GROUP WORK PROJECT # 3
GROUP NUMBER: 6200

MScFE 632: Machine Learning in Finance

FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
Boyan Davidov	Bulgaria	davidovg@abv.bg	
Ivan Shigolakov	Russia	Shigolakov@yandex.ru	
Bharat Swami	India	bharatswami1299@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	
Team member 2	
Team member 3	

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.

Feedback from GWP2

The instructors comment we have concluded that we need to work on issues as mentioned below-

1. Our presentation of data.
2. Provide real life examples to support our argument.
3. Clearly explain the code used in the assignment.

Action taken by Members-

1. To tackle the first issue we have used new more informative images from valid sources with referencing them properly.
2. This time we used NIfty50 and S&P500 historical data in boosting and stacking ensemble learning section.
3. Added the comments and text with code in the script file.

Issue 1: Optimizing Hyperparameters

1.1 Technical:

Code is in the attached colab notebook.

1.2 Non-Technical:

Among the machine learning models the most common issue is overfitting. Overfitting is a situation when a ML model has good results on training dataset but bad results on testing data.

In order to handle the issue of overfitting and maintain good scoring results, i.e to reach a generalization, one needs to use hyperparameters tuning.

In this section we will use the Support Vector Classifier as an example to describe what hyperparameters this model has and how to choose the right parameters so as to obtain generalization.

First let's enumerate all the parameters that SVC model has (with default states):

S.no.	Parameter	Default value	S.no.	Parameter	Default Value
1	C	1.0	9	cache_size	200
2	break_ties	False	10	class_weight	None
3	coef0	0.0	11	decision_function)_shape	ovr
4	degree	3	12	gamma	scale
5	kernel	rbf	13	max_iter	-1
6	probability	False	14	random_state	None
7	shrinking	True	15	tol	0.001
8	verbose	False			

The most critical hyperparameters for Support Vector Machine Model are kernel, C and gamma.

1.2.1. Hyperparameter kernel.

The 'kernel' function has several different types (linear, poly, radial basis function (rbf), sigmoid and precomputed) and basically it is used to make transformation of training dataset to a higher dimension. The purpose of transformation is to make the data linearly separable. If one can draw a line between classes the linear kernel should be used. If it's impossible to separate classes linearly and we need to use some curve-like separation then a polynomial kernel might be a better choice. Radial basis function (rbf) is better for circulate type data.

1.2.2. Hyperparameter C.

As have been pointed out in previous group work project the hyperparameter C can help us to achieve the trade-off between the training error and the margin as it determines the penalty for misclassification during the training. As a rule the less the parameter C the larger a margin which is leading to more misclassification on the training data. The situation with low C can be useful when the data points are well-separated and there is not much noise or outliers. But there is a danger in setting parameter C too small as it can lead to underfitting and losses of potential patterns. Conversely, a large C parameter is leading to training error minimization. This situation can be very useful with not well-separated data points and if there is a presence of noise or outliers. With a high C parameter we can face a problem of overfitting when the SVC model is too specific to the training dataset but has poor results on new data.

1.2.3. Hyperparameter gamma.

The parameter gamma can be treated as a coefficient for radial basis function (rdf), poly and sigmoid. This hyperparameter has a high impact on the model performance. Gamma can take the following values: scale, auto or float. Like the parameter C, gamma parameter is inversely proportional to its distance. The higher the gamma the closer the points considered for the decision boundary. And conversely, the lower the gamma the farther the points that are considered for choosing the decision boundary. There is also another important feature of gamma: the higher the value of gamma the more the decision boundary scope gets closer to the points around it leading to more risk of overfitting and the lower the gamma value the smoother and regular the decision boundary surface gets leading to a less overfitting risk.

The only question one needs to answer is how can we find the optimal hyperparameters? The answer is using hyperparameter tuning techniques. We can point out three tuning methods: grid search, random search and Bayesian optimization.

Issue 2: Optimizing the Bias-Variance Tradeoff

2.1 Technical:

We assume that an independent variable x and a dependent variable y can be represented in the following form

$$y=f(x)+\epsilon$$

Here ϵ is an error term $\epsilon \sim (0,\sigma^2)$. It accounts for randomness or noise in the data.

Suppose we have found some machine learning model to predict y:

$$y=\widehat{f}(x).$$

In essence, a model would be fitting the data to minimize some loss function like:

$$\sum_{1}^{N}(\widehat{f}(x)_n - y_n)^2$$

The more complex the model is (in terms of parameters) the smaller the error will be. Now that we have fitted the function f(x) using the train set, as soon as a new data x’ observation comes (test set), the error for the prediction of y’ will be:

$$\widehat{f}(x') \; - f(x') - \epsilon$$

One might think most of the error is due to noise, or ϵ as introduced, however, it turns out the relation of the new data observation x’ wrt to the data our model has already seen (i.e. the training set) can have a big influence on the entire performance. A robust model (well fitted) will give the same prediction when faced with x’ whatever the training set it has seen. The average error is namely the bias of the model defined as:

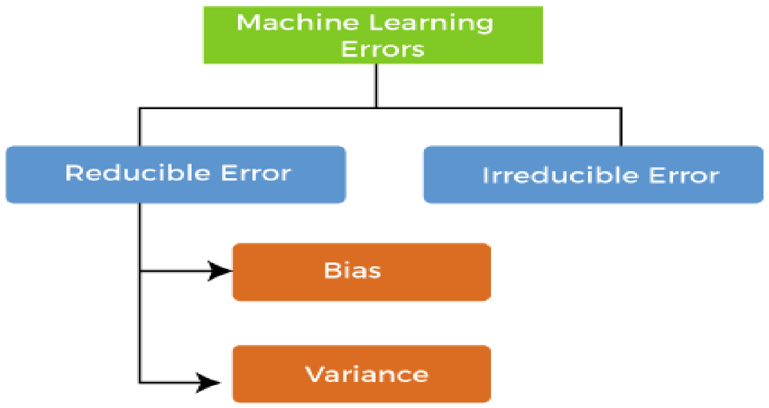
$$\text{BIAS}(\widehat{f}(x)) = (\text{E}[\widehat{f}(x')] \; - f(x')$$

Where E[...] denotes the expectation taken over samples of training sets with the same distribution as the training data. Computing the mean square error of the new prediction we have:

$$E[(\widehat{f}(x') \; - f(x') - \epsilon)^2] = (\text{BIAS}(\widehat{f}(x')))^2 + VAR(\widehat{f}(x')) + \sigma^2$$

where $VAR(\widehat{f}(x')) = E[\widehat{f}(x')^2] - E[f(x')]^2$

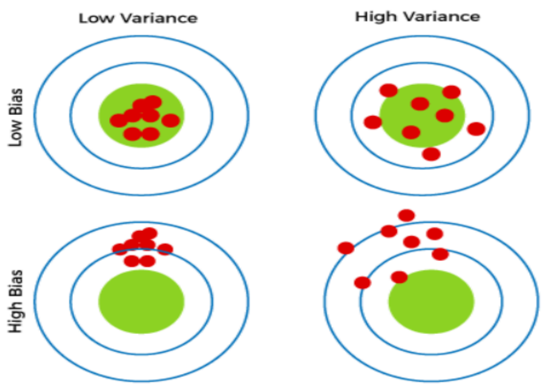
We can see that there two important terms we can control that influence the performance of our model: bias and variance. These we can minimize or at least find some good level where neither increases for marginal decrease of the other. In contrast, we can't control for the noise or randomness caused by ϵ and σ^2



Source: https://www.javatpoint.com/bias-and-variance-in-machine-learning

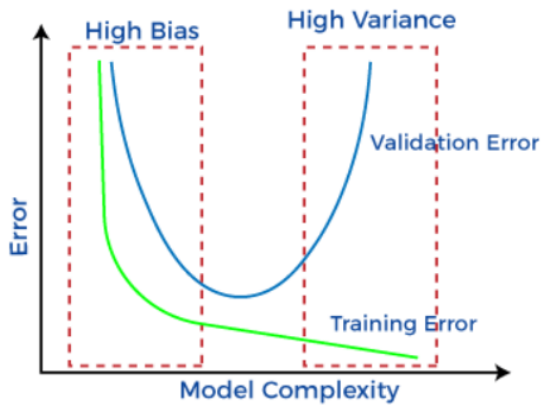
2.2 Non-Technical:

Bias is how far the prediction (or classification) of a trained model from the correct one on average. The variance is simply the magnitude of this bias. A good model will have low bias as well as low variance which can be schematically presented by the the hits on the left top target:



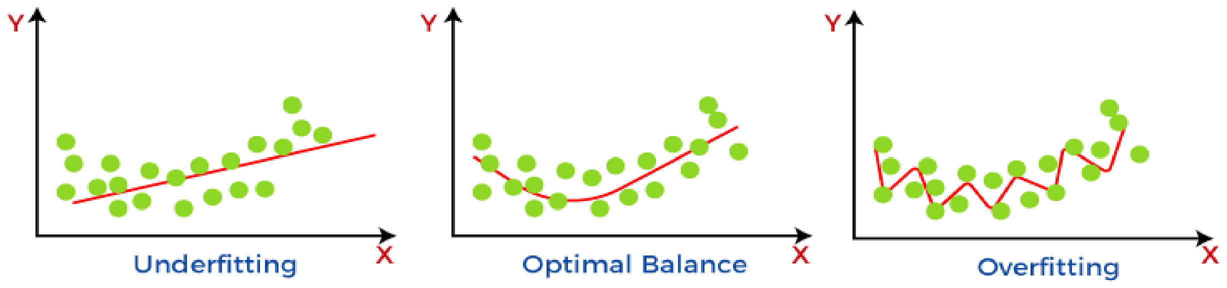
Source: https://www.javatpoint.com/bias-and-variance-in-machine-learning

A model that has high bias and high variance is the worst outcome, yet the question is whether high variance is better than high bias in general. The trade off between bias and variance is illustrated below:



Source: https://www.javatpoint.com/bias-and-variance-in-machine-learning

As it can be seen - unfortunately as the one is reduced it turns out the other increases (eg. low bias come at a price of high variance). The low bias while variance is high suggests overfitting. This is similar to a chess player that knows a certain opening by heart up to 20 moves but as soon as there is a move that has not been analyzed in the literature (even if it is a bad move) the player is not sure how to respond. Underfitting would be when our player knows only a couple of moves, on average she can play well but would be missing nuances after certain move or would take too long to take decision.



Source: https://www.javatpoint.com/bias-and-variance-in-machine-learning

To have a good model we need to arrive at the right proportion of bias and variance. This can be done by running our model with different parameters and plotting the bias vs variance. We can test with:

- Reduce/increase the features used in the model.
- Split differently the training/test data.
- Increase the Regularization.

Issue 3: Applying Ensemble Learning- Bagging, Boosting, or Stacking

Writer: Bharat Swami
Reviewer:

How can the models be used together to predict or solve a machine learning problem?
The answer to the above question is by using Ensemble Learning methods like bagging, boosting or stacking. Below, we are going to discuss these methods in more detail, and how they can be used to combine multiple models to create a model which performs best.

3.1 Non-Technical:

The process of solving a problem using machine learning involves different hard steps like starting from extracting raw data, preprocessing the data, visualizing the data, exploring the patterns, correlation between features, to modeling a ML model to predict the data and then calculating the error. We repeat the above process until we minimize the error on unseen data. These steps require understanding of complicated equations and visualization of hidden patterns.

But sometimes a single model is not able to do the job of predictions, some patterns are more complicated. If times refining a single model may cause problems like overfitting of data, issues related to computational power and memory space, etc.

To tackle the above problem we can do one thing - Combine the multiple machine learning models together. This process of combining multiple models together and creating a combination of models for a problem set is known as **Ensemble learning.**

Commonly we have three types of ensemble learning.

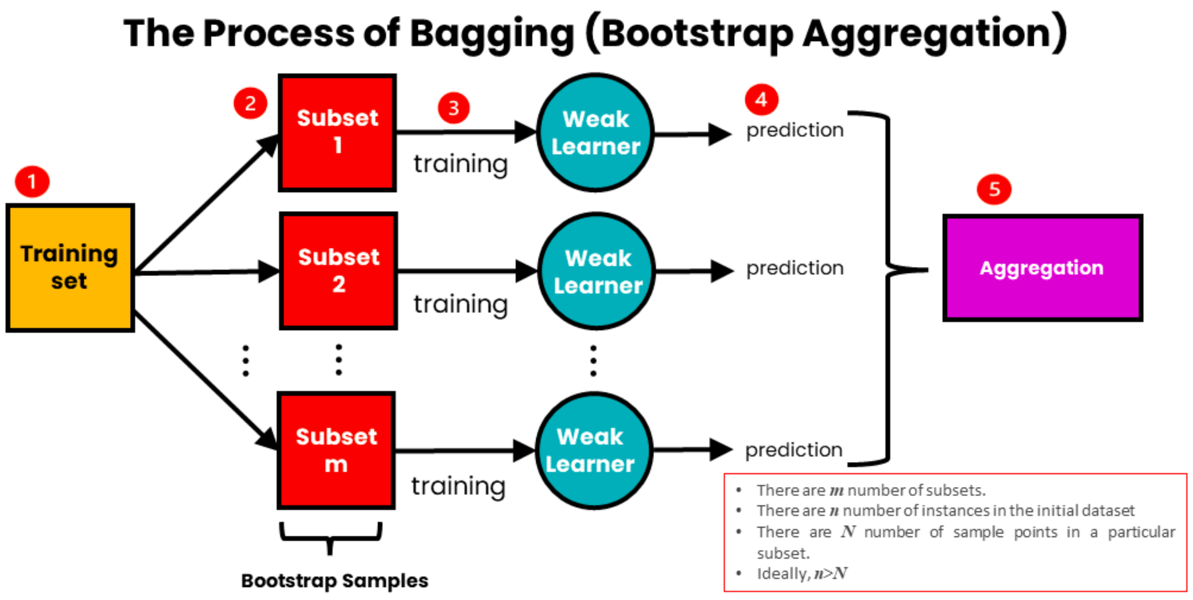
3.1.1 Bagging

Bagging is made up of two main keywords namely, bootstrapping and aggregation. Under these bagging combines multiple models to a generalized model.

When we train multiple models on the same data, there are high chances of similar results, which doesn't help us in any way.

To tackle the above problem we use a bootstrapping sampling strategy. Under this we create subsets of the dataset which use a replacement method.

And then we aggregate all the predictions from weak learners (weak models) predictions into final predictions while using methods like voting, i.e., taking the prediction as majority vote from weak models.


The Process of Bagging (Bootstrap Aggregation)
1
Training set
2
Subset 1
Subset 2
Subset m
Bootstrap Samples
3
training
training
training
Weak Learner
Weak Learner
Weak Learner
4
prediction
prediction
prediction
5
Aggregation
There are m number of subsets.
There are n number of instances in the initial dataset
There are N number of sample points in a particular subset.
Ideally, $n>N$

Above diagram shows a bagging ensemble method using multiple models on bootstrapped data.

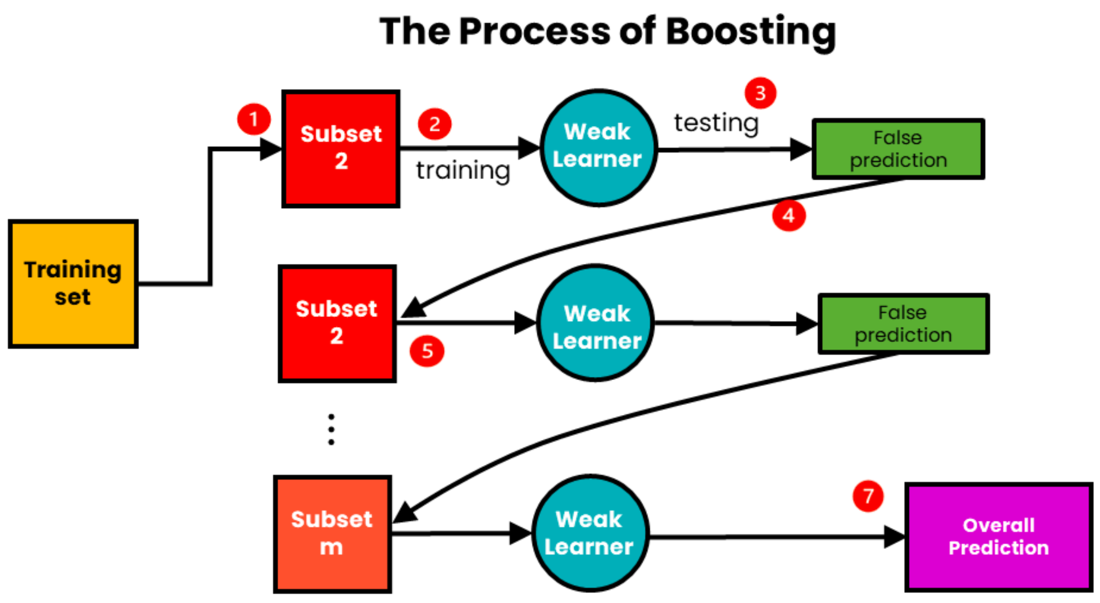
Below are the steps used in the bagging method.

- First divide the dataset into subsets of data instances, say m subsets from N initial data instances with replacements. Each subsets are of the same size N as of the initial dataset.
- Now with each subset dataset we train the weak learners. These learners are homogeneous in nature meaning they are of the same nature.
- Now each model runs on a dataset given to it and predicts the output.
- And finally these weak learner outputs are aggregated using max voting, averaging,etc. to predict the final output.

3.1.2 Boosting

Like bagging, we combine the multiple weak learners to predict the output. But unlike bagging, in boosting we are going to combine the weak learners in a sequential manner i.e, output from one weak learner is input for the next weak learner in the sequence, as shown in the diagram below.

Boosting helps to reduce the bias compared to the individual learners but using the weighting average technique. Under weighting average, we are going to reduce the weights of each correct prediction and increase the weights of each wrong prediction. This makes the next weak learners in sequence to work on errors from the previous learners.



Below are the steps involved in the boosting method-

- First we make m subsets from the original dataset.
- We train the first learner using a single subset and the incorrectly predicted data points to another subset.
- This updated subset will be the input for next weak learner.
- We repeat the above two processes until we train over all the weak learners.
- That finally we get the predicted output from the overall boosting ensemble learning.

we can also change the above steps as follow-

- At each set we use the entire dataset and change the weights of inputs for each weak learner while using the precision from the previous learner.
- If the previous learner predicts the output for an instance correctly we reduce the weights of that instance. If the output is incorrect we increase the weights of those instances and use these new updated weights for the next learner.
- By using the updated weights the next weak learner prioritizes the incorrect instances from the last learner and try to correct them and decrease the error.

3.1.3 Stacking

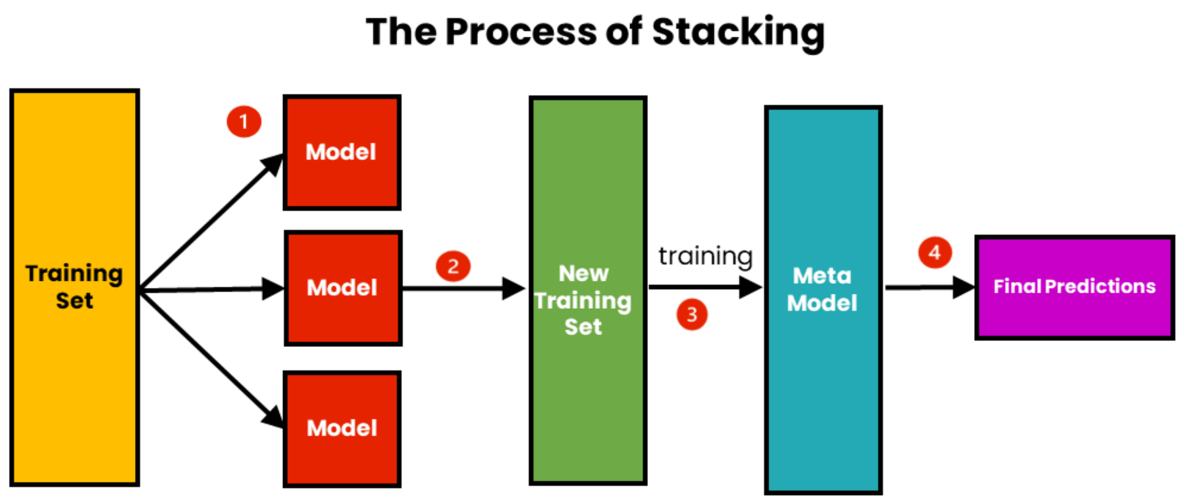
Like bagging and boosting, stacking also uses multiple models to predict the output.

- Unlike these learning, stacking combines strong learners.
- Stacking also uses the heterogeneous models.
- And finally stacking uses or creating a meta model.

Under stacking the initial dataset is used to train the heterogenous strong learner models. And these predicted data from learners are combined into a new dataset which will be input for a meta model.

The predictions from the learner are combined into a new dataset using the weighted average.

Stacking is a combination of bagging and boosting ensemble learning.



Below are the steps used in stacking-

- We train the learner on the original dataset.
- Combine the prediction from the learner using the weighting average to create a new dataset.
- Now the meta model is trained on a new dataset and predicts the final prediction.

3.1.4 Differences

Below are some main difference between the boosting, bagging and stacking-

Criteria	Bagging	Boosting	Stacking
Training	Parallel Combination	Sequential Combination	Uses Meta Model and Final While Aggregating the Prediction form Base Learners
Base Models	Homogenous Weak Models	Homogenous Weak Models	Heterogenous Weak or Strong Models
Sampling Of Subset	Replacement Based Random Sampling	No Subsets	No Subsets
Final Prediction	Max Voting or Averaging	Weighted Voting or Averaging	Meta Model
Examples	BaggingRegressor	GradientBoostingRegressor, XGBRegressor, LGBMRegressor algorithms.	StackingRegressor

3.2 Technical:

3.2.1 Bagging[1,2]

Below are mathematical steps involved with bagging-

1. Sampling using Bootstrapping-
    - Initial Dataset D with N instances i.e, $\{(x_i, y_i)\}^N_{i=1}$, create B bootstrap samples D_b for $b = 1, 2, 3, ..., B$.
    - Each bootstrap sample D_b is generated by sampling N data points from D with replacement.
2. Model Training:
    - Train a model f_b on each bootstrap sample D_b.
3. Predication Aggregation:
    - Fro regression tasks, the final prediction $\hat{y}$ is the average of the individual predictions:

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} f_b(x)$$

- For classification tasks, the final prediction $\hat{y}$ is the majority vote of the individual predictions:

$$\hat{y} = mode\{f_b(x)\}_{b=1}^{B}$$

3.2.2 Boosting[3,4]

Below are the mathematical steps for Boosting-

1. Initialize Weights:
   - Given a dataset D with N data points $\{(x_i, y_i)\}_{i=1}^{N}$, initialize the weights for each data point. Initially, all the weights are set equally:

$$w_i = \frac{1}{N},\ \forall i = 1, 2, 3,..., N$$

2. Model Training:
   - For m =1 to M (number of models):
     i. Train a model f_m using the weighted dataset.
     ii. Compute the error rate ϵ_m of the model f_m:

$$\epsilon_m = \frac{\sum_{i=1}^{N} w_i . I(y_i \neq f_m(x_i))}{\sum_{i=1}^{N} w_i}$$

where I is the indicator function that is 1 if $y_i \neq f_m(x_i)$ and 0 otherwise.

     iii. Compute the model's weights α_m:

$$\alpha_m = log(\frac{1-\epsilon_m}{\epsilon_m})$$

     iv. Update the weights for the data points:

$$w_i \leftarrow w_i . exp(\alpha_m . I(y_i \neq f_m(x_i))),\ \forall i = 1, 2,..., N$$

     v. Normalize the weights for the data points:

$$w_i \leftarrow \frac{w_i}{\sum_{j=1}^{N} w_j}, \ \forall i \ = 1, 2, \ldots, N$$

3. Prediction Aggregation:
   - The final prediction $\hat{y}$ is the weighted sum of the individual model predictions:

$$\hat{y} = sign(\sum_{m=1}^{M} \alpha_m f_m(x))$$

3.2.3 Stacking[5,6]

Below are the mathematical steps for stacking:

1. Base Model Training:
   - Given a dataset D with N data points $\{(x_i, y_i)\}_{i=1}^{N}$, train K base models

$$\{f_1, f_2, \ldots, f_K\}.$$

2. Generate Meta-Features:
   - for each data points x_i, generate meta-features by predicting the outputs from each base model:

$$z_i = \{f_1(x_i), f_2(x_i), \ldots, f_K(x_i)\}$$

   - Construct a new dataset D' with the meta-function z_i and original targets y_i:

$$D' = \{(z_i, y_i)\}_{i=1}^{N}$$

3. Meta-Model Training:
   - Train the meta-model g using the dataset D':

$$g(z_i) = g(f_1(x_i), f_2(x_i), \ldots, f_K(x_i))$$

4. Predictions:
   - For a new input x_{new}, obtain the predictions from each base model:

$$z_{new} = \{f_1(x_{new}), f_2(x_{new}), \ldots, f_K(x_{new})\}$$

   - Use the meta-model to predict the final output:

$$\hat{y} = g(z_{new})$$

3.2.4 Code

Code is in the attached colab notebook.

Review Section

	Wrote	Reviewed
Boyan Davidov	Issue 2	Issue 2
Ivan Shigolakov	Issue 1	Issue 3
Bharat Swami	Issue 3	Issue 1

Marketing

As we mentioned in GWP1 and GWP2 machine learning algorithms can be very useful in the market, predicting different things. We can predict the output in binary form, for example whether asset price will go up or down or we have also predicted the price with certain probability. All these are done using different machine learning models, namely-

1. Regression Model - OLS
2. Classification Model - Random Forest
3. ANN model with non-linearity using different activation function

All of the above have shown great results and have a great amount of potential to do in future with the help of questions answered in this GWP.

1. We can use Hyperparameter tuning to tune our hyperparameter for a model to give a more desired result.
2. We can study the Bias- Variance behavior as we have done in the code attached to find the point where we trade-off one for other that much to get the maximum from our model.
3. In some cases our data need more than one model to predict the output with good accuracy, or we can combine two or more models to get better results. So we can use the Ensemble learning where we can combine two or more models in different ways to get the job done.

Every day markets produce tons of data and if we process them properly and give some patterns or trends to create a strategy which produces a good sharpe ratio with good PnL ratio we can create a good profit. To do so we need various models to find the relations between our data and predict the future price with great confidence.

Machine Learning models can do the above tasks easily for us.

These tons of data can be reduced to useful data by using dimensional reduction techniques like Principal Component Analysis (PCA), etc which we have mentioned in the GWP2.

References:

1. Breiman, L. (1996). Bagging Predictors. Machine Learning, 24, 123-140.
2. Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. In International Workshop on Multiple Classifier Systems (pp. 1-15). Springer.
3. Freund, Y., & Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences, 55(1), 119-139.
4. Schapire, R. E. (2003). The Boosting Approach to Machine Learning: An Overview. In Nonlinear Estimation and Classification (pp. 149-171). Springer.
5. Wolpert, D. H. (1992). Stacked Generalization. Neural Networks, 5(2), 241-259.
6. Breiman, L. (1996). Stacked Regressions. Machine Learning, 24, 49-64.
7. Bagging, Boosting, and Stacking in Machine Learning
8. Mbali Kalirane (2024). Ensemble Learning in Machine Learning: Stacking, Bagging and Boosting
9. Paul Wilmott - Machine Learning: An Applied Mathematics Introduction
10. Bias and Variance in Machine Learning - Javatpoint