GROUP WORK PROJECT # 3
GROUP NUMBER: 4735

MScFE 632: Machine Learning in Finance

FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
Marin Stoyanov	Bulgaria	azonealerts@gmx.com	
Guoying Li	United States	liguoying1019@gmail.com	
Ashutosh Kumar	India	skdubey.ashutosh@gmail.com	

Statement of integrity: By typing the names of all group members in the text boxes below, you confirm that the assignment submitted is original work produced by the group (excluding any non-contributing members identified with an "X" above).	
Team member 1	Marin Stoyanov
Team member 2	**Guoying Li**
Team member 3	**Ashutosh Kumar**

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed. **Note:** You may be required to provide proof of your outreach to non-contributing members upon request.
N/A

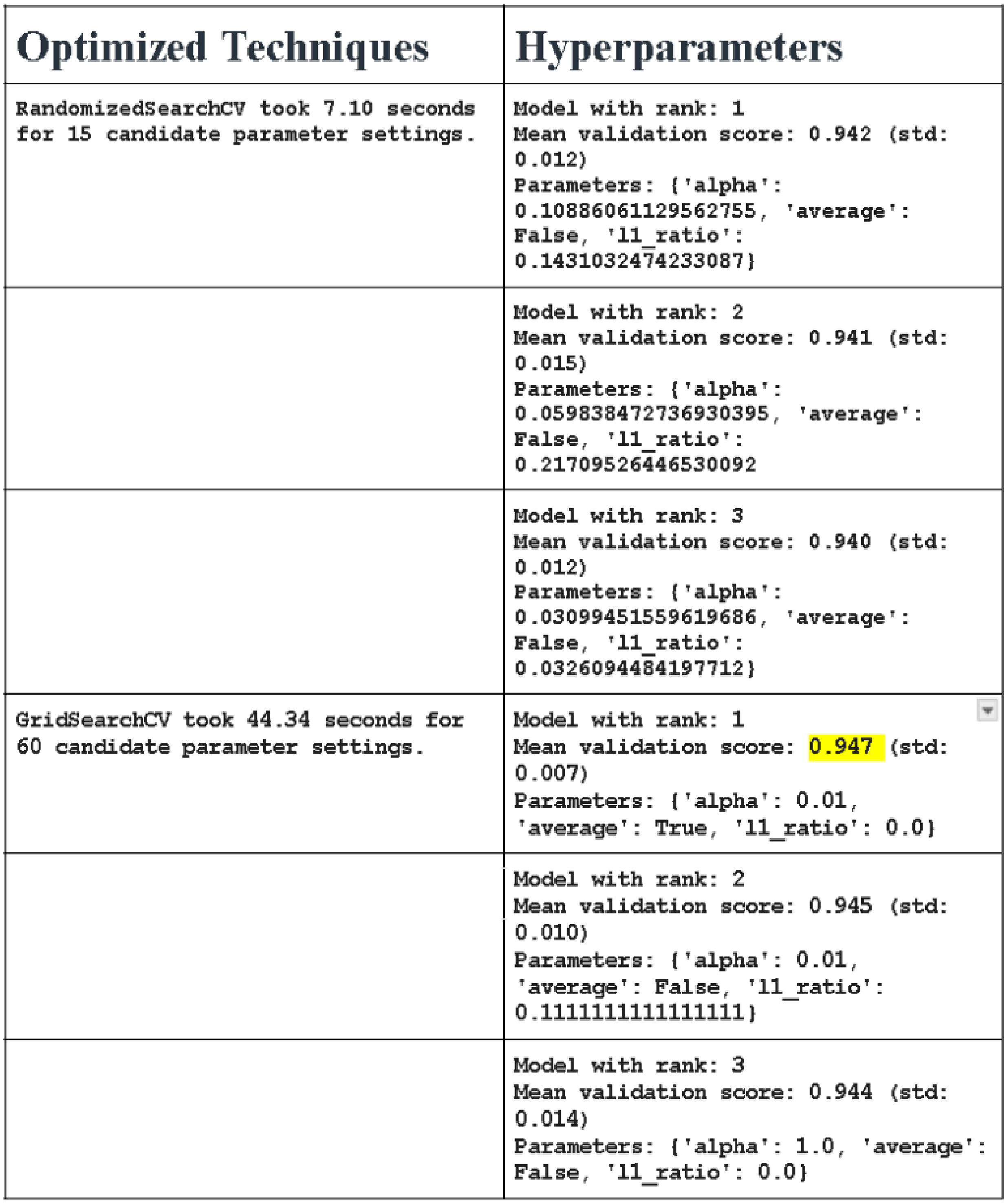
Optimizing Hyperparameters

Technical

In this project, we compare random search with grid search and see which optimization techniques perform better than the other in hyperparameter tuning. We first load digits dataset, apply SGD Classifier, split the data into training and test data sets, and find the top three best models for random search and grid search, based on mean validation score. Both optimization techniques have a good mean validation score at 94%. Grid search performs slightly better than random search and with less standard deviation than random search, while random search took less time than grid search.

With grid search, we found that precision, recall and f1 score on average at 96%. If we dive into the details and can observe that number zero and four has the highest F1 Score and number eight has the lowest F1 Score.

In table 1 we are showing a comparison between Random search and grid search. They are used to do the same job (hyperparameter optimization), but the difference is in the speed and in the approach. While grid search is doing a brute force method and going through all the combinations in the available (predefined) hyperparameter space, random search does not exhaustively try all parameter combinations. Instead, it selects random combinations of parameters to train the model and uses cross-validation to evaluate the performance of the model. This is why it is the more practical and efficient method for tuning hyperparameters. This is the preferred method for improving model performance by finding a good set of hyperparameters, especially for cases with a great number of hyperparameters or when it's computationally expensive to try all combinations of parameters.



Optimized Techniques	Hyperparameters
RandomizedSearchCV took 7.10 seconds for 15 candidate parameter settings.	Model with rank: 1 Mean validation score: 0.942 (std: 0.012) Parameters: {'alpha': 0.10886061129562755, 'average': False, 'l1_ratio': 0.1431032474233087}
	Model with rank: 2 Mean validation score: 0.941 (std: 0.015) Parameters: {'alpha': 0.059838472736930395, 'average': False, 'l1_ratio': 0.21709526446530092
	Model with rank: 3 Mean validation score: 0.940 (std: 0.012) Parameters: {'alpha': 0.03099451559619686, 'average': False, 'l1_ratio': 0.0326094484197712}
GridSearchCV took 44.34 seconds for 60 candidate parameter settings.	Model with rank: 1 Mean validation score: 0.947 (std: 0.007) Parameters: {'alpha': 0.01, 'average': True, 'l1_ratio': 0.0}
	Model with rank: 2 Mean validation score: 0.945 (std: 0.010) Parameters: {'alpha': 0.01, 'average': False, 'l1_ratio': 0.1111111111111111}
	Model with rank: 3 Mean validation score: 0.944 (std: 0.014) Parameters: {'alpha': 1.0, 'average': False, 'l1_ratio': 0.0}

Table 1. Grid and Random Search comparison

Non-Technical

To optimize hyperparameters, we need to:

- select hyperparameters,
- define hyperparameter space, which is a range for each hyperparameter,
- select optimization methods such as grid search, random search, or Bayesian optimization. [1]

Each has its advantages and trade-offs. We also need to train the model with different hyperparameter combinations and use a performance metric such as:

- accuracy, [2]
- F1 score, [2]
- loss function to assess the model's performance for each set of hyperparameters. [2]

We can find the optimized hyperparameters via random search or Grid Search, and thus adjust the search space or hyperparameter values [3]. This iterative approach helps in gradually converging towards optimal hyperparameters. Implementation of K-fold cross-validation can ensure that the model's performance is not biased by the particular training-validation split. Cross-validation involves training the model on different subsets of the data and averaging the performance metrics [3]. We evaluate the selected hyperparameters from validation set or unseen data. After selecting the optimal hyperparameters using the validation set, we use test data to assess the model's performance and see if there is an overfitting issue. To prevent overfitting, we also consider regularization techniques (eg. L1 or L2 Regularization). [5]

Method	**Pros**	**Cons**
Random Search	Random search can handle high-dimensional parameter spaces more effectively.	It may lack exhaustive search of the entire hyperparameter space and it might take more iterations to find optimal hyperparameters.The slightly worse performance for the randomized search is likely due to a noise effect and would not carry over to a held-out test set.
Grid Search	Grid search is easy to implement and exhaustive by searching all possible combinations of the entire hyperparameter space.	- Computationally expensive: It can be computationally expensive especially for large hyperparameter spaces and high-dimensional parameter space.
Bayesian Optimization	It can be efficient in exploring promising regions. In addition, it can be suitable when evaluating the objective function is computationally expensive.	Since Bayesian optimization uses a probabilistic model to estimate the objective function, Bayesian optimization can be more complex than grid or random search to implement. In addition, the performance depends on the accuracy of the surrogate model used.

Table 2. Pros and Cons of Random Search. Grid Search and Bayesian Optimization

Hyperparameters include:

- learning rate,
- batch size,
- number of hidden layers,
- number of neurons in each layer.

Learning rate refers to the size of the steps taken during the optimization process. Too high of a learning rate may cause the model to overshoot the minimum, while too low may lead to slow convergence or getting stuck in local minima. [6]
Batch size represents the number of training samples used in one iteration. A larger batch size can lead to faster training but requires more memory. A smaller batch size provides a more stochastic training process.
The number of hidden layers determines the depth of the neural network. As the depth of the neural network increases, it can capture more complex features but may also be prone to overfitting and require more data. Too few the number of neurons may result in underfitting, while too many may lead to overfitting. To minimize the loss function, we use gradient descent, compare grid search with random search to find the optimal combination.

Optimizing the Bias-Variance Tradeoff

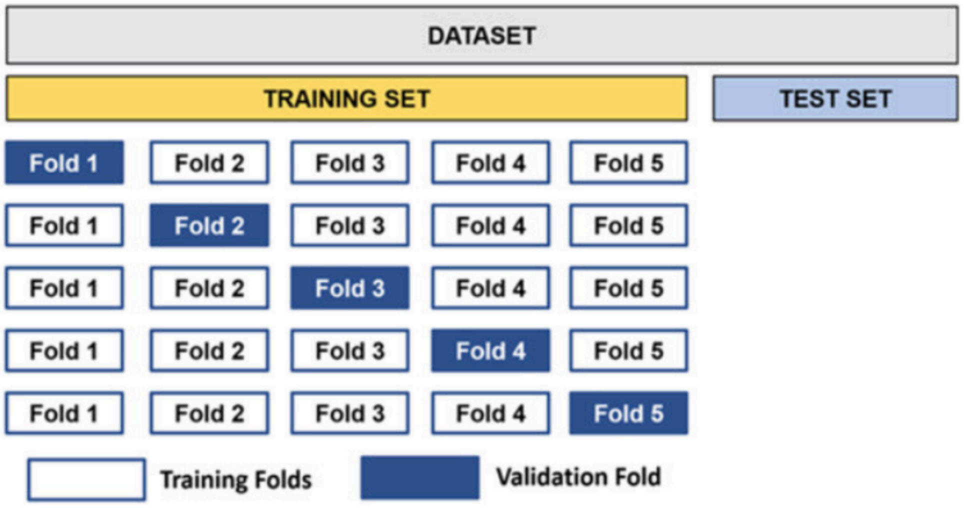
Technical

Underfitting brings high bias and low variance so to tackle this tradeoff we need to find the balance between underfitting and overfitting. [7]

Usually underfitting happens when there are not enough features used for modeling or at least the ones chosen are not important and cannot explain the relationship with the target variable [7]. So to avoid it, one can use Lasso regression to see which are the most important features and then select wisely which to use for modeling. [5]

The case with overfitting is a bit different. Overfitting mostly happens when the model has learned too much (even the noise in the data) and has a big R squared value in the training data but fails to generalize well and has a low R squared value over the unseen data [8]. So to avoid this from happening one can implement a cross-validation technique which represents a split of the data to train and test subsets. [3]

After this the training data is divided into training and validation subsets which are called a Fold. and since the data is split into k smaller equal sized folds so this is how it’s name is given: k-fold cross validation. K−1 segments are used for training and the remaining segment is used for validation. After this, a kind of rotation is executed where this time the k segment used for validation is set to a different segment while the rest of the data is used for training and this rotation repeats until every one of the segments has been used as validation set once while the rest of the data has been used for training. This is how at one point of time all of the data has been used either for training or for validation. Imagine it like every time the model learns something new (see the figure below for reference).



Source: Srivastava, Amiy, et al. "Ensemble Prediction of Mean Bubble Size in a Continuous Casting Mold Using Data Driven Modeling Techniques." *Machine Learning with Applications*, vol. 6, 2021.

Fig.1. K-fold cross validation schematics

The k-fold method can be described like this:

- *Randomize the dataset*
- *Split the data into train and test*
- *Rotate training and validation subsets*
- *Use the average of of all the folds as a optimal evaluation score*
- *Evaluate over the test data*

If in case the dataset is very big with a lot of features you can again tackle it with PCA where you can lower the dimensionality of the dataset. By implementing it actually you are doing some linear combinations with the features and then selecting the best ones. It is all good except that with this approach you lose a bit of the information from all the features, but the worst part is that also you lose a lot of the explainability and it makes it very much harder for further model interpretation. This makes Lasso the better choice for feature selection.

Non-techincal

Let's compare the train and test errors so that we could make some conclusions if there is under or overfitting. As we can see the training error is zero so we can conclude that this model is not underfitting.

The test errors are not that big but still we cannot say that it will generalize well to unseen data. We need some more analysis to be done.

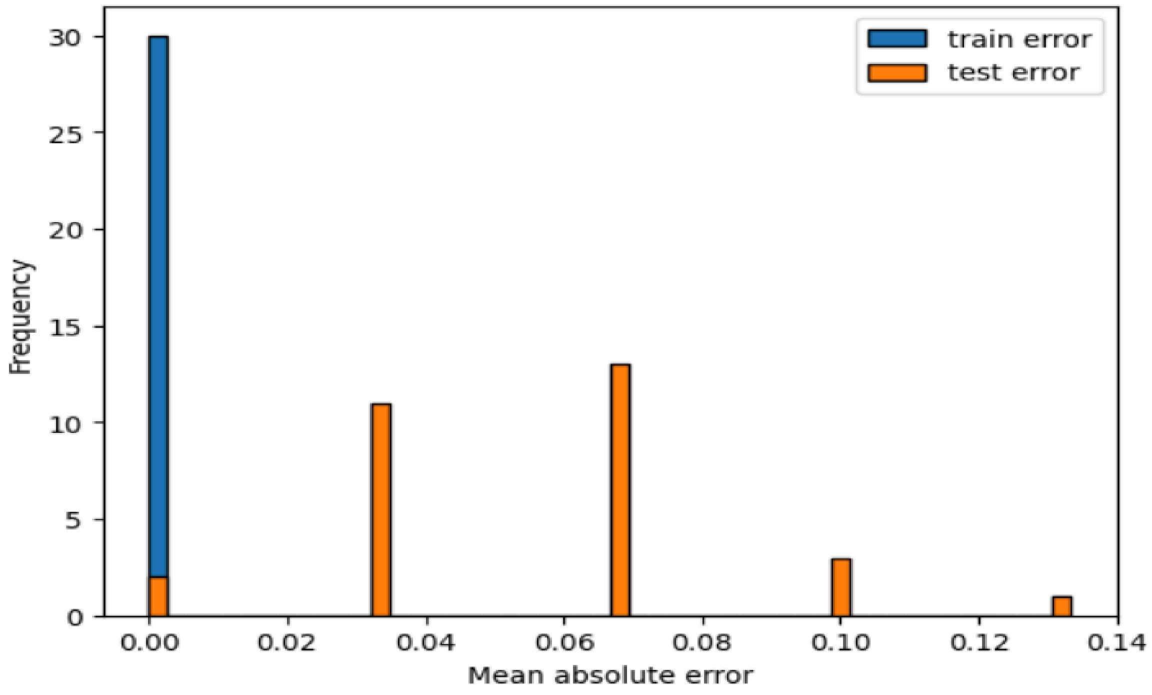


Fig.2 Train and test errors distribution via Cross-Validation

Information and confirmation about the tree depth we can reach from the validation curve analysis. In our case we are using a DecisionTreeRegressor and we are optimizing the depth of the tree, so that we could see at what depth what kind of bias vs variance we get. In the following chart we can see the visualization of this type of analysis.

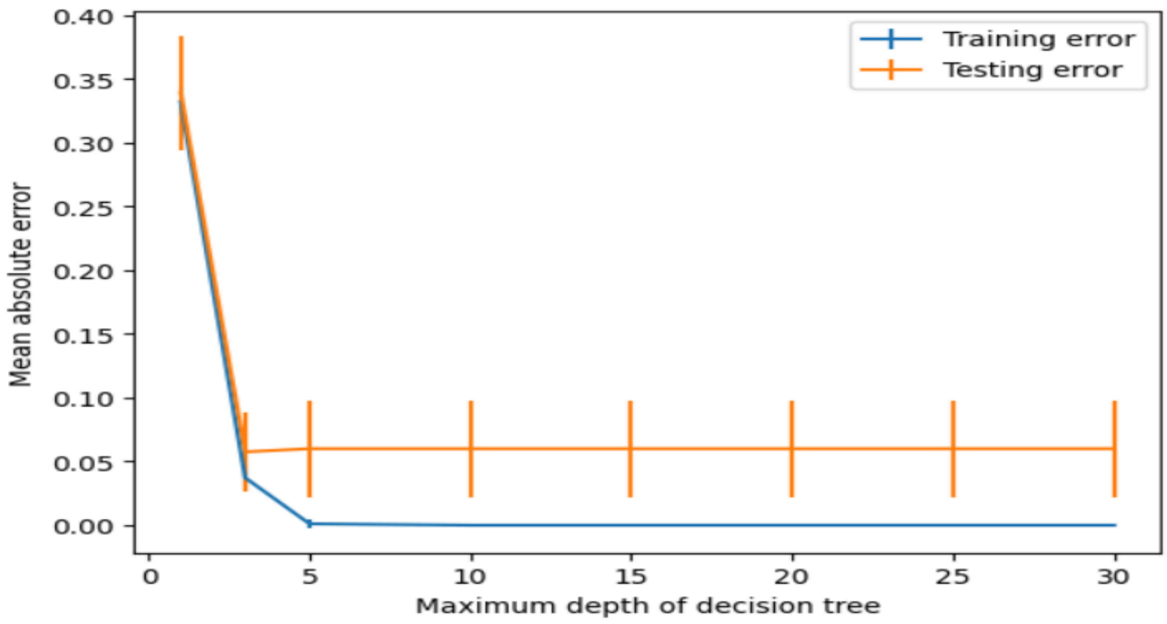


Fig.3. Validation Curve for Decision Tree using Training Errors
We can clearly see that at tree depth level 5 there is no more progressing lower of both the training and testing errors so this is a good value to choose for this parameter.

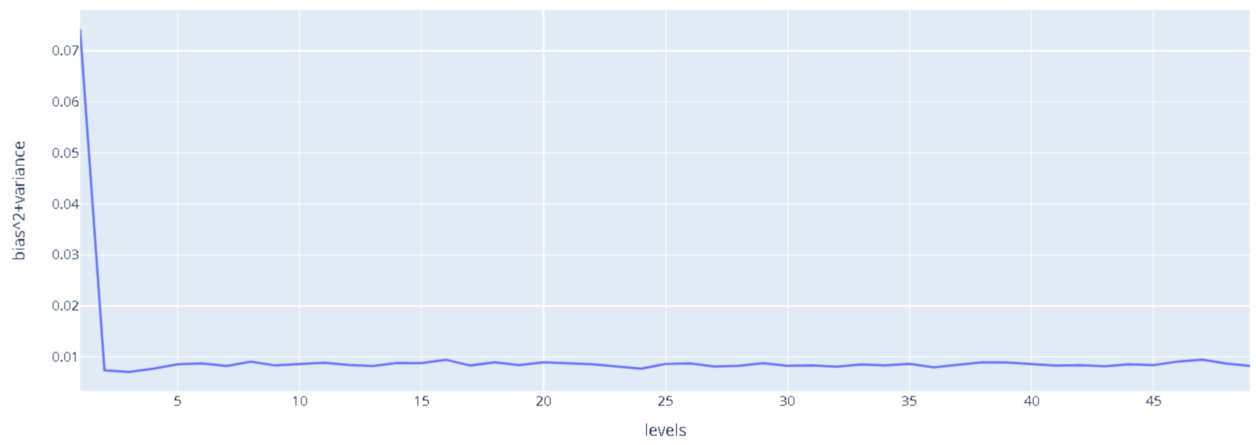


Fig.4. Bias variance tradeoff

Here we can see the bias-variance decomposition and we can conclude that level of tree depth is important towards lowering this bias-variance metric. It is well known that trees generally work well with non linear data and many features. Also we can see that after level 2 or level 3 of tree depth there is no more noticeable change in bias variance trade of so this will help us by saving the time for not necessary further calculations and further tree depth analysis. We can spare additional computing resources and use them elsewhere rather than compute more tree depth.
Combining this information with the previous analysis we calmly can set the tree depth parameter to 5 and we know that we will spare some additional computational time and resources by putting a reasonable setting for tree depth.

Ensemble Learning—Bagging, Boosting, or Stacking

Technical

To understand and compare the different Ensemble techniques we will be using the technical analysis below for a common classification problem.

Classification problem to analyze- Prediction of Luxembourg Index (LUXX) return exceeding 0.25% in any direction.

Techniques to be used- Bagging, Stacking, AdaBoost (AB) and Gradient Boosting (GB)

Predictors- combination of country indices and technical indicators. Technical indicators included are slow to fast moving average ratio (SMA_ratio), Relative Strength Index (RSI) and Rate of Change (RC).

Split of data- 80/20 for train/test data split

Tuning of hyperparameter- Learning rate, n_estimators

Assumptions- AdaBoost by default has Decision Tree classifiers as base learners with max_depth =1 as default. For the Gradient Boost classifiers, we assume default max_depth as 3. To reduce the computational time we are not changing all the hyperparameters to train the data and predict the output. The same is done with the XGBoost model as well.

In the Random Forest Classifier we are using max_depth as 2 and n_estimators as 10.

In StackingClassifier we are using Logistic regression as the final estimator

We are using these ensemble learning techniques to see if it's working better than the no-skill model (individual model) and we find that these techniques are performing better than the individual model to predict the outputs. We will be checking this with the help of the ROC curve metric as below and find out which models are performing well based on the AUC metric.

Fig

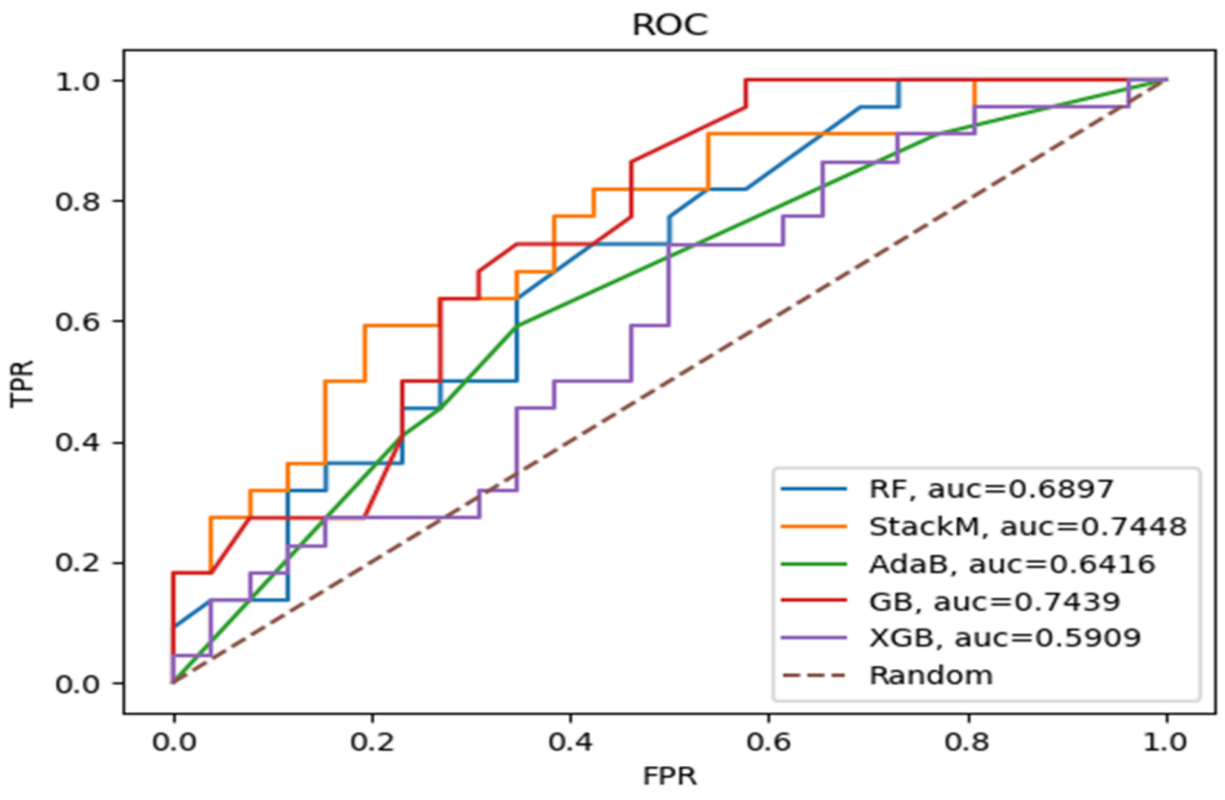


Fig.5. ROC curve metric comparison

Most of the models are having AUC greater than 70% which is a good value to say all the models are performed well. For models AdaBoost and XGM AUC is less than 70% but still it's a good number and if we adjust the hyperparameter then it will perform better than this.

Non-Technical

Ensemble Learning:

As the name suggests it is a technique to ensemble or combine predictions from multiple models to improve overall performance while leveraging the diversity of multiple models.

We have many Ensemble learning techniques for example- Bagging, Boosting or Stacking.

Bagging:

Bagging, also known as Bootstrap aggregation, is a way to reduce variance within a noisy dataset. In this technique instead of training on whole data we take sample data on which weak classifiers are trained ensuring that the sample size is sufficient to train. Bagging is restricted to homogeneous learners.

Boosting:

Boosting is a tree-based method in which weak learners are processed in sequential manner so that each weak learner is improved from the previous learner which means the final model is improved along the process. It is different from other parallel methods and hence it takes more time than Bagging (a parallel based process).

Stacking:

Stacking is different from the above two learning techniques as it is not restricted to homogeneous learners like Bagging. Also, Stacking is a parallel learning technique as opposed to boosting. However, Bagging is also parallel learning but in stacking the learning is done on sub-models' outputs and then learn how to combine them to make a better output prediction.

Marketing

(I imagine that I am speaking to a customer who is wondering whether to use ML or not)

My opinion is that there is no question about it and you should most definitely DO use such tools and just to name a few of its benefits will make you make up (wake up) your mind. So what are the benefits of using ML tools:

- You can process enormous quantities of data, such that if you have to do it manually it will take you if not years then maybe months (I doubt that you can do it), while with ML tools you can do it for minutes or at most hours.
- You can find hidden insights, patterns, trends in the data (especially in financial data), that are invisible to the human eye and will be left unseen and may stay not found at all.
- Make predictions with a very high accuracy (especially if the models are well designed with optimized hyper parameters). This will create the possibility for you to plan many steps in advance on your firm strategy or investing plan.
- Do a sentiment analysis and see "where the winds are blowing to". This will make you more adaptive and aware of the herd mentality, which will give you the opportunity of reactive risk management and avoiding unnecessary risks or events.
- You will be able to rank stock or other instruments on specific criteria, such as performance, volatility, growth potential, value, and quality and even more: with the so called reinforcement learning you will be able to make the computer to learns from its own trial and errors, to develop and execute optimal trading strategies (by itself).

Of course to be most transparent I will share with you some negative features like:

- You will have to hire a team of experts to do ML and research for you, which is not cheap at all
- If you don't like the idea of additional spending you can learn to do it by yourself, which especially at the beginning is quite difficult, but I promise you the efforts will be worth it. This is the harder but better choice because after that you will be able to manage all the processes and look and find the insight for yourself, rather than rely on others to do it for you.
- If you want to develop some kind of neural network for generative machine learning and use it like a chat bot, you will have to invest in a lot of hardware and processing power which as you know is not cheap.

I think that there is no question about whether to use ML or not. You have to do it, otherwise you will be like a dinosaur in the modern time and guess who will be more efficient, effective, quick in their work and will produce better results in research and development of whatever niche you can imagine.

Wake up your mind, the choice is yours.

REFERENCES:

1. Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, Marius Lindauer; Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges; Cornell University Submitted on 13 Jul 2021 (v1), last revised 24 Nov 2021 (this version, v3)
2. De Diego, I.M., Redondo, A.R., Fernández, R.R. et al. General Performance Score for classification problems. Appl Intell 52, 12049–12063 (2022). https://doi.org/10.1007/s10489-021-03041-7
3. Shayan Tabe-Bordbar, Amin Emad, Sihai Dave Zhao & Saurabh Sinha; A closer look at cross-validation for assessing the accuracy of gene regulatory networks and models; Nature Scientific Reports, published 26 April 2018; https://www.nature.com/articles/s41598-018-24937-4.pdf
4. Andonie, R. Hyperparameter optimization in learning systems. J Membr Comput 1, 279–291 (2019). https://doi.org/10.1007/s41965-019-00023-0
5. Robert Tibshirani, Regression Shrinkage and Selection via The Lasso: A Retrospective, Journal of the Royal Statistical Society Series B: Statistical Methodology, Volume 73, Issue 3, June 2011, Pages 273–282, https://doi.org/10.1111/j.1467-9868.2011.00771.x
6. Jason Brownlee; Understand the Impact of Learning Rate on Neural Network Performance; Deep Learning Performance; September 12, 2020; https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/
7. Jason Brownlee; Overfitting and Underfitting With Machine Learning Algorithms; Machine Learning Algorithms; August 12, 2019; https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/
8. Demšar J, Zupan B (2021) Hands-on training about overfitting. PLoS Comput Biol 17(3): e1008671. https://doi.org/10.1371/journal.pcbi.1008671
9. https://scikit-learn.org/stable/auto_examples/datasets/plot_digits_last_image.html#sphx-glr-auto-examples-datasets-plot-digits-last-image-py
10. https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html#sphx-glr-auto-examples-applications-plot-face-recognition-py
11. Dataset: http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz
12. https://scikit-learn.org/stable/auto_examples/decomposition/plot_faces_decomposition.html#sphx-glr-auto-examples-decomposition-plot-faces-decomposition-py

13. https://scikit-learn.org/stable/auto_examples/cluster/plot_dict_face_patches.html#sphx-glr-auto-examples-cluster-plot-dict-face-patches-py
14. https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_multioutput_face_completion.html#sphx-glr-auto-examples-miscellaneous-plot-multioutput-face-completion-py
15. https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html#sphx-glr-auto-examples-model-selection-plot-randomized-search-py
16. https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#sphx-glr-auto-examples-classification-plot-digits-classification-py
17. https://duchesnay.github.io/pystatsml/auto_gallery/ml_lab_face_recognition.html
18. https://scikit-learn.sourceforge.net/0.8/auto_examples/applications/face_recognition.html
19. https://towardsdatascience.com/bayesian-optimization-with-python-85c66df711ec
20. https://drlee.io/bayesian-optimization-with-python-b544255757d3
21. https://machinelearningmastery.com/what-is-bayesian-optimization/
22. https://modal-python.readthedocs.io/en/latest/content/examples/bayesian_optimization.html
23. https://www.analyticsvidhya.com/blog/2021/05/bayesian-optimization-bayes_opt-or-hyperopt/
24. https://pyro.ai/examples/bo.html
25. https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/
26. https://www.geeksforgeeks.org/stacking-in-machine-learning/
27. Ensemble comparison example from WQU Machine Learning in Finance Module 5