

Machine Learning Engineer Nanodegree

Capstone Project Report

Chu, Chun Yin Alexander

January 2018

I. Definition

Project Overview

The project is focus on financial data forecasting using the time series data set given by two sigma investments¹. By comparing my method to the rest of the data scientists in the world. Stock market prediction has long been an attractive area to people. The successful prediction of a stock's future price could yield significant profit. If this problem solved can help the market much more efficient in reflecting on informations. Also, that will improve the capital flow, to the most market demanded sector.

This dataset² contains anonymized features pertaining to a time-varying value for a financial instrument. Each instrument has an id. Time is represented by the 'timestamp' feature and the variable to predict is 'y'.

¹ "Two Sigma Financial Modeling Challenge | Kaggle." <https://www.kaggle.com/c/two-sigma-financial-modeling>. Accessed 14 Jan. 2018.

² "Two Sigma Financial Modeling Challenge | Kaggle." <https://www.kaggle.com/c/two-sigma-financial-modeling/data>. Accessed 14 Jan. 2018.

Problem Statement

Problem statement — Predict the change in price of some stock using data like fundamental, technical data and the data derived from those two.

Solution statement —Using XGBoost with preprocessing the data, time series split the data into train and test set. I plan training from timestamp 0 to 905 and validate from timestamp 906 to 1812

Metrics

The model performance³ will be evaluated on the R value between the predicted and actual values. The R value similar to the R squared value, also called the coefficient of determination.

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \mu)^2}.$$

To calculate R:

$$R = \text{sign}(R^2) \sqrt{|R^2|},$$

late R,

where y is the actual value, μ is the mean of the actual values, and \hat{y} is the predicted value. (in finance, given the high ratio of signal-to-noise, even a small R can deliver meaningful value!)

Negative R values are clipped at -1, i.e. the score you see will be $\max(-1, R)$.

I think the first reason why this competition would use R rather than R^2 or mean square error (MSE) since the error change from square to normal and linear. Second, R^2 is comparing the predicted sum of squared error with the sum of squared deviations from the true target. MES do

³ "Two Sigma Financial Modeling Challenge | Kaggle."

<https://www.kaggle.com/c/two-sigma-financial-modeling#evaluation>. Accessed 14 Jan. 2018.

not compare with the deviation but the number of data point. So, R^2 is comparing how good you are doing different from you keep guessing with the mean.

II. Analysis

Data Exploration and Exploratory Visualization

The training data is using the .h5 file format instead of the standard .csv format to achieve faster read speeds. The training set file is available for download and offline modeling outside of Kernels. The test set is not available for download.

Here is the few rows and columns. ⁴

	id	timestamp	derived_0	derived_1	derived_2	derived_3
0	10	0	0.370326	-0.006316	0.222831	-0.21303
1	11	0	0.014765	-0.038064	-0.017425	0.320652
2	12	0	-0.010622	-0.050577	3.379575	-0.157525
3	25	0	NaN	NaN	NaN	NaN
4	26	0	0.176693	-0.025284	-0.05768	0.0151

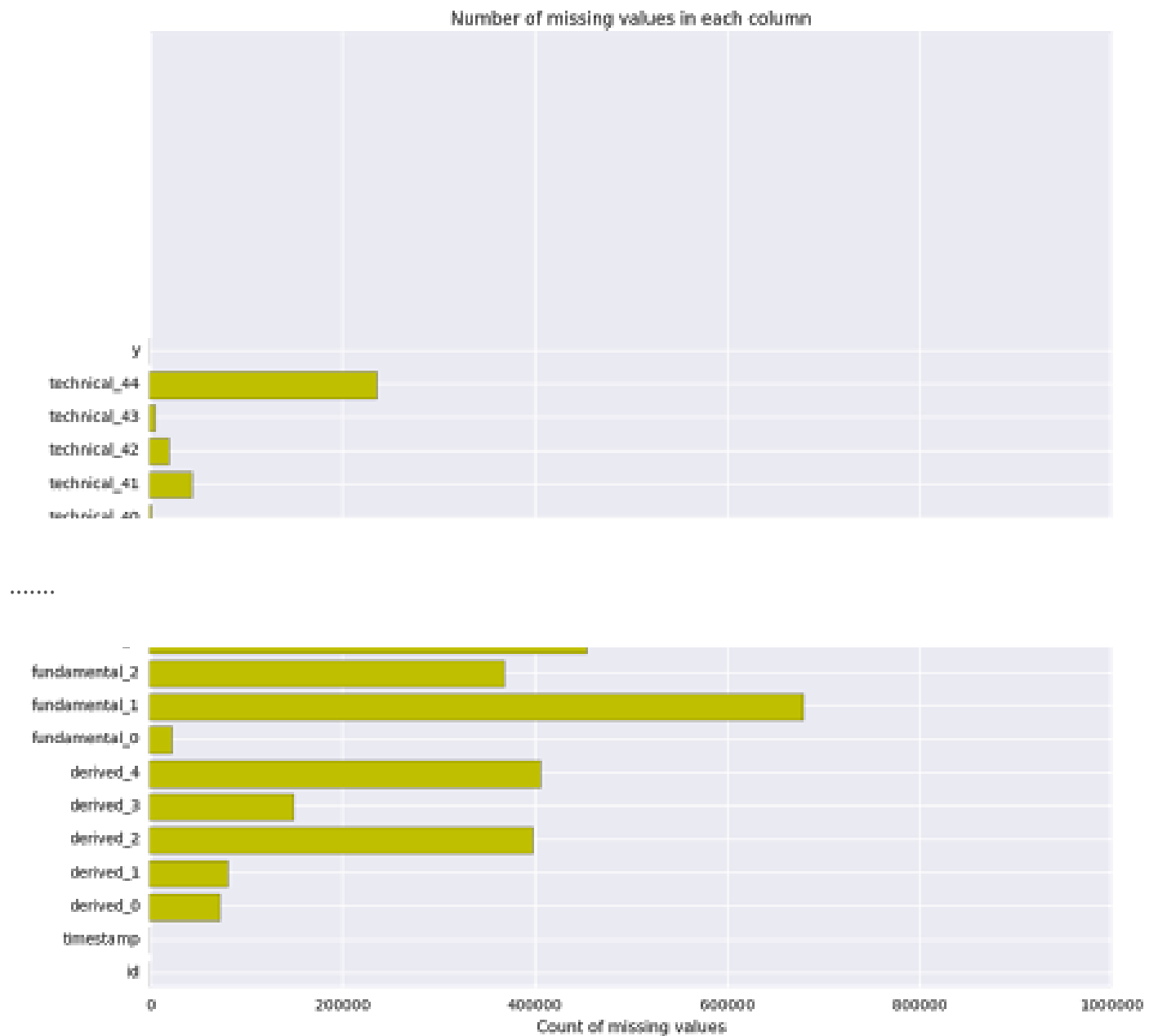
So there are 111 columns present in the dataset.

- 1 id column
 - From 0 to 2158
- 1 timestamp column
 - From 0 to 1812
- 5 columns with name prefix 'derived'
- 63 columns with name prefix 'fundamental' - fundamental_0 to fundamental_63 - 'fundamental_4' is missing.
- 40 columns with name prefix 'technical' - technical_0 to technical_44 - technical_4, technical_8, technical_15, technical_23, technical_26 are missing.

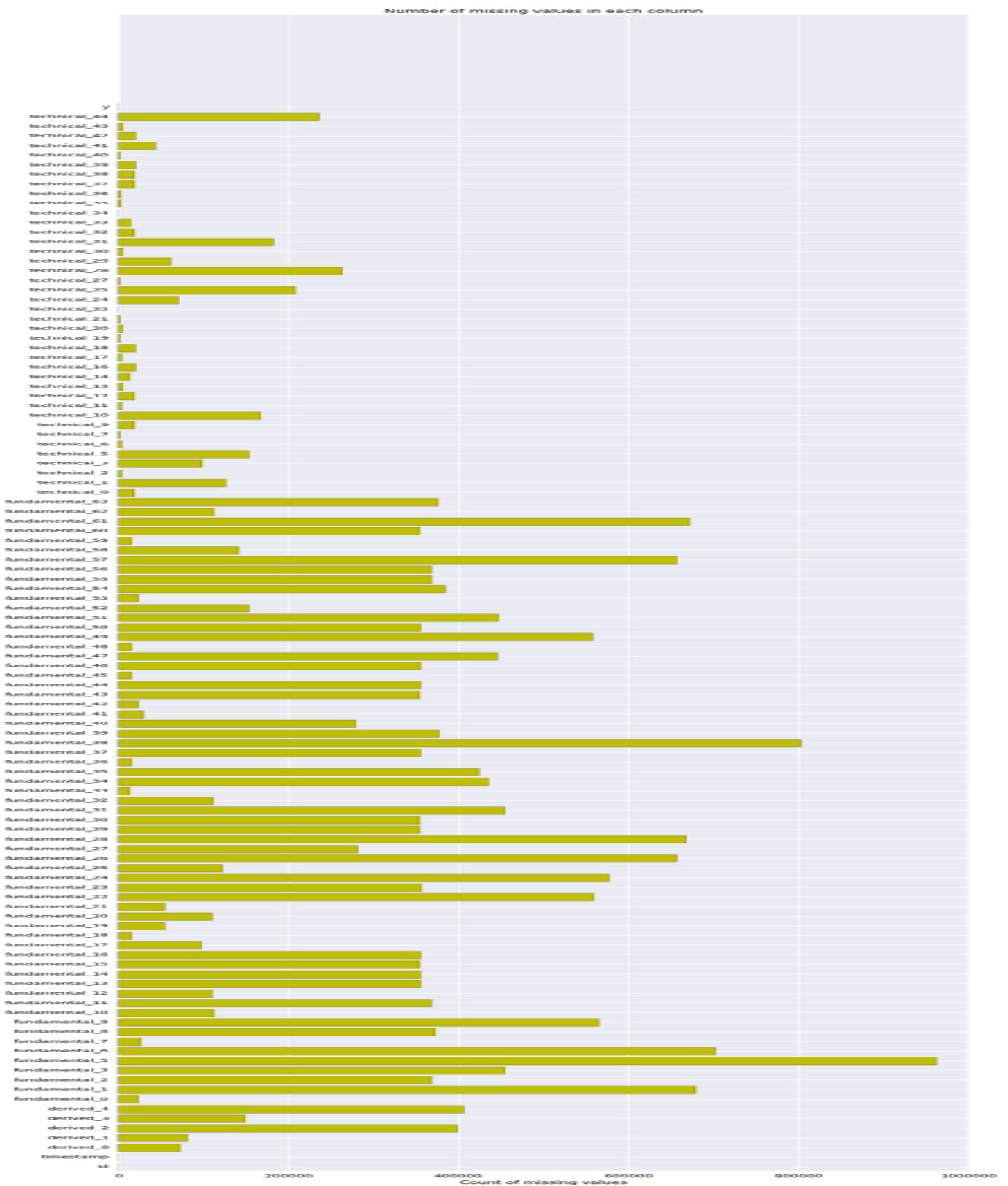
⁴ "Simple Exploration Notebook | Kaggle." 9 Dec. 2016, <https://www.kaggle.com/sudalairajkumar/simple-exploration-notebook-5>. Accessed 14 Jan. 2018.

- 1 target variable named 'y'

Here is the missing values in each column. It shown that the missing data are more often in fundamental columns.



Here is the overview.



Algorithms and Techniques

I choose XGBoost to be the starting algorithms because of their reputation of accuracy in the kaggle community. Also, by further combining other algorithms to see whether that can improve the performance in cross validation.

XGBoost is short for "Extreme Gradient Boosting", where the term "Gradient Boosting" is proposed in the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman. XGBoost is based on this original model.⁵ Boosting is a two-step approach, where one first uses subsets of the original data to produce a series of averagely performing models and then "boosts" their performance by combining them together using a particular cost function (=majority vote). Unlike bagging, in the classical boosting the subset creation is not random and depends upon the performance of the previous models: every new subsets contains the elements that were (likely to be) misclassified by previous models.













The base model is using tree base learner to train.

Benchmark








I choose the bronze medal level for the benchmark because a bronze medal level means the performance of the model is in top 10% among all participant, which I consider is great performance.

Private leaderboard (63% of the testing data)

⁵ "Introduction to Boosted Trees — xgboost 0.6 documentation."
<http://xgboost.readthedocs.io/en/latest/model.html>. Accessed 14 Jan. 2018.

199	▼ 57	Gorodec		0.0187546	46	10mo
200	▼ 69	chw		0.0187289	21	10mo
201	▼ 145	vecxoz		0.0187234	54	10mo
202	▲ 370	daten-kieker		0.0187102	3	1y
203	▼ 128	Leandro Coelho		0.0186969	146	10mo
204	▼ 13	Bojan Tunguz		0.0186817	31	10mo
205	▲ 55	SorenMindermann		0.0186715	17	10mo
206	▲ 69	stella 2		0.0186651	7	1y
207	▼ 110	RunningWolf		0.0186609	22	10mo
208	▲ 8	Zonemercy		0.0186601	18	10mo
209	▲ 34	CalmeToi		0.0186518	11	10mo
210	▲ 175	Mads		0.0186391	36	10mo

Public Leaderboard (37% of the testing data)

203	▼ 18	HelloWorld		0.0149692	11	1y
204	▼ 106	Steven		0.0149691	3	1y
205	▼ 191	Nissim Matatov		0.0149679	17	10mo
206	▼ 16	MikeS		0.0149671	9	10mo
207	▼ 191	Grzegorz Sionkowski		0.0149669	37	10mo
208	▼ 47	mgchbot		0.0149669	5	10mo
209	▼ 19	sswt		0.0149668	33	10mo

III. Methodology

Data Preprocessing

First, it is needed to read the .h5 file into a pandas dataframe in order to process it.

```
with pd.HDFStore("train.h5", "r") as train:

    # Note that the "train" dataframe is the only dataframe in the file

    df = train.get("train")
```

It is also needed to split the data into training and testing data set by the timestamp 0 to 905 and 906 to 1812

```
train = df.loc[df['timestamp'] < 906]
```

```
test = df.loc[df['timestamp'] >= 906]
```

For testing data set it is also needed to split the target and the feature for validation

```
test_features = test.drop('y', axis=1)
```

```
test_target = test[['id', 'y']].copy()
```

Since the stock data are oftentimes have some fluctuation in some time interval, I first need to calculate the standard deviation with a moving time windows which is using 10 timestamp.

```
excl = ['id', 'sample', 'y', 'timestamp']
```

```
cols = [c for c in train.columns if c not in excl]
```

```
roll_std = train.groupby('timestamp').y.mean().rolling(window=10).std().fillna(0)
```

I only use the data with standard deviation less than 0.009 to train our model.

```
train_idx = train.timestamp.isin(roll_std[roll_std < 0.009].index)
```

```
y_train = train['y'][train_idx]
```

After that, convert the pandas dataframe, both features and target, to xgboost own data structure. DMatrix is an internal data structure that is used by XGBoost which is optimized for both memory efficiency and training speed.⁶

```
xgmat_train = xgb.DMatrix(train.loc[train_idx, cols], label=y_train)
```

Notice that I did not do anything to the missing values in our data that input into the model because xgboost has taken care of that. This can be viewed as automatically "learn" what is the

⁶ "Python API Reference — xgboost 0.6 documentation."
http://xgboost.readthedocs.io/en/latest/python/python_api.html. Accessed 15 Jan. 2018.

best imputation value for missing values based on reduction on training loss.⁷ This may enhance the noise in the data that why the process of removing outlier is important in this model.

Implementation

I implemented the score to calculate the r instead of the normal r2_score that directly import from Scikit-learn. Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy⁸. So, there are a lot of metrics in Scikit-learn that you can use to access the

```
def score(y_true, y_pred):  
  
    r2 = r2_score(y_true, y_pred)  
  
    return np.sign(r2) * abs(r2)**(1/2.0)
```

Fro XGBoost model, there are a lot of parameter for tuning the model. Here is the one I used in this model.

```
params_xgb = {'min_child_weight' : y_train.size*0.0001,  
  
              'base_score'      : y_train.mean(),  
  
}
```

min_child_weight	min_child_weight [default=1] minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to
------------------	--

⁷ "What are the ways of treating missing values in XGboost? · Issue #21" 12 Aug. 2014, <https://github.com/dmlc/xgboost/issues/21>. Accessed 15 Jan. 2018.

⁸ "scikit-learn - Wikipedia." <https://en.wikipedia.org/wiki/Scikit-learn>. Accessed 29 Jan. 2018.

	<p>minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be.</p> <p>range: [0,∞]</p>
base_score	<p>base_score [default=0.5]</p> <p>This is the initial prediction score of all instances, global bias. So, I set it to the mean of the target value 'y'</p>

Then, pass it to train the booster.

```
booster = xgb.train(params_xgb, xgmat_train, verbose_eval=False)
```

For testing, I first transform the data to DMatrix format.

```
xgmat_test = xgb.DMatrix(test_features[cols])
```

Passing it to the booster to predict.

```
bst.predict(xgmat_test)
```

I calculate the score to see the result.

Refinement

The model in the above is not good. The score is negative. So, I need to fine tune the model.

There are many change. For example, I no longer use only a single xgboost to predict the target.

Fro XGBoost model, there are a lot of parameter for tuning the model. Here is the one I used in this model.

```
params_xgb = {'objective'      : 'reg:linear',
              'tree_method'    : 'hist',
```

```

'grow_policy'      : 'depthwise',

'eta'              : 0.05,

'subsample'        : 0.6,

'max_depth'        : 12,

'min_child_weight' : y_train.size*0.0001,

'colsample_bytree' : 1,

'base_score'       : y_train.mean(),

}

```

Objective	Objective is actually the default one, linear regression.
tree_method	Since the time in training in 'auto' mode is long. The tree_method choose 'hist', it represent fast histogram optimized approximate greedy algorithm. It uses some performance improvements such as bins caching.
Grow_policy	This is also default one. There are two choices: {'depthwise', 'lossguide'}. 'depthwise': split at nodes closest to the root. 'lossguide': split at nodes with highest loss change.
eta (alias: learning_rate) default=0.3	This change to 0.05, the lower eta have better convergence in this dataset. Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative.
subsample [default=1], range: (0,1]	subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting.

max_depth [default=6] , range: [0,∞]	maximum depth of a tree, increase this value will make the model more complex / likely to be overfitting. 0 indicates no limit, limit is required for depth-wise grow policy. I set this to 10 since this dataset is more complex.
min_child_weight [default=1] range: [0,∞]	<p>minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be.</p> <p>Since y_train.size = 764338, I start with the that, found increasing performance with decreasing this weight. Finally, stop at dividing 2000.</p>
colsample_bytree [default=1] range: (0,1]	subsample ratio of columns when constructing each tree. I set this to 1, using all the columns each time.
base_score	<p>base_score [default=0.5]</p> <p>This is the initial prediction score of all instances, global bias. So, I set it to the mean of the target value 'y'</p>

Here is the grid search for the max_depth and min_child_weight, sorted by higher score first. The model train on the first half of the timestamp, obtaining score by testing on the later half of the timestamp.

max_depth	min_child_weight	score
12	76.4	0.0291403359650
12	114.6	0.0289330944120
13	57.3	0.0288087085710
13	95.5	0.0286928123980
14	191.0	0.0286456985960
12	38.2	0.0286161684020
14	114.6	0.0285954288730
11	57.3	0.0284960215720
12	191.0	0.0283643263510
14	38.2	0.0281680332190
10	191.0	0.0281640239422
11	95.5	0.0280435498290
14	76.4	0.0277361700200
10	382.0	0.0267445617762
9	382.0	0.0265443582685
10	764.0	0.0262169736716
9	191.0	0.0261893564036
9	764.0	0.0257429580869
7	191.0	0.0251050824763
7	382.0	0.0249542334161
7	764.0	0.0245867093522
10	1,910.0	0.0240942719891
9	1,910.0	0.0239349466253
5	191.0	0.0230011081001
5	382.0	0.0227983605920
7	1,910.0	0.0227404275084
5	764.0	0.0226831081760
5	1,910.0	0.0217024504051
3	764.0	0.0202620149491

3	382.0	0.0198712312770
3	191.0	0.0198364273703
3	1,910.0	0.0187761180331

Beside the parameters, a stand alone argument that pass to train api is num_boost_round, which is number of boosting rounds or trees to build. I use 8 booster and average them to further generalise it.

```
num_boost_round = 16

bst_lst = []

for i in range(8):

    params_xgb['seed'] = 623913 + 239467 * i

    bst_lst.append(xgb.train(params_xgb,

                             xgmat_train,

                             num_boost_round=num_boost_round,

                             verbose_eval=False).__copy__())
```

Finally, get the score to see its performance.

```
pr_lst = []

xgmat_test = xgb.DMatrix(test_features[cols])

for bst in bst_lst:

    pr_lst.append(bst.predict(xgmat_test))

pred = test_target.copy()

pred['y'] = np.array(pr_lst).mean(0)

print score(test_target['y'], pred['y'])
```

IV. Results

Model Evaluation and Validation

The final model reasonable and aligning with solution expectations. It is able to achieve weighted average R = 0.02705289531876, better than the last team of the bronze level which the score is 0.0186609.

weighted average = (average score of the 3-fold * 1/3 + score of using first 2 the 3-fold * 2/3 + score of using 2-fold * 1/2)/(1/3+2/3+1/2)

The 0,1,2 represent the dataset splitted into 3-fold by timestamp. The last one is the 2 fold score.

		score		
train	test	score	average of the 3-fold	weighted average
0	1	-0.00609203951406		
0	2	0.02678628853970		
1	2	0.02324283535370	0.01464569479311	
0, 1	2	0.03169091509690		
first half	second half	0.02914033596500		
				0.02705289531876

The final parameters of the model is appropriate. The final model been tested with last haft of timestamp to evaluate whether the model generalizes well to unseen data. The model is robust enough for the problem.

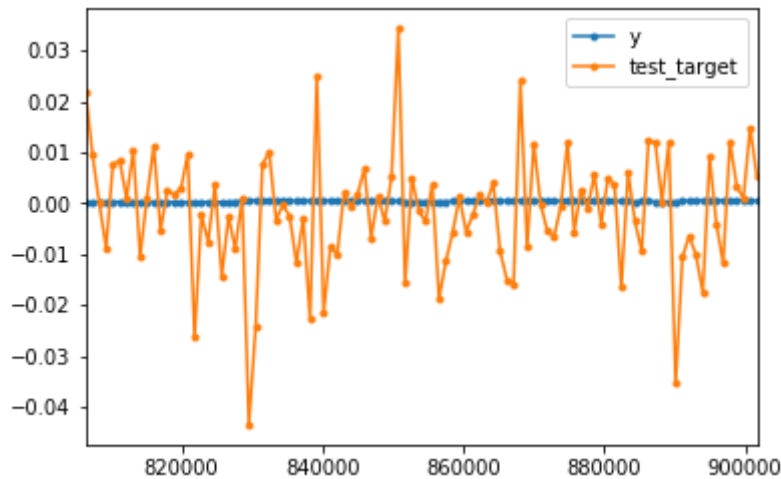
Justification

The final results found is stronger than the benchmark result reported earlier. In finance, given the high ratio of signal-to-noise, even a small R can deliver meaningful value. Thus this model is a significant result which R = 0.02705289531876.

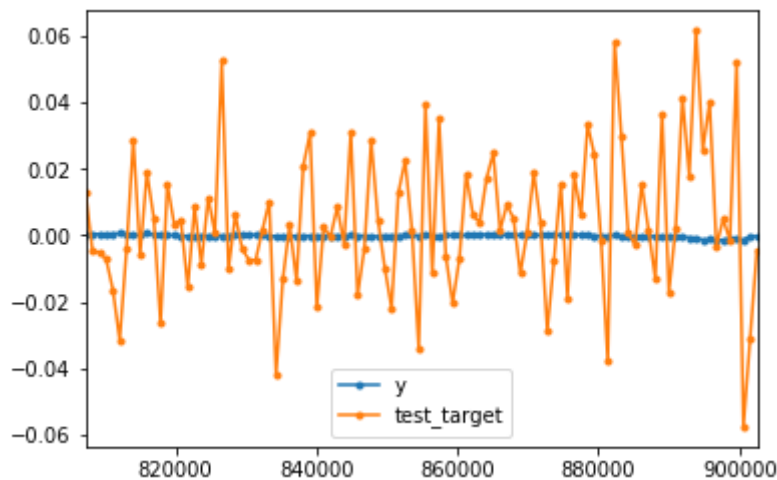
V. Conclusion

Free-Form Visualization

Here is the predicted 'y' and the 'test_target' of id = 112 across the testing set. The vertical axis is the y and test_target value, horizontal one is the pandas index which can treat as timestamp.



Here is the predicted 'y' and the 'test_target' of id = 2047 across the testing set.



From the above two pictures, you can see there is large fluctuation in the true target. That explain why the R value is low. The picture is aimed to visualize how the model performance with this corresponding R value.

Reflection

1. Preprocessing

- a. Visualize the data to get a general understanding on how the data looks like.
 - b. See whether it is skew or some data is missing needed to deal with.
 - c. Test-train split the data
 - i. Both 3-fold and 2-fold is done
2. Modeling
- a. Tuning parameter for the better model
 - b. Keep attention on overfitting

In the k-fold validation, i had difficult in using the grid search scikit-learn api. Since i'm dealing with time series. The future data can't mix together with the past data. So, in the end, I am using two for-loop to do two parameter grid search and only using half split to the dataset.

It is interesting when plotting the final prediction verses the true test target, because the fluctuation is really large and the model is trying to keep between the mean. It is difficult when thinking how to deal with the missing data. The final model and solution fit my expectations of finance world. The process I used during this project can generally applied to time series forecasting.

Improvement

There is always room to improve, for example, after reading the winner's interview ⁹ ¹⁰, I think the lagging feature, momentum feature and ensemble different model can be tested in the future. If you used your final solution as the new benchmark, do you think an even better solution exists?

The model that win the kaggle competition is certainly a better solution. One thing I learn in this project is that, the last few percent of increase in performance take many times the original effort of putting the first model.

⁹ "Two Sigma Financial Modeling Challenge, Winner's Interview: 2nd" 25 May. 2017, <http://blog.kaggle.com/2017/05/25/two-sigma-financial-modeling-challenge-winners-interview-2nd-place-nima-s-hahbazi-chahhou-mohamed/>. Accessed 26 Jan. 2018.

¹⁰ "Two Sigma Financial Modeling Code Competition, 5th Place Winners" 11 May. 2017, <http://blog.kaggle.com/2017/05/11/two-sigma-financial-modeling-code-competition-5th-place-winners-interview-team-best-fitting-bestfitting-zero-circlecircle/>. Accessed 26 Jan. 2018.