The Lightweight IBM Cloud Garage Method for Data Science

Architectural Decisions Document Template

1 NoteBook Table of Contents and Objectives

We will be using a Kaggle competition dataset with the objective to target the next month sales items: https://www.kaggle.com/c/competitive-data-science-predict-future-sales

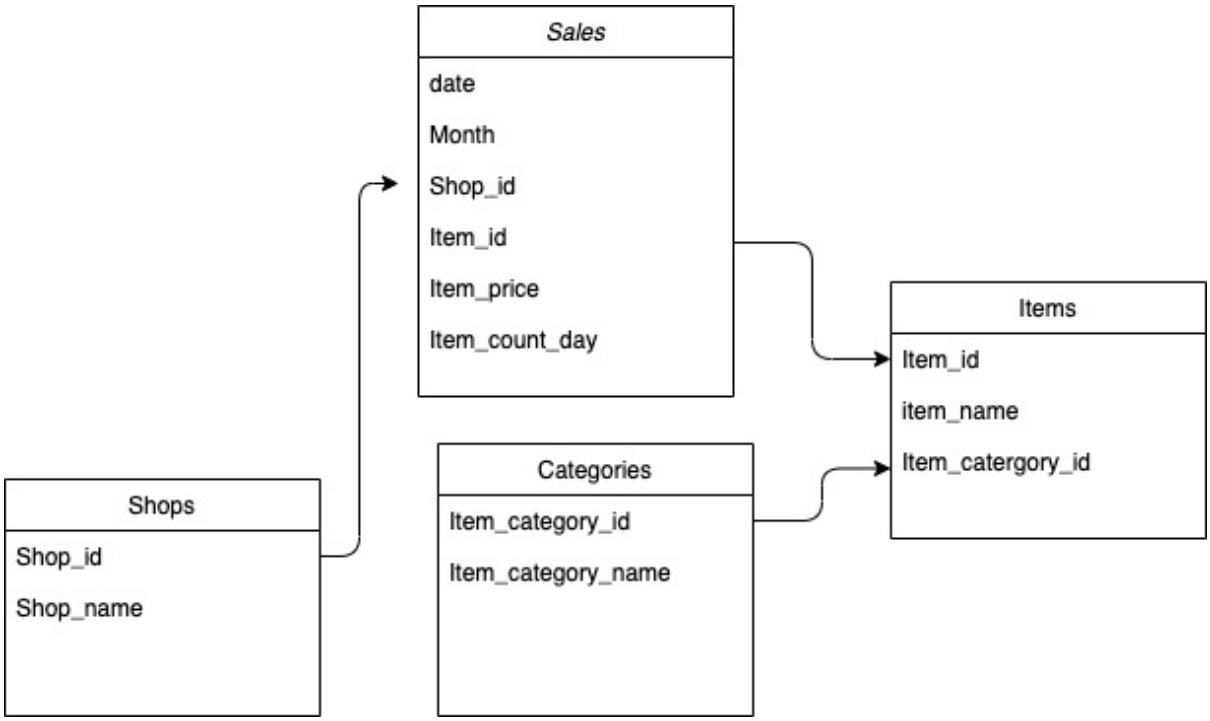
The objective if to see if deep learning can bring better results than a boosting tree on a structured and static dataset

Contents

2 Extract, Transform, Load (ETL)

There are 5 dataset available.

1. The sales_train: This dataset is composed of the sales and return for every single item since the January 1st, 2015 by shop_id and by item_id
2. The items: This dataset provides more information about the items available. It provides an item_id, item_name and item_category. The item_id is the same than the sales_train and should be use to merge the dataset together
3. The Shops : This dataset provides a Shop_id and a shop name. This can be merge to the main sales dataset towards the shop_id
4. The Categories: This dataset provides category_item name and id and can be merged with the items dataset towards the id.



In the sales dataset, the dates are presented as such : 01.01.2013 (Object) . This format is not understood by Python and can lead to wrong data exploration. I have used the below line of code to be able to change the dates into a datetime YYYY-MM-DD format.

```
startTime = datetime.now()
date_parts = train['date'].apply(lambda d: pd.Series(int(n) for n in d.split('.')))
date_parts.columns = ['day', 'month', 'year']
train['date'] = pd.to_datetime(date_parts)
```

3 Exploratory Data Analysis and Features Engineering (EDA)

3.1 Outliers and NaN management

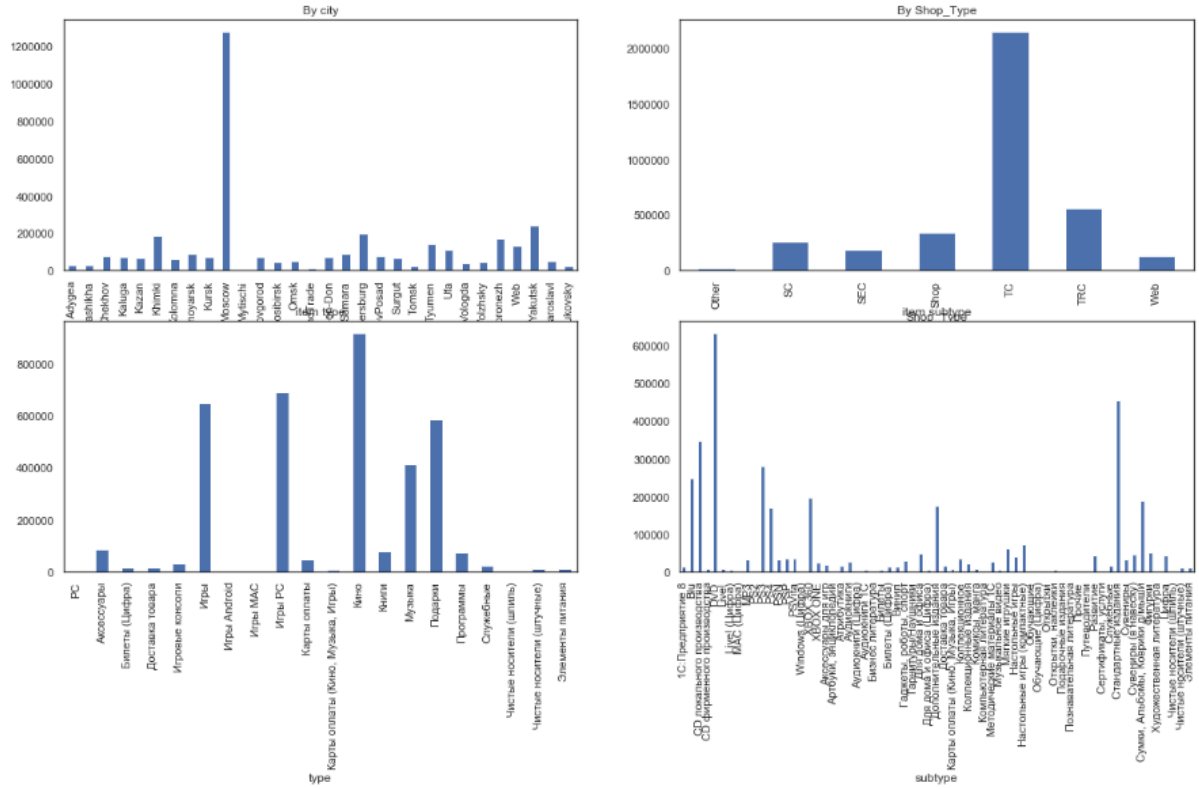
We found outliers and NaN that we have replaced with Medians. We have also found shop duplicates that we have removed

3.2 Datasets exploration

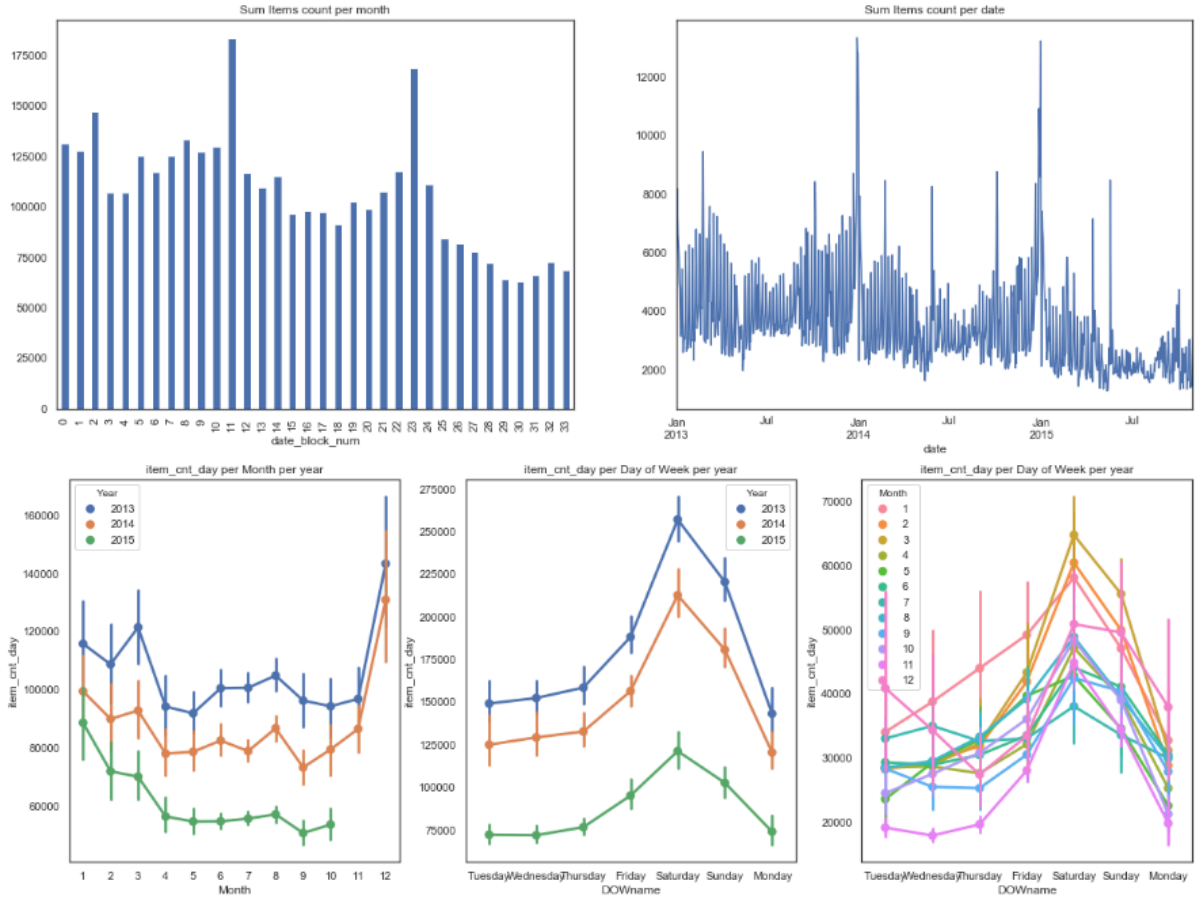
The categories, shops and items datasets have much more data than just a name in their "name". I have worked to extract and split the information to create more features.

- Shop_name can be divided into shop name, city, and shop type
- Category name can be divided into name, type and subtype

This helped us to see that most sales happened in Moscow



The date analysis and seasonality has also brought some interesting outcomes. We can see that the amount of sales is in a downward long-term trend and that we have monthly and weekly seasonality. While the objective is to estimate the number of items sold the whole next month, weekly seasonality remains important. For example, Saturday has the highest number of sales. A month with 5 Saturdays can then have more sales than a month of 4 Saturdays



3.3 Data Structure and features creation

The below are all the different steps that have been performed to prepare the data. After preparation the data is 2.4 Gb big.

1. Prepare test data
2. Create Initial empty Matrix with Month number, Shop_id, and Item_id
3. Concatenate the Matrix data with the Test data to have all shops and items in Matrix
4. Update datatypes
5. Compute revenue of train_set
6. Structure train data like Matrix and add item count to Matrix
7. Merge Shops, Items and Category data into Matrix
8. Update Matrix Datatypes
9. Create Lag features for 1,3,6,12 months on items counts
10. Compute average items count per month
11. Compute average items count per month per shop
12. Compute average items count per month per shop
13. Compute average items count per month per category id
14. Compute average items count per month per shop id per category id
15. Compute average items count per month per shop id per type code
16. Compute average items count per month per shop id per subtype code
17. Compute average items count per month per city_code
18. Compute average items count per month per item_id per city_code
19. Compute average items count per month per type_code
20. Compute average items count per month per subtype_code
21. Compute average items price
22. Compute average items price per month
23. Add price lag per month
24. Drop lag features
25. Compute revenue per shop per month
26. Compute revenue per shop
27. Number of days in Month and days of week
28. Item shop last sale
29. Item last sale
30. Item shop first sale
31. Item first sale

4 The Catboost Model

The data is composed of continuous and categorical data. The great advantage of the Catboost against another boosting tree is that it does not need heavy preprocessing for the categories. It is only require mentioning the categories column, which can help saving a lot.

In order to tune the hyper parameters I have used a grid search with below settings:

```
grid = {
'learning_rate': [0.05,0.1,0.2,0.3,0.4,0.5]
   ,'depth':[2,4,6,8,10]
   ,'l2_leaf_reg': [1, 3, 5, 7, 9]
   }
```

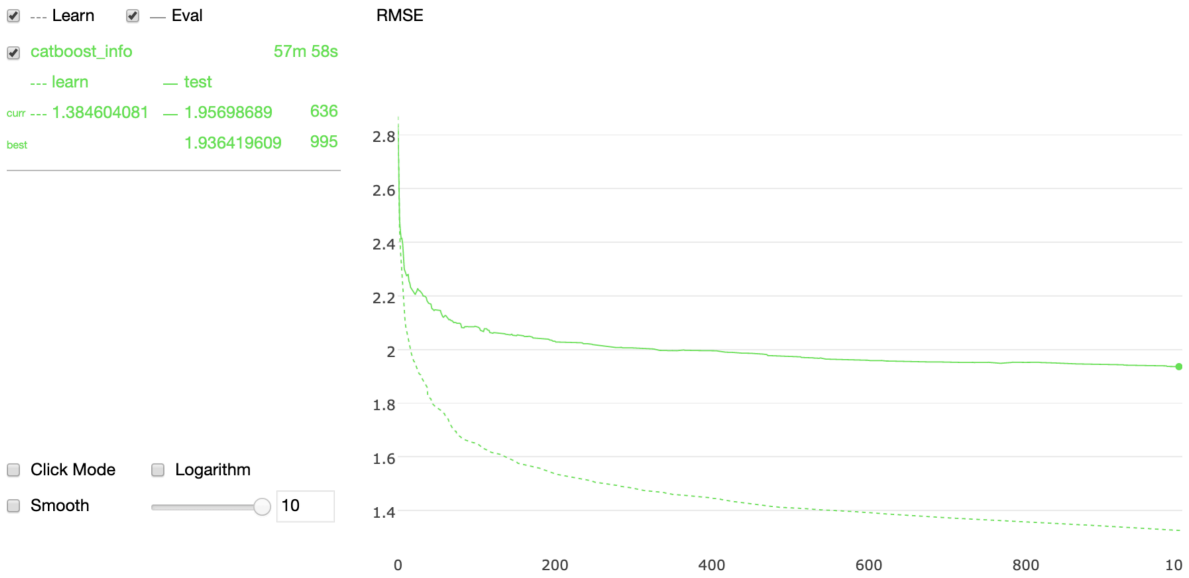
As the dataset is quite large with more than a million row, I have randomly selected 10% of the train set and the test set to run the hyper parameter tuning. I have noted that it is important to totally restart the Kernel before launching the hyper parameter tuning to increase speed.

The over fitting risk is quite visible on the chart. You can see the learning RMSE decreasing every time a higher tree depth is tested while the RMSE test is slightly increasing over time. The large ups and downs in the test RMSE are the change in learning rate. The highest learning RMSE were hit when the learning rate was low (0.05) and the lows when the learning rate was high (0.5) this difference was however not visible on the test part. The L2_leaf had very little impact on both test and learn RMSE

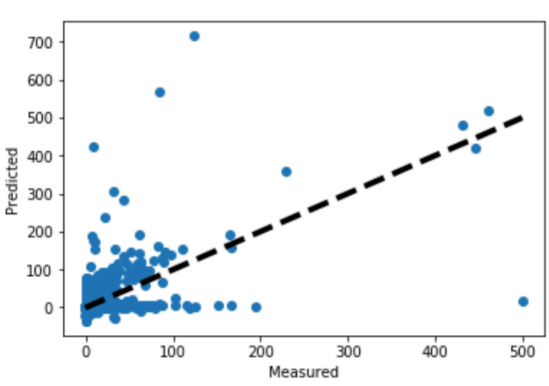
Catboost comes with an easy way to calibrate the hyperparameters. The Tuning test suggested the below:

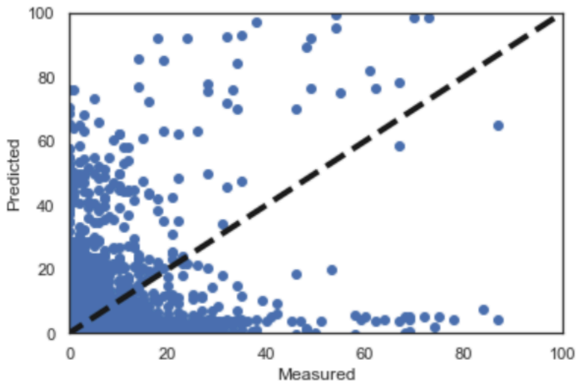
```
{'depth': 6, 'l2_leaf_reg': 1, 'learning_rate': 0.3}
```

On the whole dataset, it took 1h30 minutes for catboost to do 1000 iterations.

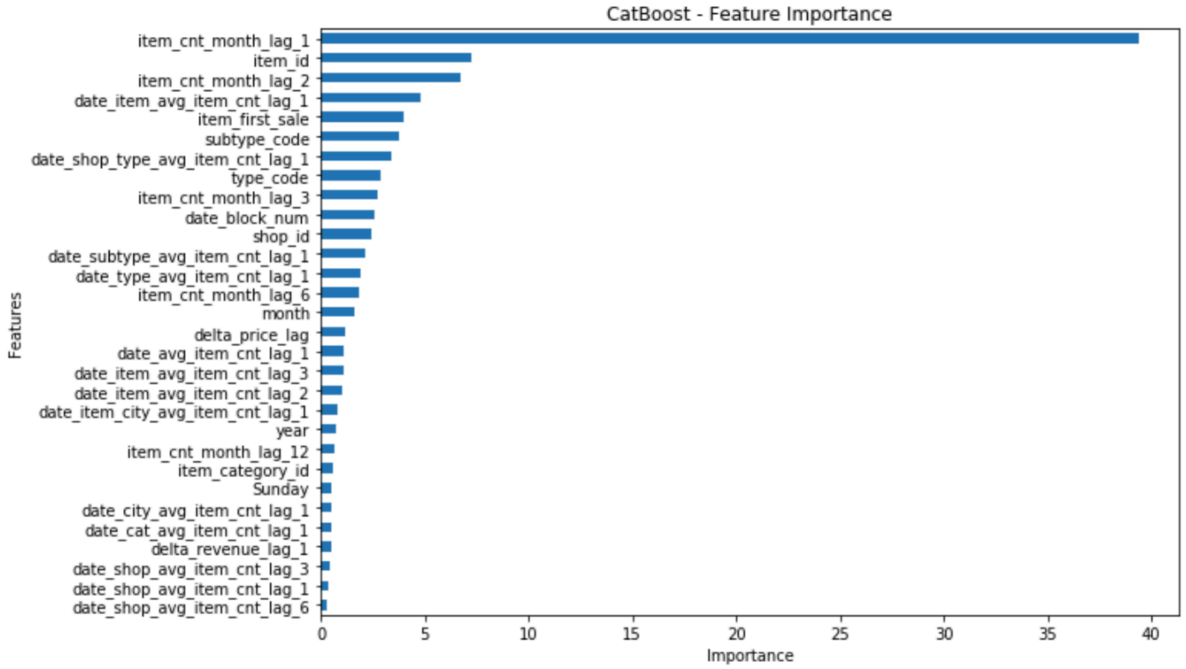


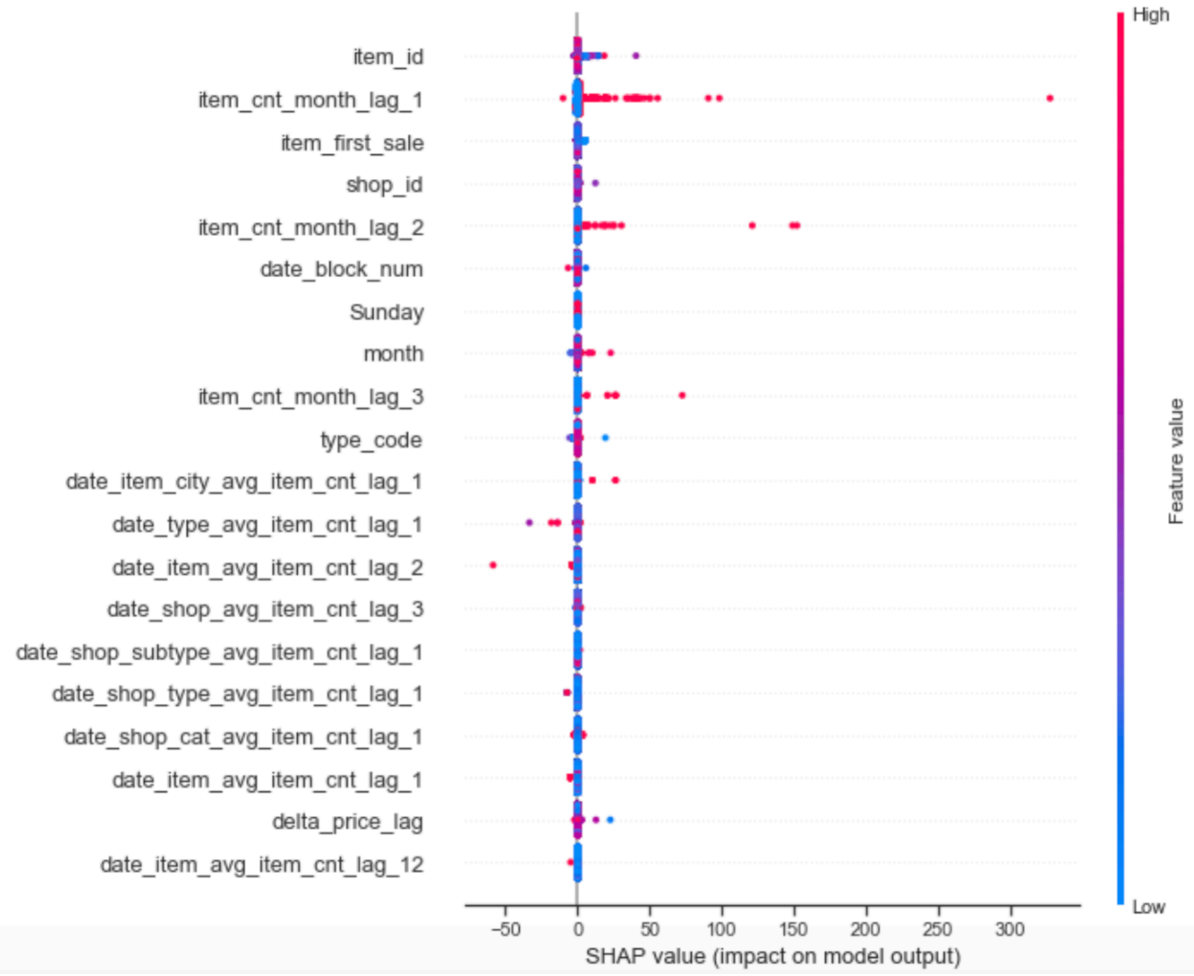
The RMSE is 3.36611. We can see that overall the model has succeeded to predict the outco mes but when zooming for the 100 also, the results are not brilliant.





Last month item sold is by far the most important feature





5 Multi Layer Perceptron Model

The first big advantage that Catboost has over Neural Network it is its ability to work with categorical data in a much easier way. Also, for Catboost it is unnecessary to scale your data since the base learners are trees, and any monotonic function of any feature variable will have no effect on how the trees are formed. This saved a lot of time the in the preparation process.

We ran a very simple model it took 10 hours to run while the Catboost took 1h40

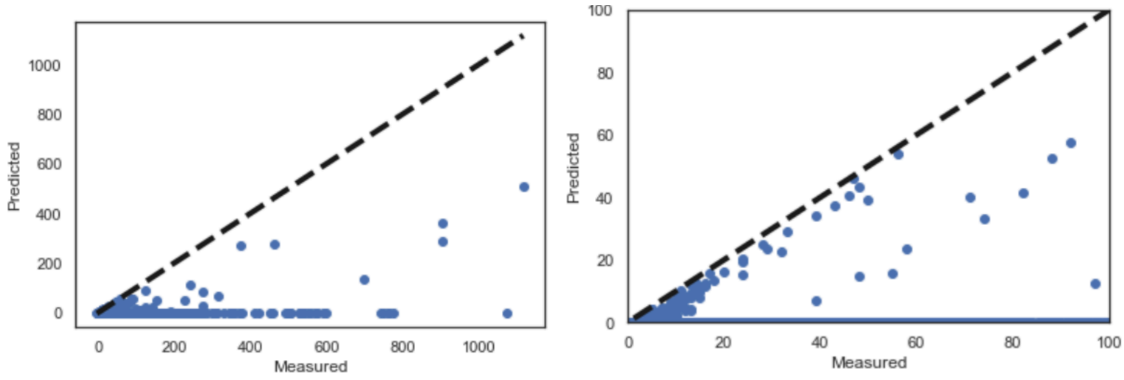
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 4)                 1180
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 10
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 3
=================================================================
Total params: 1,193
Trainable params: 1,193
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)

history = mlp.fit(trainAttrX, TrainY,
          validation_data=(testAttrX, testY),
          epochs=7, batch_size=10000, callbacks=[es])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 8346003 samples, validate on 2782001 samples
Epoch 1/7
8346003/8346003 [==============================] - 7455s 893us/step - loss: 8683334975.8897 - val_loss: 30339082.2436
Epoch 2/7
8346003/8346003 [==============================] - 6518s 781us/step - loss: 52243272.7924 - val_loss: 16006169.4421
Epoch 3/7
8346003/8346003 [==============================] - 6293s 754us/step - loss: 40044086.5957 - val_loss: 12293692.4879
Epoch 4/7
8346003/8346003 [==============================] - 6321s 757us/step - loss: 24799754.5027 - val_loss: 20138783.1843
Epoch 5/7
8346003/8346003 [==============================] - 6368s 763us/step - loss: 13357590.7273 - val_loss: 2915601.7642
Epoch 6/7
8346003/8346003 [==============================] - 6328s 758us/step - loss: 1414569.3409 - val_loss: 161872.8061
Epoch 7/7
8346003/8346003 [==============================] - 6412s 768us/step - loss: 76002.8137 - val_loss: 100186.7771
```

The RMSE is 2.9706. This is smaller than the Catboost. We can see on chart zoomed in, the results are much less scattered than for the Catboost



6 Conclusions

CatBoost was much easier to set up with categorical and continuous data
CatBoost ran much faster than the MLP
CatBoost has a very easy way to tune Hyperparameters
CatBoost has shown limitations for such datasets
MLP has shown better performance at a much higher energy cost
MLP seems to show greater opportunities for improvement than CatBoost

On such dataframe where more than 200k combinations have to be predicted, a Neural Network seems to be a better choice although it requires much more energy to run

Next step: Run the MLP on a much more powerful machine