DBSCAN clustering

DBSCAN is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points with many nearby neighbors, marking as outliers points whose nearest neighbors are too far away. It has 3 steps: First, find the points in the neighborhood of every point, and identify the core points with more than min pts neighbors. Second, find the connected components of core points on the neighbor graph, ignoring all non-core points. Those connected components are clusters. Third, assign each non-core point to a nearby cluster if the cluster is a neighbor, otherwise assign it to noise.

C. Random Forest Classifier

Random forests are similar to bringing together the efforts of decision tree algorithms. The teamwork of several trees improves the performance of a single random tree. The random forest training algorithm applies a general bootstrap aggregation. When given a training set x with responses Y, bootstrapping repeatedly selects a random sample with replacement from the training set and adapts the trees to these samples. After training, predictions for unseen samples can be made by taking the majority vote.

D. AdaBoost Classifier

AdaBoost is a particular method of training a boosted classifier. A boosted classifier is a classifier in the form of a summation of weak learners taking an input and returning a value indicating the class of the input. The AdaBoost Algorithm begins by training a decision tree in which each observation is assigned an equal weight. After evaluating the first tree, it increase the weights of those observations that are difficult to classify and lower the weights for those that are easy to classify. The second tree is therefore trained on this weighted data.

E. Gradient boosting Classifier

As with AdaBoost, it is a boosted classifier. The main difference between AdaBoost and the gradient boosting algorithm lies in the way the two algorithms approach the weak learners' shortcomings. The gradient boosting algorithm, instead of playing with the data weights, uses the gradients of the loss function to measure how well the model coefficients fit the underlying data.

F. Neural Net Classifier

This model optimizes the logarithmic loss function through a stochastic gradient descent, whereby the training is iterative: at each time step, the partial derivatives of the loss function with respect to the model parameters are calculated to update the parameters.

III. The data: feature selection

We represented the type of traders through 2 binary variables (MIX=11, HFT=01, NON_HFT=10). This made the correlations between the type of traders (categorical variable) and the other data possible.

Subsequently, the ability of the gradient boosting algorithm to rank the dimensions in order of importance was used and this algorithm was run with a range of features from 1 to the

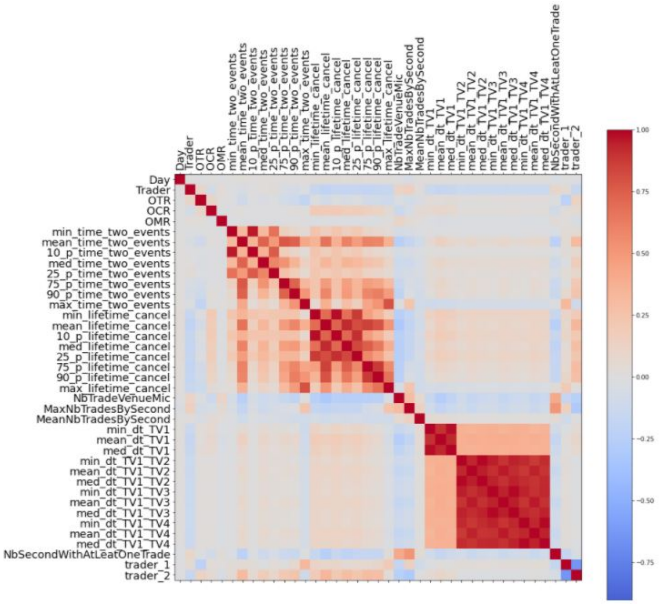

Fig. 1: Correlation matrix

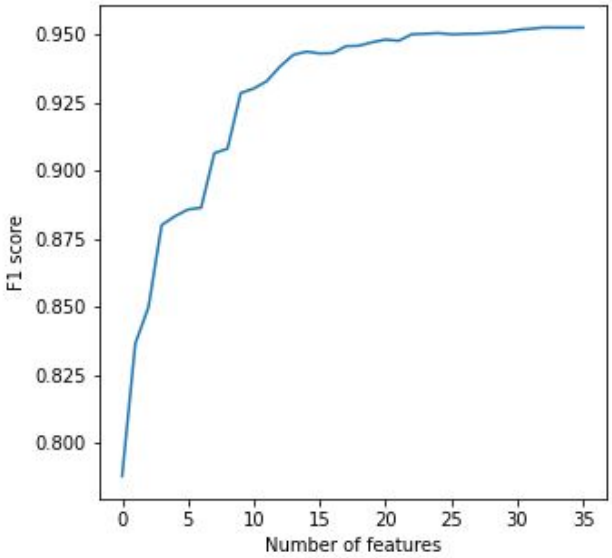

Fig. 2: f1 score obtained depending on the number of features

total number of features to see the impact on the f1_score of the number of features taken into consideration.

The most significant feature is the maximum time between 2 events which is logical because a HTF trader will tend to be faster. We can also see that with 7 features we already have an f1 score above 0.9.

IV. Results and Discussion

A. K_means

Doing the k_means algorithm was helpful as it showed with the elbow method that taking 3 clusters made sense. Nevertheless the accuracy was very disappointing compared to classification.

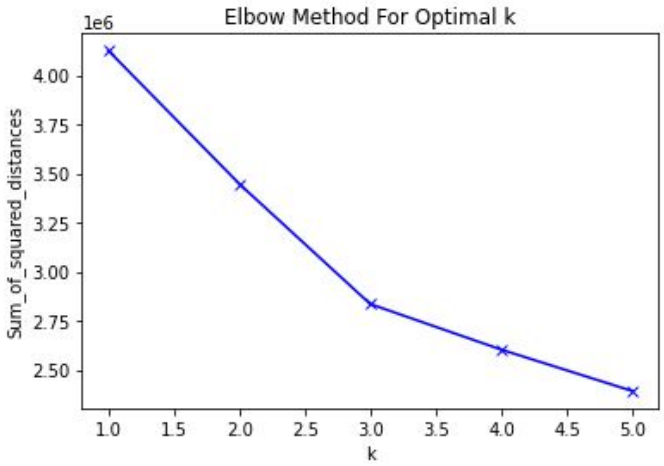

Fig. 3: Elbow method

B. DBSCAN

The number of cluster chosen by DBSCAN is reasonable. With our hyperparameter setting: epi=1 and min_samples=500, we get 5 clusters and a silhouette score of -0.21195 which is not terrible. But most of points are clustered into 3 groups which corresponds well to the real labels.

C. Random Forest Classifier

We tried several values for the hyperparameters, for max_depth=20, as long as we use between 50 and 140 estimators, we get an f1_score= 0.939 and with more than 150 estimators we have f1_score= 0.951. For n_estimators=150, the f1_score is very sensitive to the depth, for example for max_depth=5 one has f1_score=0.735 while for max_depth=10 we obtain f1_score= 0.89, the maximum (0.951) is reached with approximately max_depth=20. The main limitation was that the algorithm was slow.

D. AdaBoost Classifier

Among the Decision Tree based algorithms used, AdaBoost had disappointing results, as for a number of estimators between 100 and 1000, the f1_score is between 0.86 and 0.88. This is probably due to the large size of the space, as AdaBoost is less efficient in complex environment.

E. Gradient boosting Classifier

This was the most successful approach in our study. With the parameters n_estimators = 200, max_depth=10 we can achieve f1_score= 0.976 which is unequaled by any of the alternative methods. The results remain quite successful even with reduced number of estimators and depth. For instance, with n_estimators = 100 and max_depth=5 we obtained f1_score= 0.927. (Note that we used eXtreme Gradient Boosting).

F. Neural Net Classifier

The f1_score is rapidly limited to 0.86 obtained after 500 iteration, then after 1000 iteration the score decrease due to over-fitting. Moreover, the method converges quite quickly, after 50 iterations we already have f1_score=0.807.

V. Conclusion and Future Work

To conclude, the amount of data given in this study was adequate to achieve a valid classification. Besides the data was in a high dimensional space and removing the 3/4 of the dimensions had little influence on the obtained performances. In the end, xgboost was the best performing method, actually this classification algorithm is currently one of the most powerful state-of-the-art. Therefore, it came as no surprise to us that this algorithm was selected for our submission.

A little disclaimer: We made the choice of not giving a biblography, as the explanations given in the second paragraph were very general, and the algorithms used were based on examples from the scikit learn site or from very common libraries. Besides, the figures are exclusively from our experiments.

VI. Appendix

You can find above the results of feature importance obtained by xgboost.

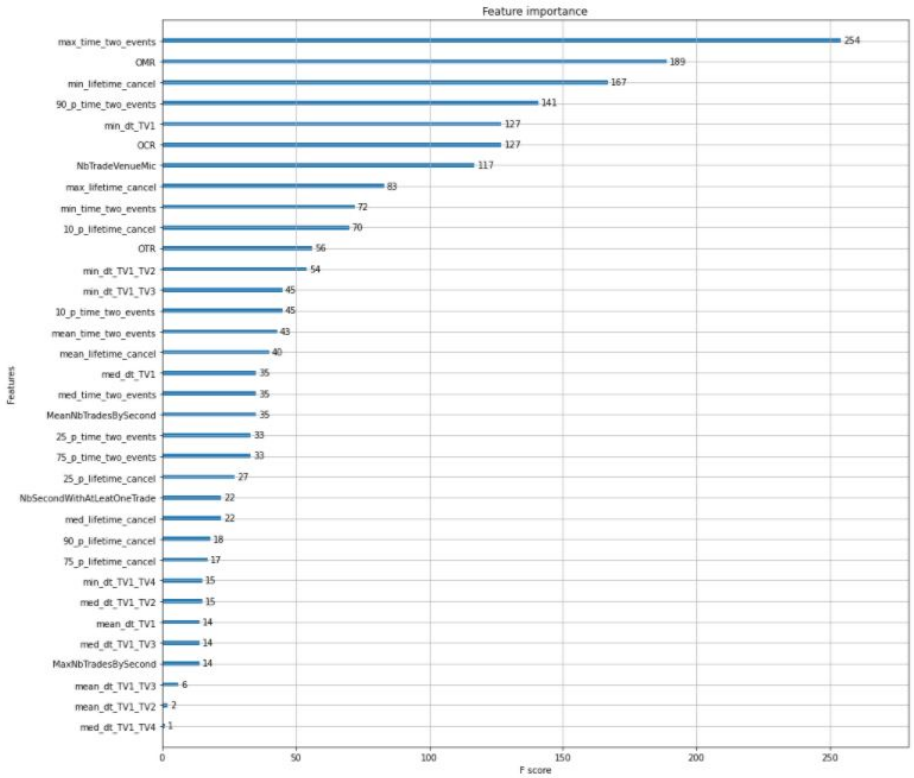

Fig. 4: Appendix: Feature importance