Data Analysis

Code Github address: https://github.com/piccogun/monthly_sales_forecast

Data Preprocess

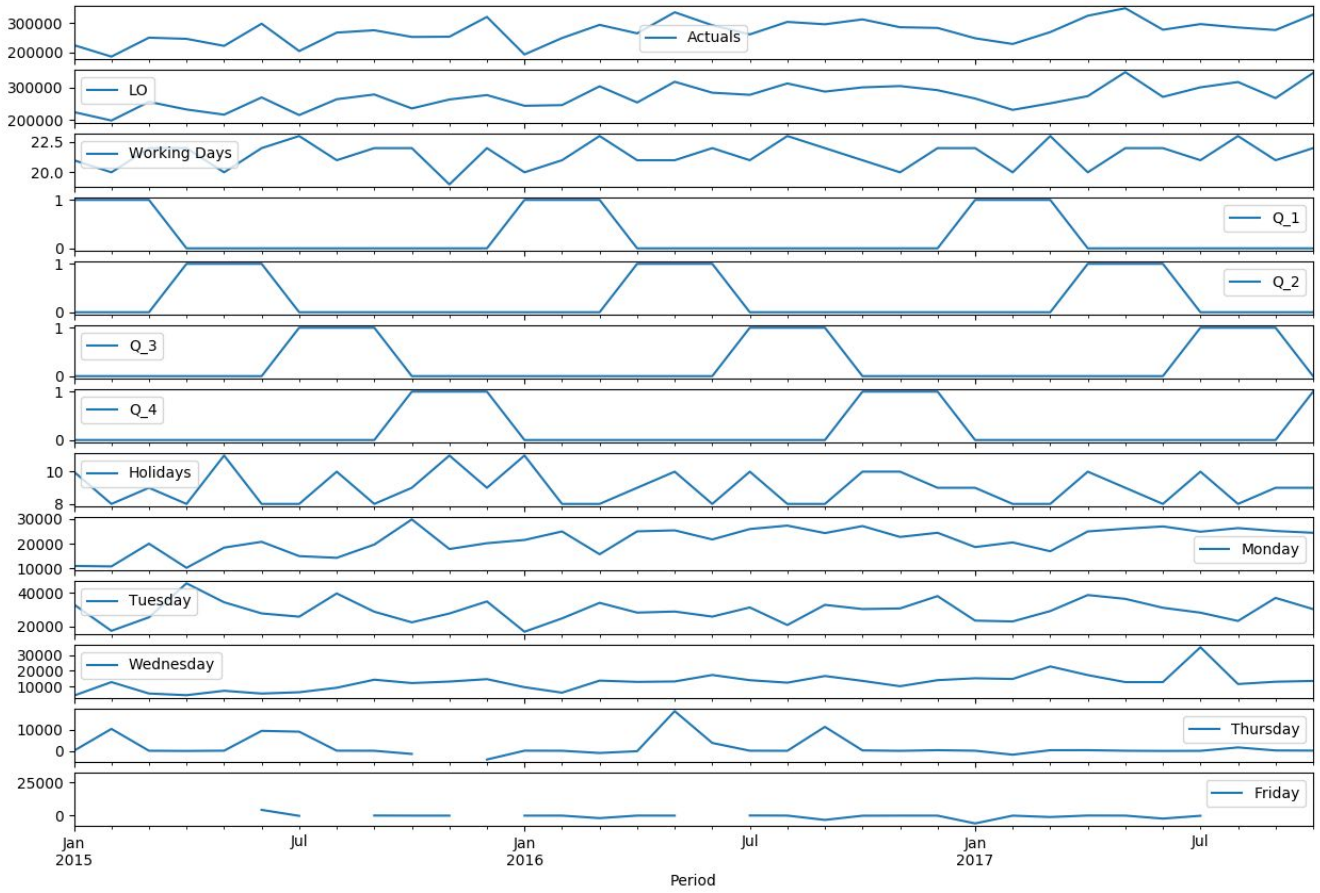
Raw Data

- y_t: full month actuals, t is in monthly basis
- raw features: actuals history (monthly), daily sales, LO (monthly), weekdays calendar, holidays
- window: monthly rolling (ToDo: to expand the window, i.e. including more months)
- Some problems found in the original data:
    - some daily sales are not NaN during holidays, for example, 31-01-2015 is stated as Saturday in the Calendar but still have sales for 14.2.
    - some working day daily sales are missing (even in the whole month), for example, 6, 13, 20, 27 February 2015 are all working Friday, but the daily sales data is missing.
    - Solution: In business scenario it needs to communicate with the data vendor and to add a new calendar such as shop event days. But in this case, we believe that the data is cleaned properly. So we ignore the NaNs, as xgboost model can handle missing data.

Feature Extraction

- **daily sales**: to unify the time dimension (the same frequency), daily data needs to be transformed into monthly data. Considering working days may have influence in the shopping behaviors, one way to transform the daily sales into monthly feature is to calculate average sales for each working day in a month. As a result, we get monthly Monday average sales, Tuesday average sales .etc., which also took feature of ‘weekdays’ into account. (We use average sales not total sales for each working day because for forecasting not all the daily sales are available in the month.)
- **LO**: use as a feature directly
- **working days**: use as a feature directly
- **holidays**: use as a feature directly
- **seasonal or trend**: set four quarters in the year as dummy variables
- **other features**: self-lag and above features with lagged value (ToDo)
- Alternative time series for To Do: given that daily series may also have an influence on the final actual in the month, the time series may be formed on a daily basis from 15th to 25th for each month.

Data Observation



The LO looks closely related to the Actuals, and there may be seasonal trend. Thus we include quarterly dummy variables as features.

Data Transformation

The model we used is based on tree structures. The splits of the tree only depend on the threshold, so the normalization of features is not needed. Also it is immune from many statistical assumptions, such as uncorrelated features or stationary targets (some additional features made from linear combinations of the original features can even produce better results). So we don't perform any transformation for the moment.

why it doesn't care about scaling of features
https://stats.stackexchange.com/questions/353462/what-are-the-implications-of-scaling-the-features-to-xgboost
math explanation
https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

Bootstrap Resampling

There are two layers of resampling

1. Randomly split the whole training data into a training set and validation for n iterations, to produce n predictions for n validation sets. By these predictions we find the 95 % confidence interval for each validation date (all the time periods are validated). It will be discussed more in the section 'Model Development' - 'Evaluation and Confidence Interval' in this document. For a 10 iteration example this looks like

prediction_results

Period	iterate_0	iterate_1	iterate_2	iterate_3	iterate_4	iterate_5	iterate_6	iterate_7	iterate_8	iterate_9	CI_up	CI_low	In_CI
2015-01			229367.4								229367.40625	229367.40625	0
2015-02	215567.56		231047.19								230660.196875	215954.553125	0
2015-03				222309.11			226367.11				226265.65937500002	222410.55937499998	0
2015-04	241730.3										241730.296875	241730.296875	0
2015-05				232498.72		254211.23			217087.67		253125.60859375	217858.22421875	1
2015-06				239733.81			249794.45	243561.48			249482.8046875	239925.19609374998	0
2015-07				224969.52		235716.73		214898.38			235179.3734375	215401.93203124998	0
2015-08			256502.08				257429.58	246909.9		272250.84	271139.248828125	247629.319140625	1

2. Use randomized grid search to tune parameters on xgboost. Since there are many hyperparameters in xgboost, one need to choose the optimal set from a list made by customized possible hyperparameter values, as shown below. The RandomizedSearchCV from sklearn package samples uniformly to pick the combination of the hyperparameters, cross validation folds is defined as 3. More discussions can be found in the latter part of this document.

```
params_dic = {
  'silent': [1], # not showing running messages
  # learning task parameters
  'learning_rate': [0.001, 0.01, 0.1, 0.2, 0,3],
  'base_score': [target_mean], # initialize prediction score (global bias) to make sure the trees 'catching up' faster
  # tree based parameters
  'max_depth': [3, 6, 10, 15, 20],
  'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
  'colsample_bytree': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
  'min_child_weight': [0.5, 1.0, 3.0, 5.0, 7.0, 10.0],
  'gamma': [0, 0.25, 0.5, 1.0],
  # regularization parameters
  'alpha': [0.0001, 0.001, 0.1, 1.0, 5.0, 10.0, 50.0]
```

Model Development

The task is to find a model f so that

$$y_t = f(MonS_t, TueS_t, WedS_t, ThurS_t, FriS_t, LO_t, WD_t, HD_t, Dummy_t)$$

ToDo: include lag-values

Model Choice

- Usually one should try out different algorithms and make comparisons. Due to the strict project deadline and requirement on the accuracy of the result, I use the most robust model based on my personal experience.
- We choose **XGBoost models**, as it is immune from statistical assumptions (linearly combined features sometimes even show more importance), easy to interpret and robust enough to model the multivariate time series with correlated features.
- We do not use the **neural network** models as they may look like a black box because of the hidden layers so that it is harder to interpret. It is suitable for big data set with various types of information, such as text, image..
- Another choice would be **Vector AutoRegressive model (VAR)**, which linearly captures dynamics between time series. Modeling VAR would firstly require checking for stationarity and to make adjustments accordingly, such as differencing. It would be interesting to test their performance in the future.

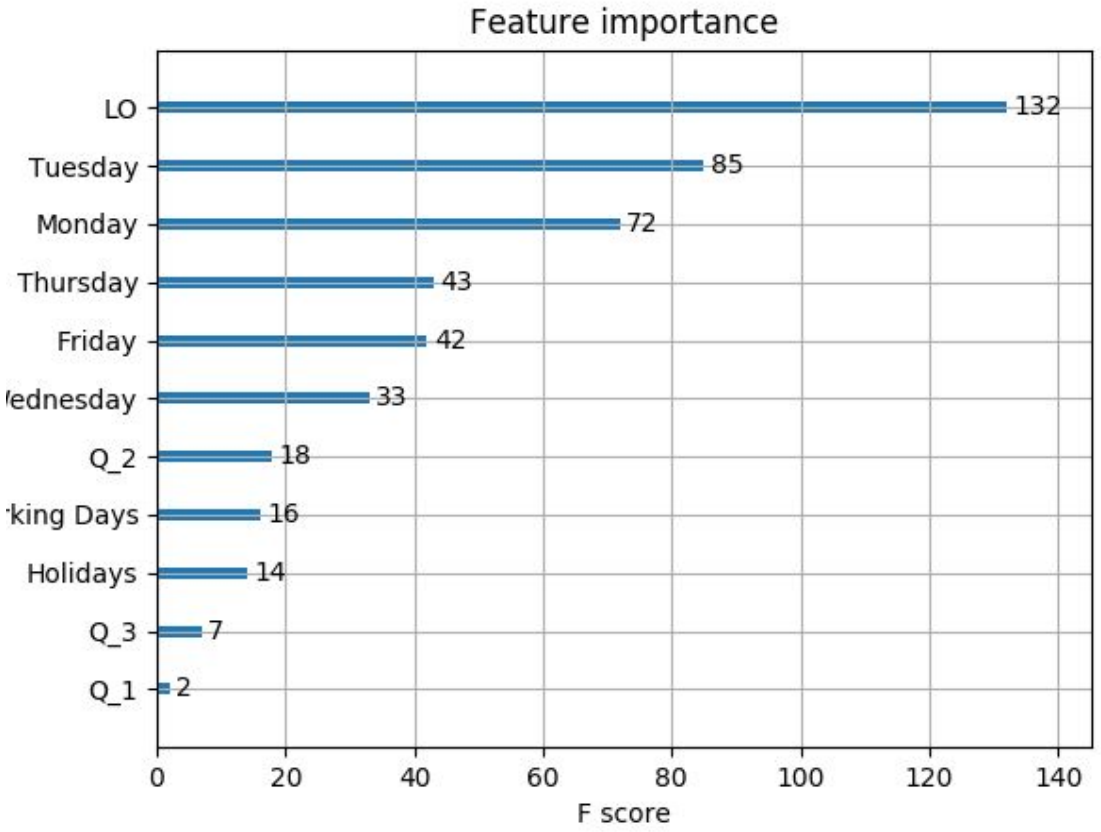
How XGBoost is comparatively better than other techniques:

1. Regularization:
    - Standard GBM implementation has no regularisation like XGBoost.
    - Thus XGBoost also helps to reduce overfitting.
2. Parallel Processing:
    - XGBoost implements parallel processing and is faster than GBM .
    - XGBoost also supports implementation on Hadoop.
3. High Flexibility:
    - XGBoost allows users to define custom optimization objectives and evaluation criteria adding a whole new dimension to the model.
4. Handling Missing Values:
    - XGBoost has an in-built routine to handle missing values.
5. Tree Pruning:
    - XGBoost makes splits up to the max_depth specified and then starts pruning the tree backwards and removes splits beyond which there is no positive gain.

reference:
https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

Further theories of xgboost can be found later.

Feature Importance



We used default xgboost regressor to give out importance score for the features. (The code is not included in the project). LO is clearly the most important feature.

The ideal computation consists in producing this kind of graph for each forecast, however due to strict deadline we will add this to the future to do list.

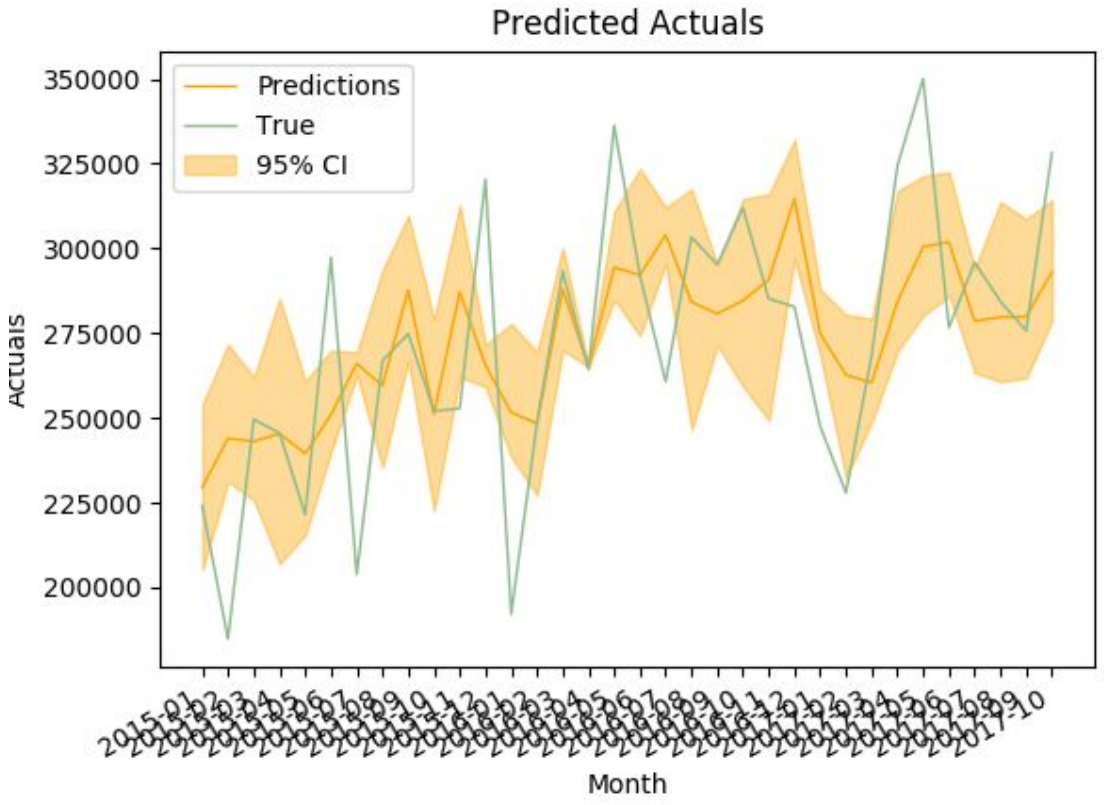
Evaluation and Confidence Interval

1. Resample 80% dataset to training set and validation set randomly for n iterations with replacement (Bootstrap), to build an empirical distribution function.
2. Predicted target values and mse of validation set and training time are computed for each iteration training.
3. We choose the model that produced lowest mse as the best_model to do the forecast at each forecast date.

4. Empirical Confidence Interval of 95% on predicted target values is calculated for each date after bootstrap iterations. If the true Actuals fall into this confidence we assign the correct_score = 1 and 0 otherwise.
5. The ‘Predictive Power’ is calculated by summing up the correct_scores then divided by total true target size.
6. Repeat the whole process for each forecasting date.

Attention: the bootstrapping iterations should be as large as possible given the time resources. The sample size is usually the same as the dataset, but we use 80% to get validation confidence interval.

The following figure is an example of 10 bootstrap iterations for a model training at a forecast date.



Forecast Result

- Some facts:
    - There are iterations on bootstrap, hyperparameter search and xgboost model training in this project.
    - The hyperparameter search is parallelizable by using multi-processors.

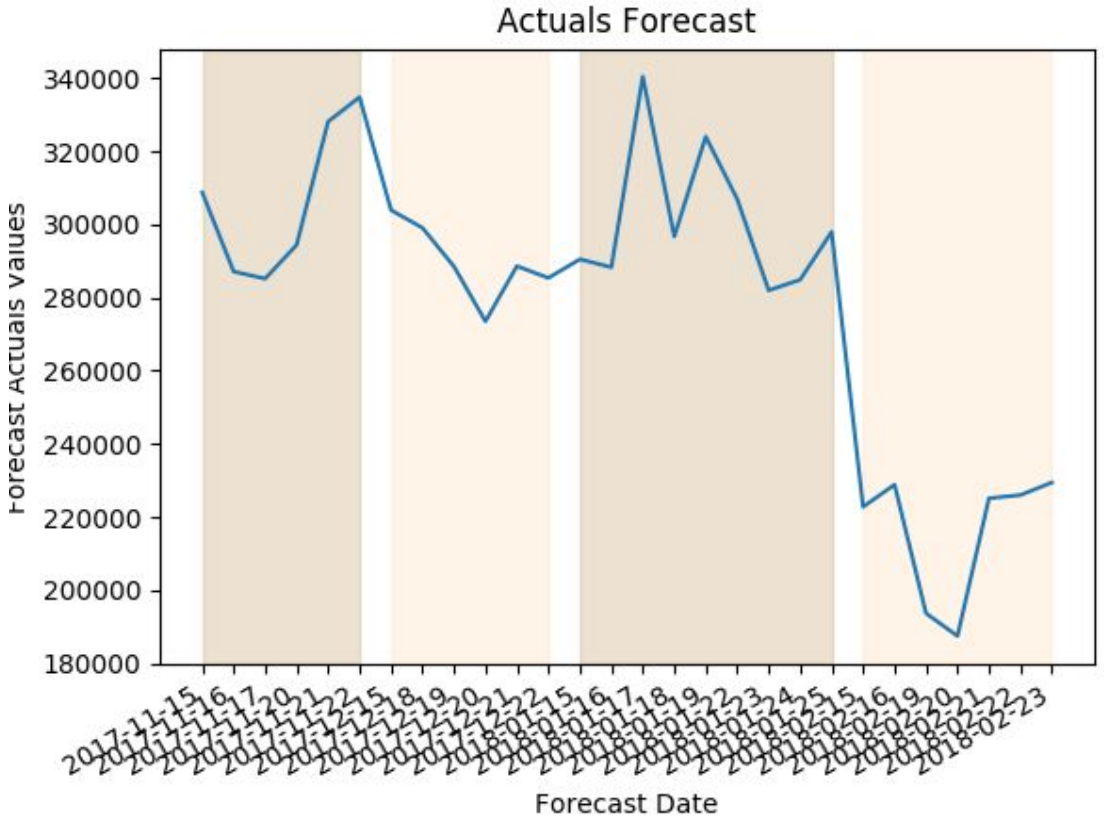
    - The xgboost model training is parallelizable by using multi-threads on multi-core computers.
    - Xgboost trains much faster on GPU than CPU.
- Empirical result:
    - More iterations of bootstrap produces higher Predictive Power (better confidence interval). As a result, with a strict deadline, more processors, more computer cores (multi-threading) and GPU is preferred if it is available.

Low Hardware Configuration

Processor: 1.8 GHz Intel Core i5
Memory: 4 GB 1600 MHz DDR3
Thread: 2
Forecast with 10 iterations:

	Forecast	Predictive Power	Running Time
2017-11-15	299503.875	0.23529411764705882	0:01:07.868623
2017-11-16	296251.96875	0.4117647058823529	0:00:55.415539
2017-11-17	293885.1875	0.11764705882352941	0:00:55.781991
2017-11-20	295879.65625	0.4411764705882353	0:00:58.085654
2017-11-21	314378.6875	0.4117647058823529	0:00:51.083503
2017-11-22	307246.84375	0.2647058823529412	0:00:55.309872
2017-12-15	308857.40625	0.4411764705882353	0:00:53.234827
2017-12-18	285500.1875	0.20588235294117646	0:00:47.634849
2017-12-19	252981.046875	0.20588235294117646	0:00:47.357811
2017-12-20	300316.375	0.3235294117647059	0:00:46.923688
2017-12-21	299590.15625	0.4117647058823529	0:00:46.466887
2017-12-22	284275.15625	0.2647058823529412	0:01:07.376650
2018-01-15	299903.1875	0.2647058823529412	0:00:51.014983
2018-01-16	297270.96875	0.2647058823529412	0:00:56.167067
2018-01-17	306162.65625	0.23529411764705882	0:00:58.401895
2018-01-18	264089.90625	0.29411764705882354	0:00:50.170285
2018-01-19	291006.0625	0.38235294117647056	0:01:00.147695
2018-01-22	296940.5	0.35294117647058826	0:00:50.395910
2018-01-23	309297.84375	0.17647058823529413	0:00:57.371277
2018-01-24	284576.5	0.23529411764705882	0:00:51.796351
2018-01-25	294218.5	0.20588235294117646	0:00:56.162757
2018-02-15	215334.15625	0.35294117647058826	0:00:56.072387
2018-02-16	219210.546875	0.2647058823529412	0:00:55.182933
2018-02-19	225092.03125	0.11764705882352941	0:00:49.096604
2018-02-20	230520.03125	0.3235294117647059	0:00:52.634272
2018-02-21	234219.453125	0.35294117647058826	0:00:57.527948
2018-02-22	216671.765625	0.29411764705882354	0:00:52.096881
2018-02-23	219357.8125	0.3235294117647059	0:00:53.294984

The predictive powers are very low, approximately in the range of 0.1 to 0.4. In total it took almost an hour to produce the result.



The vertical bars indicate forecast results for the same month.

High Hardware Configuration

Model Name:MacBook Pro
Processor Name:Intel Core i7
Processor Speed:2.6 GHz
Number of Processors:1
Total Number of Cores:6
L2 Cache (per Core):256 KB
L3 Cache:9 MB
Hyper-Threading Technology:Enabled
Memory:16 GB

The Predictive Power is relatively much higher and more stable - in average of 0.7.
The whole process took almost 5 hours 11 minutes.

forecast_result

	Forecast	Predictive Power	Running Time
2017-11-15	301834	0.7058823529411765	0:11:27.253527
2017-11-16	291318.46875	0.7058823529411765	0:11:51.794251
2017-11-17	300365.40625	0.7352941176470589	0:12:53.913913
2017-11-20	284120.40625	0.7058823529411765	0:12:43.750820
2017-11-21	297147.375	0.7058823529411765	0:12:39.471989
2017-11-22	289021.59375	0.7058823529411765	0:11:26.012278
2017-12-15	295229.65625	0.7352941176470589	0:11:19.496221
2017-12-18	294620.125	0.7352941176470589	0:11:21.569847
2017-12-19	306603.28125	0.7058823529411765	0:11:17.062916
2017-12-20	296023.46875	0.7058823529411765	0:11:19.858396
2017-12-21	297197.46875	0.7647058823529411	0:11:21.237450
2017-12-22	278442	0.7058823529411765	0:11:27.581897
2018-01-15	273268.71875	0.7352941176470589	0:11:05.986621
2018-01-16	306391.8125	0.7352941176470589	0:10:38.077130
2018-01-17	288191.875	0.7647058823529411	0:10:31.746013
2018-01-18	304546.59375	0.7058823529411765	0:10:27.217002
2018-01-19	317516.5	0.6764705882352942	0:10:23.754876
2018-01-22	289118.5	0.7058823529411765	0:10:21.194138
2018-01-23	325161.1875	0.7352941176470589	0:10:43.817722
2018-01-24	302030.3125	0.7352941176470589	0:12:10.133328
2018-01-25	323443.84375	0.7058823529411765	0:11:15.723547
2018-02-15	212593.84375	0.7058823529411765	0:10:53.345355
2018-02-16	230676.296875	0.7058823529411765	0:10:16.368490
2018-02-19	223959.59375	0.7058823529411765	0:10:16.440776
2018-02-20	210885.46875	0.7058823529411765	0:10:15.242252
2018-02-21	197953.875	0.7647058823529411	0:10:15.130274
2018-02-22	235678.265625	0.7058823529411765	0:10:15.363213
2018-02-23	221836.125	0.7058823529411765	0:10:17.635397

Key Hyperparameters

For xgboost modeling, results are sensitive to hyperparameters, which need to be tuned carefully. Here we choose Randomized Search for tuning hyperparameters, with n_iter combinations of hyperparameters uniformly sampled (if the parameters are given in lists). Ideally GridSearch provides better accuracy as it tries out all the combinations, but in cost of time and computer capacity.

Randomized Search

```
RandomizedSearchCV(xgb_model, param_distributions=params_dic, n_iter=200, cv=3, verbose=0,
                                   iid=False, n_jobs=-1)
# param_distributions: if a list is given, sample uniformly
# n_iter: not all the combinations are tested, n_iter set the number of total combinations,
# the more the better prediction but slower running time
# n_jobs: try as many processors as possible
# iid: return the average score across folds, not weighted by the number of samples in each test set
# cv: KFold cross validation
```

Xgboost hyperparameters

```
params_dic = {
  'silent': [1], # not showing running messages
  # learning task parameters
  'learning_rate': [0.001, 0.01, 0.1, 0.2, 0,3],
  'base_score': [target_mean], # initialize prediction score (global bias) to make sure the trees 'catching up' faster
  # tree based parameters
  'max_depth': [3, 6, 10, 15, 20],
  'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
  'colsample_bytree': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
  'min_child_weight': [0.5, 1.0, 3.0, 5.0, 7.0, 10.0],
  'gamma': [0, 0.25, 0.5, 1.0],
  # regularization parameters
  'alpha': [0.0001, 0.001, 0.1, 1.0, 5.0, 10.0, 50.0]
```

XGBoost Theory

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Objective Function (loss+regularization):

$$\text{obj}(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \quad , \quad \hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

K: number of trees

Additive Training

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\cdots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

Customize loss function other than just mse, by taylor expansion on second order, the specific objective function at step t (optimization goal for the new tree):

$$= \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

let

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \to \{1, 2, \cdots, T\}$$

T: number of leaves, w: vector of scores on leaves, q: assign each data point to the corresponding leaf

Regularization

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

rewrite the objective value with the t-th tree:

$$\text{obj}^{(t)} \approx \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w^2_{q(x_i)}] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T$$

$$\text{obj}^{(t)} = \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T$$

where $G_j = \sum_{i\in I_j} g_i$ and $H_j = \sum_{i\in I_j} h_i$

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

optimized objective $$\text{obj}^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

Basically, for a given tree structure, we push the statistics gi and hi to the leaves they belong to, sum the statistics together, and use the formula to calculate how good the tree is. This score is like the impurity measure in a decision tree, except that it also takes the model complexity into account.

Iterate overall all possible trees and the tree with the lowest score is our next best tree. BUT it is not practical to scan through all the possible trees. So, the loss function is modified further to find the **next best split** and now you can call it the gain.

Learn the tree structure (more splits)
optimize one level of the tree at a time. Specifically we split a leaf into two, gain score is :

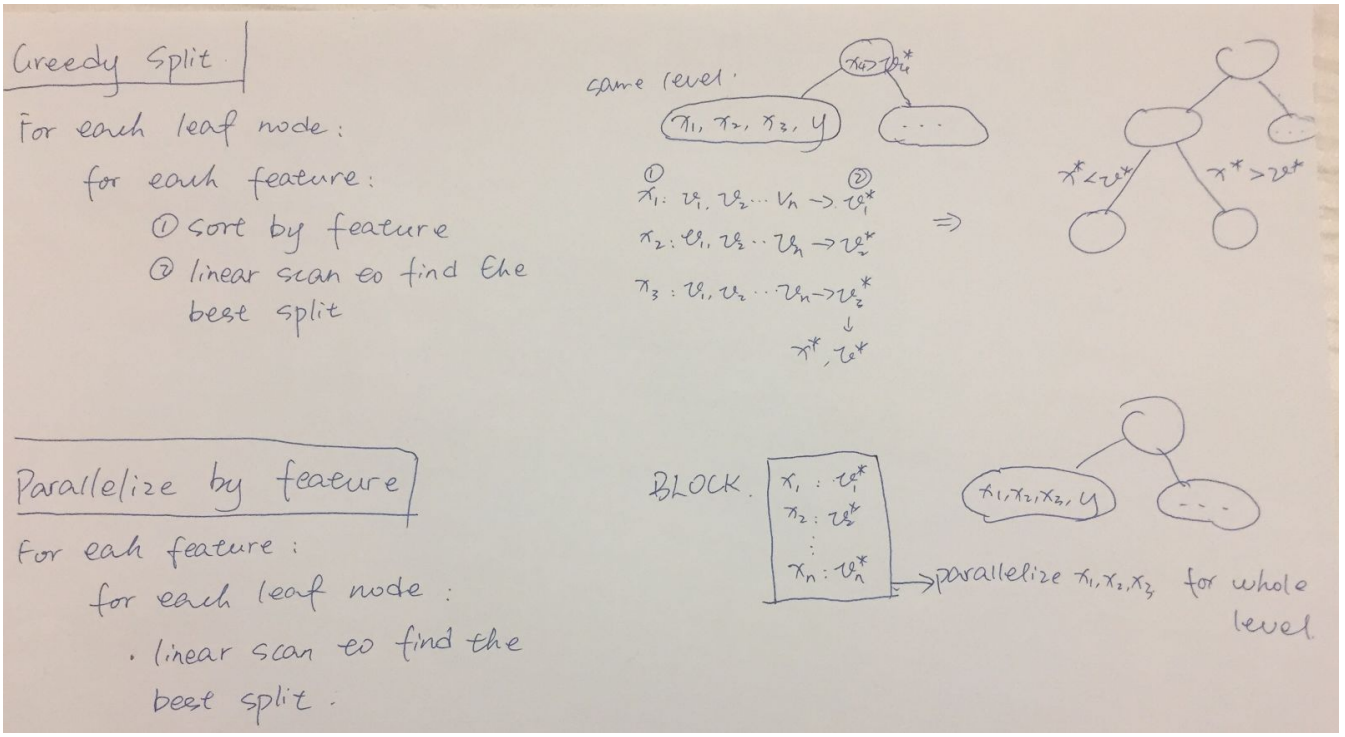
$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma$$

if the gain is smaller than γ, we would do better not to add that branch. This is exactly the **pruning** techniques in tree based models

https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

Parallelization

Parallelize Split Finding at Each Level by Features



http://zhanpengfang.github.io/418home.html

Model Parameters

reference
https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
https://xgboost.readthedocs.io/en/latest/parameter.html

1. General Parameters: Guide the overall functioning
2. Booster Parameters: Guide the individual booster (tree/regression) at each step
3. Learning Task Parameters: Guide the optimization performed

General Parameters
booster = gbtree
silent=0, 1 for not showing running messages
nthread = max number of threads if not set

Booster Parameters (for tree booster only)

	default	meaning	alias in GBM
eta	0.3	shrinking weights on each step, typically 0.01-0.2	learning_rate
base_score	0.5	Initialize prediction score of all instances, global bias. For regression problem, may make too	

		many trees just to catch up the mean(label) and less trees to solve the real task. So for classification problem 0.5 may make sense since target values are 0 or 1. For regression problems one may like to start with mean(target) to learn faster.	
min_child_weight	1	too high values may lead to under-fitting, should be tuned using cv	
max_depth	6	higher value may lead to overfitting, should be tuned using cv, typically 3-10	
max_leaf_nodes		if it is defined, max_depth is ignored	
gamma	0	min loss reduction to make a split, large value may lead to underfitting, should be tuned	min_split_loss
subsample	1	fraction of columns to be randomly samples for each tree, occurs in each boosting iteration, too small may lead to under-fitting, typically 0.5-1	
colsample_bytree	1	faction of columns to be randomly sampled for each split, in each level, typically 0.5-1	max_features
lambda	1	L2	reg_lambda
alpha	0	L1	reg_alpha

tree_method	auto	● The tree construction algorithm used in XGBoost. See description in the reference paper.
		● Distributed and external memory version only support tree_method=approx.
		● auto: Use heuristic to choose the fastest method.
		  ● For small to medium dataset, exact greedy (exact) will be used.
		  ● For very large dataset, approximate algorithm (approx) will be chosen.
		  ● Because old behavior is always use exact greedy in single machine, user will get a message when approximate algorithm is chosen to notify this choice.
		● exact: Exact greedy algorithm.
		● approx: Approximate greedy algorithm using quantile sketch and gradient histogram.
		● hist: Fast histogram optimized approximate greedy algorithm. It uses some performance improvements such as bins caching.	

		• gpu_exact: GPU implementation of exact algorithm. • gpu_hist: GPU implementation of hist algorithm.	
max_bin	256	max number of discrete bins to bucket continuous features, larger value improves the optimality of splits with higher computation time, used only when 'hist' is set	
predictor	cpu_predictor	'gpu_predictor' using GPU for prediction, defult when 'gpu_hist'	?

GBM:
https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/

Learning Task Parameters

- objective [default=reg:squarederror]
  - reg: squarederror, squaredlogerror, logistic..
- eval_metric [default are rmse for regression and error for classification]
  - rmse, mae, logloss, error, merror, mlogloss, auc
- seed [default = 0]
  - can be used for generating reproducible results and for parameter tuning

Tuning steps

1. Choose a relatively **high learning rate**. Generally a learning rate of 0.1 works but somewhere between 0.05 to 0.3 should work for different problems. Determine the **optimum number of trees for this learning rate**. XGBoost has a very useful function called "cv" which performs cross-validation at each boosting iteration and thus returns the optimum number of trees required.
2. **Tune tree-specific parameters** (max_depth, min_child_weight, gamma, subsample, colsample_bytree) for decided learning rate and number of trees. Note that we can choose different parameters to define a tree and I'll take up an example here.
3. Tune **regularization parameters** (lambda, alpha) for xgboost which can help reduce model complexity and enhance performance.
4. **Lower the learning rate** and decide the optimal parameters .

Other Concepts

Bootstrap Confidence Interval

The number of bootstrap repeats defines the variance of the estimate, and more is better, often hundreds or thousands.

The size of the sample taken each iteration may be limited to 60% or 80% of the available data. This will mean that there will be some samples that are not included in the sample. These are called out of bag (OOB) samples.

Bagging and Boosting