EPFL Machine Learning project 2: Calibrate a model of OTC markets

Qais Humeid, Louis Jacques Vu-long Leclair, Ki Beom Kim
EPFL Lausanne, Switzerland

***Abstract*—This paper discusses various machine learning methods used to learn a function based on a model for Over-The-Counter (OTC) markets. We concluded that the best model is a Neural Network, which delivers appropriate results. Finally, we solve a constrained minimization problem that finds the input vector that minimizes our learned function.**

I. Introduction

This project is based on the working paper titled "Frictional Intermediation in Over-The-Counter Markets" by Julien Hugonnier, Benjamin Lester, and Pierre-Oliver Weill. The general idea behind their research is to maximize the utility of buyers and sellers of municipal bonds with multiple dealers (intermediaries). In our project, we use machine learning to calibrate the OTC market model of Hugonnier, Lester, and Weill to transaction level data coming out of the municipal bonds market.
The municipal bonds market data consists of 9 entries per row. The first 8 entries are input parameters, while the 9th is output value which represents the norm:

$$O(\theta) = ||T_{\text{data}} - T_{\text{model}}(\theta)||$$

Where T_{data} and T_{model} have already been found previously.

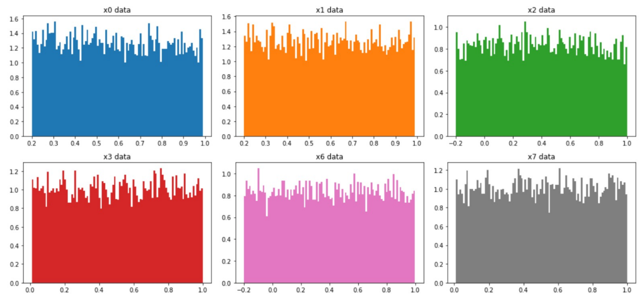
The objective is to apply various machine learning techniques to learn our function $O(\theta)$. After we get a reasonable estimator $O(\hat{\theta})$, we need optimize it in order to find the input vector θ^* that minimizes this function.

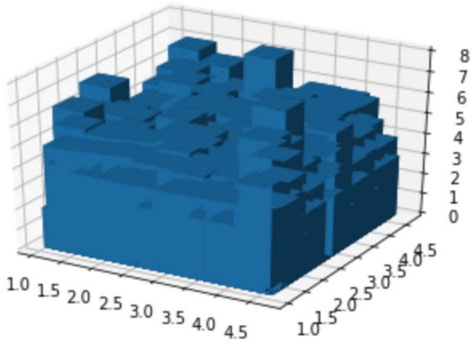
II. Data analysis and exploration

The paper mentions that the input data had been generated independently from the following hypercube:

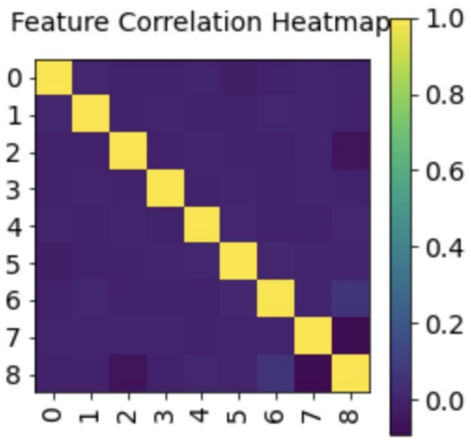
$$[0.2, 0.99] \times [0.2, 0.99] \times [-0.2, 1] \times [0.01, 1] \times [0.8, 4]^2 \times [-0.2, 1] \times [0.01, 1]$$

We quickly verify this heuristically by looking at the histograms (and joint histogram for ((θ_5, θ_6)). The histograms below were generated with the first 10,000 entries of the data set.





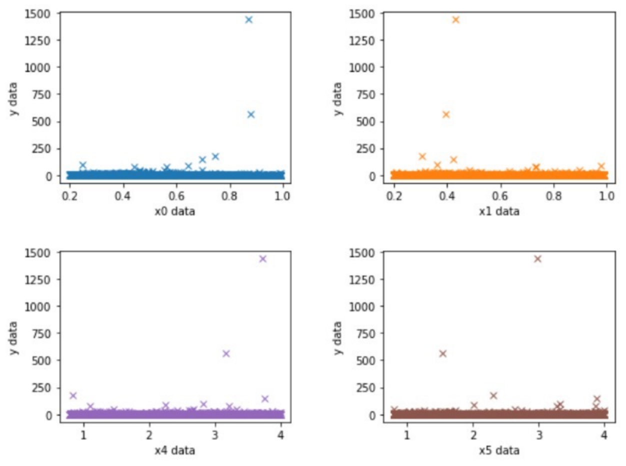
Clearly, the histograms point that the data are uniformly distributed. Furthermore, looking at the correlation heat map (0 through 7), we verify independence.

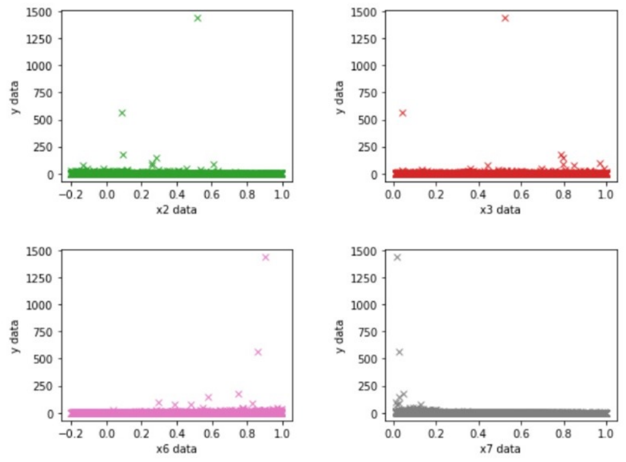


Since we know the theoretical distribution of our input data, we can easily compute the summary statistics, which are:

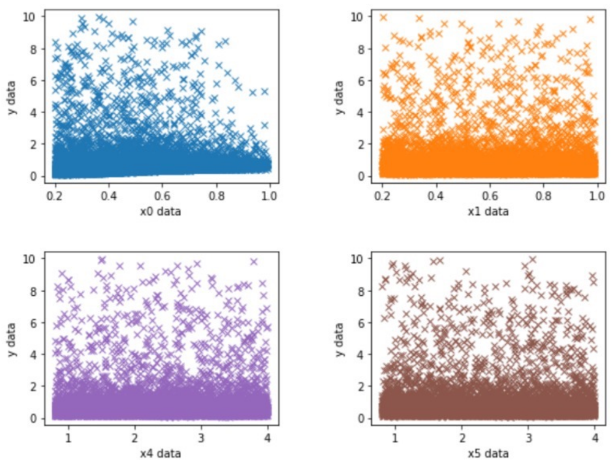
Feature	x0	x1	x2	x3	x4	x5	x6	x7
Mean	0.5917	0.5951	0.3994	0.5046	2.4053	2.3984	0.4025	0.5086
Std. dev.	0.2281	0.2287	0.3456	0.2869	0.9226	0.9241	0.3456	0.2856

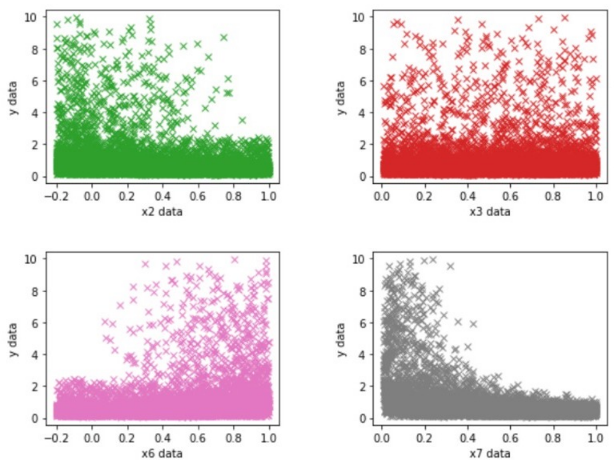
Moving on, before building our models, it makes sense to look at how the response varies with the input. Below are the scatter plots of the output vs the inputs.





We realized that for some input values, we have abnormally high norms, which results into ”flattened” graphs. Therefore, we chose to remove all norm values greater than 10 to make our graphs more ineligible.





From this, we gain interesting results. For the input values of x_2, x_7, we notice that the norm tends to be much higher for lower input values (negative correlation). Also, for higher values of x_6, we have higher norm values (positive correlation). For the rest of the variables, it seems that there is no apparent pattern. This conclusion can also be reached from the heat map when looking at column 8 (which represents the norm). As we can see, there are darker squares in rows 2 and 7 (indicating negative correlation) and a lighter square in row 6 (indicating a positive correlation). This means that variables x_2, x_6, x_7 will be the most influential.

III. Learning Our Function

Our first goal is to find a reasonable approximation of the function $O(\theta)$. Since it represents the norm, we know that it is bounded from below by zero, i.e., $O(\theta) \geq 0, \forall\theta$.

A. *Linear Regression, LASSO, SGD Regressor, XGBOOST*

The best way to build a model is by starting simple and adding complexity as you go. Therefore, we began by using the simplest techniques we know. The main libraries that we used to help us implement more advanced techniques were SciKitLearn and XGBoost.

First, we looked at the fit the resulted from fitting a simple linear model to our data using the mean-absolute error. We got the following results:

MAE	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7
1.1887	-0.3597	-0.166	-1.6155	0.0435	0.0555	-0.0308	1.5908	-3.0252

Second, we implemented a LASSO to see if the coefficients of the features that are apparently uncorrelated with the response are shrunk to zero; this was not the case. To find the best trade off parameter, we used cross validation with a 75-25 train-test split. We got the following results:

MAE	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7
1.0423	0.0000	0.0000	0.0000	0.0522	0.0522	0.0000	1.6081	0.0000

Third, we decided to fit another regularized model, but this time, we used the Mean-absolute error and a L1-penalty. In specific, its name is SGDRegressor from the SciKitLearn library; as it uses SGD to find the optimal parameters. We got the following results:

MAE	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7
1.1220	-0.4041	-0.2290	-1.5526	0.0000	0.0000	0.0000	1.5443	-2.9751

Finally, we tried XGBoost with 'gbtree' booster. Unlike other methods shown above, this method is based on decision tree algorithm. We obtained lowest mean absolute error of 0.571089, which is much better compared to other methods we already implemented above. However, since the model is very complex and it is difficult to minimize itself. Hence this method is not recommended especially for this kind of project.

The conclusions that we reached are all of these models are completely incorrect. For these linear models, our response is unconstrained. This means that negative values are permitted. This goes against our rational as the value of $O(\theta)$ we are trying to predict MUST be greater than or equal to 0. One way this can be remedied is by restricting the coefficients of our model to be positive; this can easily be implemented using SciKitLearn. In this case, however, another issue presents itself. Forcing our coefficients to be positive means that our function will be increasing in θ. Since our input space is positive, the minimum value will be achieved at $\theta = 0$. Also, we saw earlier that x_2, x_7 are negatively correlated with the norm. A positive signed coefficient for these two variables would be counter intuitive.

Another way one can go about this is by forcing any negative value of $\hat{O}$ to be 0 i.e. define $\tilde{O} = \max(0, \hat{O})$. This results in another issue. All negative values are forced to 0 meaning that we will have A LOT of minima at 0. This is counter productive as some negative values might be smaller than others, but both are mapped to 0. This might lead to the loss of important information, especially in the second part of this project in which we need to minimize our function.

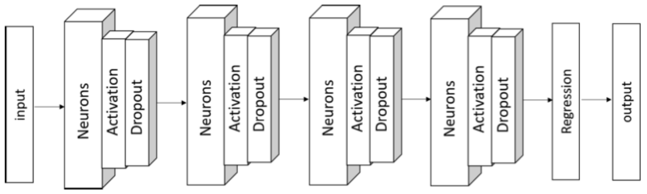
Therefore, we will have to look for alternative and more complicated methods that satisfy our requirements.

B. Neural Nets and Convolutional Neural Nets

The rationale behind using NNs is their high degree of flexibility. Professor Hugonnier also sent us a research paper suggesting we implement their methods (https://arxiv.org/pdf/1904.10523.pdf).

We decided to implement the model shown in the research paper. The parameters are detailed in the figure below:

Parameters	Options
Hidden layers	4
Neurons(each layer)	200
Activation	ReLu
Dropout rate	0.0
Batch-normalization	No
Initialization	Glorot_uniform
Optimizer	Adam
Batch size	1024



Please note that we initialization parameter was not used. We also decided that the dropout rate should remain at 0 as we do not have many input parameters.

We are sure that this NN will output only positive values, as the ReLu activation function only results in values larger than 0.
We were really impressed with the results of the NN which were:

	$Epoch_0$	$Epoch_1$	$Epoch_2$	$Epoch_3$	$Epoch_4$	$Epoch_5$	$Epoch_6$	$Epoch_7$	$Epoch_8$	$Epoch_9$
MAE	0.436	0.379	0.327	0.304	0.297	0.262	0.301	0.277	0.249	0.219

Next, we wanted to see if a CNN would produce better results. We used the following parameters:

Learning rate	0.001
No. Epochs	10
Batch size	32
Activation function	ReLU
1st conv. layer	3x3, 30 feature maps
2nd conv. layer	3x3, 60 feature maps
Hidden layer	50

The result of the CNN model was:

	$Epoch_0$	$Epoch_1$	$Epoch_2$	$Epoch_3$	$Epoch_4$	$Epoch_5$	$Epoch_6$	$Epoch_7$	$Epoch_8$	$Epoch_9$
MAE	0.474	0.450	0.466	0.453	0.469	0.460	0.470	0.472	0.463	0.466

not much better than the previously implemented NN. Moreover, learning our function took a very long time. We realized that learning our function via a CNN is a complete overkill. First, it is very time consuming. Second, our input data is only of 8 dimensions. A CNN is usually used on very high dimensional data such as images.

Therefore, we decided to stick with the result given by the NN.

IV. Optimizing Over The Input Space

The second part of our project consisted of finding the value θ^* that minimizes $\hat{O(\theta)}$, given the constraint that $\theta \in [0,1]^2 \times \mathbf{R}_+^6$. In mathematical terms:

Let $C = [0,1]^2 \times \mathbf{R}_+^6$. We want:

$$\theta^* = \arg\min_{\theta \in C} \hat{O(\theta)}$$

Since we learned our function using NN, is very complicated. The best way to minimize a complicated function according so some constraints is numerically via projected gradient descent (PGD). This will be done according to the following procedure:

The projection will be done by thresh-holding all values smaller than 0 to 0 for all features, and by thresh-holding all values greater than 1 to 1 for the first two features.

Our function is also not convex, thus, we are not guaranteed to find the global minimum (only local convergence). We will repeat this procedure several times with different random starting initialization for θ.

We chose to run three runs of 500 different starting points (different seeds), totalling to 1500 different initial values of θ. As for the PGD optimizer, we used a built in optimization function provided by SciPy that allowed us to bound our input space. Below are tables which list the "average" θ along with the "average" minimum, along with the value of θ that results in the absolute minimum for each run.

Minimum average (Set 1), MAE = 0.45937

θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
0.48851	0.50962	0.59072	0.51026	0.53332	0.55462	0.50875	0.62048

Minimum average (Set 2), MAE = 0.43753

θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
0.47537	0.50652	0.59096	0.50192	0.52344	0.56336	0.49446	0.58496

Minimum average (Set 3), MAE = 0.47290

θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
0.50571	0.50539	0.56180	0.51547	0.50914	0.55353	0.47834	0.64293

Absolute minimum (Set 1), MAE = 0.04257

θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
0.07977	0.80797	0.07394	1.28058	0.97989	0.36174	0.24932	0.14062

Absolute minimum (Set 2), MAE = 0.03899

θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
0.05360	0.47977	0.76223	0.20533	0.91204	1.25343	0.81162	0.12514

Absolute minimum (Set 3), MAE = 0.06657

θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
0.33173	0.29821	0.65092	0.66920	0.77390	0.97088	0.28081	0.79630

As seen in the tables above, the average θ stays similar across the 3 separate runs (each with 500 initial values). The average minimum value of $\hat{O}$ is around 0.45. For the values of θ that give the absolute minimum value of $\hat{O}$ for each run, the results vary slightly. Although the vectors are not exactly the same, they are still within the same order of magnitude (and not too far apart). Also, the different minima find are all of the order 10^{-2}.

As a final remark, we recommend the following values of θ:

MINIMUM AVERAGE:

θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
0.47537	0.50652	0.59096	0.50192	0.52344	0.56336	0.49446	0.58496

ABSOLUTE MINIMUM:

θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
0.05360	0.47977	0.76223	0.20533	0.91204	1.25343	0.81162	0.12514

V. Conclusion

In conclusion, the goal of this project was to find a reasonable approximation of $O(\theta) = ||T_{\text{data}} - T_{\text{model}}(\theta)||$. After discovering the basic models such as linear regression and LASSO are not appropriate, we decided to apply more complicated techniques such as Neural Networks and Convolutional Neural Networds. We finally settled with a regular NN as CNN was too ambitious for the type of problem we had. Although our result obtained via the NN is very complicated, we do not really care about the interpretation of the resulting function. As we are trying to learn the "norm", interpretation does not really matter. The important thing is that our function results in positive values (which it did). Finally, we were able to optimize our resulting function over 1500 different random starting point giving consistent results for the minima achieved.