Predict Cell Differentiation Stages via Gradient Boost Tree

Yun Yan (yy1533@nyu.edu)

Apr 6 2016

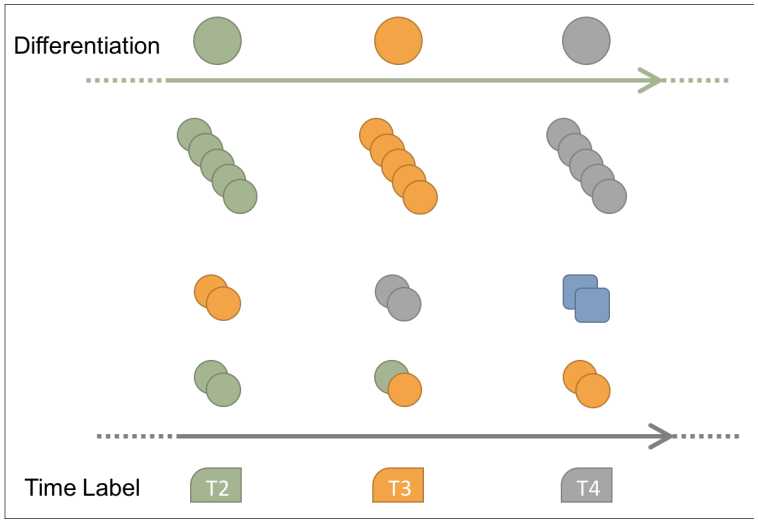
Outline

- Data science problem beneath our project
  - Project analogy: Otto Product Classification Challenge
- Perform data learning via xgboost
  - Classification in probabilistic perspective
  - Gradient boost tree model
  - Result
- Why need more data
- Planned Enhancement
  - Dimension reduction
  - Deal with entity resolution
- General Discussion
  - Drawback/blind area of model

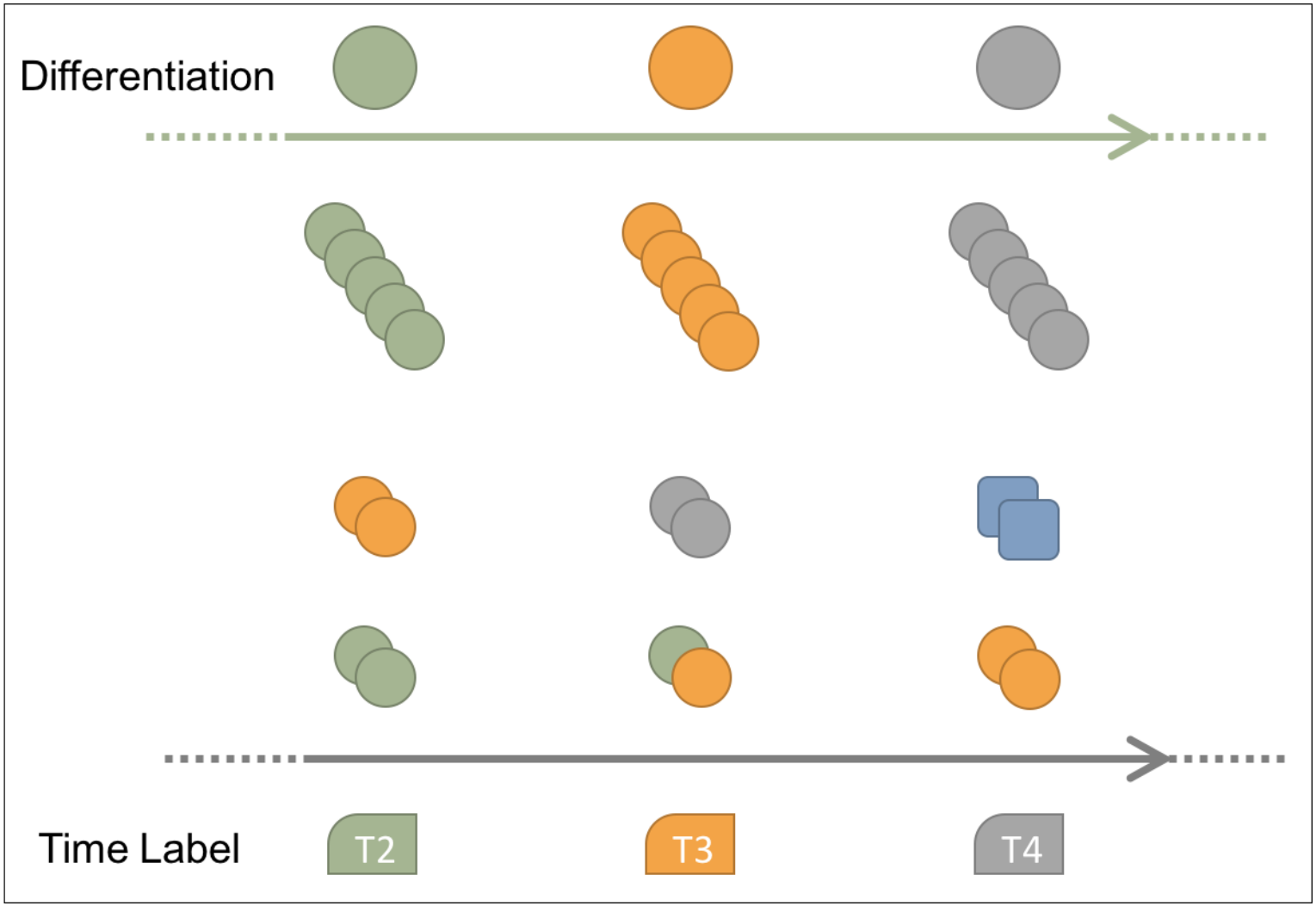
Data sciences challenges

- Competition on Kaggle (here)
- "… due to our diverse global infrastructure, many identical products get classified differently …"

- Cell differentiation
- "… due to sample collection is not necessarily synced with differentiation, many identically matured cells get labelled differently"



Why heterogeneous?



Data sciences challenges (cont)

	Otto Product Challenge	Cell Differentiation Time
Problem	Multi-labeling on same type of product, i.e. entity resolution	Multi-labeling on cells with similar maturarity level
Input	Product features (e.g. length, color)	Gene expressions per cell (sc-RNA-seq)
Model	Softmax Classifier	Softmax Classifier
Output	Probability that product falls into given category	Probability that cells reach at given differentiation stage

Section Summary

- Cell differentiation stage prediction project is analogous to Otto product competition
- Share the data sciences problem
- Similar computational framework could probably solve life sciences challenges

Outline

- Data science problem beneath our project
  - Project analogy: Otto Product Classification Challenge
- Perform data learning via xgboost
  - Classification in probabilistic perspective
  - Gradient boost tree model
  - Result
- Why need more data
- Planned Enhancement
  - Dimension reduction
  - Deal with entity resolution
- General Discussion
  - Drawback/blind area of model

Xgboost, a big & lazy winner

- Xgboost is key element of 1st Winner's solution
- Power of Ensemble: wisdom of weak learners
  - Random forest
  - Adaboost
  - Gradient Boosting
- Boosting > RF > Bagging > Single Tree (see Ref 2)
- Xgboost = implementation of gradient boost tree (see slides later)

Analysis Pipeline for our project

Raw tab file
- Genes x Cells

`io.R`

Training and test dataset
- Cells x Genes
- Label vector

`model_xgboost.R`

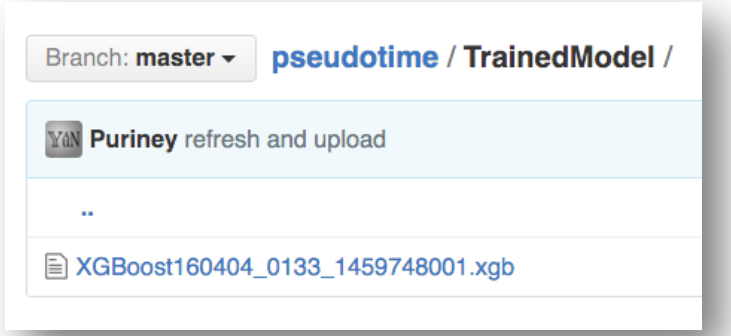
Tree Classifier Model

`model_xgboost.R`

Probability Map on test dataset & Evaluation
- Cells x Time

```
> corner(raw)
             esc1_0h_10_rsem esc1_0h_11_rsem esc1_0h_13_r
0610007L01RIK        5.043080        0.000000         0.00
0610007P14RIK        4.175626        5.724997         6.01
0610007P22RIK        1.937263        0.000000         0.00
0610009B22RIK        0.000000        0.000000         0.00
0610009D07RIK        4.573308        0.000000         0.00
```

```
> corner(tr_x)
             0610007L01RIK 0610007P14RIK 0610007P22RIK 0610009B22RIK 0610009D07RIK
esc1_0h_10_rsem      5.043080      4.175626      1.937263             0      4.573308
esc1_0h_13_rsem      0.000000      6.018430      0.000000             0      0.000000
esc1_0h_18_rsem      5.471163      0.000000      0.000000             0      6.612462
esc1_0h_19_rsem      0.000000      5.826943      0.000000             0      0.000000
esc1_0h_21_rsem      0.000000      5.881477      0.000000             0      0.000000
> head(tr_y)
[1] h0 h0 h0 h0 h0 h0
Levels: h0 h6 h12 h18 h24 h30 h36 h48
```




```
> head(te_PredProbM)
          [,1]        [,2]       [,3]        [,4]        [,5]        [,6]        [,7]        [,8]
[1,] 0.8816421 0.005944975 0.02508449 0.007792724 0.006807924 0.0303574912 0.0300322436 0.012338050
[2,] 0.9806254 0.004665819 0.01031600 0.001372232 0.001031863 0.0008126805 0.0004851386 0.000690924
[3,] 0.9455202 0.003513946 0.02261764 0.005038270 0.007133683 0.0093963090 0.0046692370 0.002110715
[4,] 0.9118249 0.004694429 0.06721365 0.002572345 0.004350583 0.0016389237 0.0037722199 0.003932943
[5,] 0.9267581 0.007896024 0.03787285 0.006578859 0.006483393 0.0048993947 0.0062650349 0.003246349
[6,] 0.7595973 0.021684406 0.16005373 0.014128289 0.016255667 0.0073420894 0.0124910185 0.008447539
```

What does black-box give us?

- Belong to supervised learning algorithm
- Cost/Loss function: Multi-classes Log Loss $-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log p_{ij}$
  - Binary logistics regression $-\frac{1}{N}\sum_{i=1}^{N}[y_i \log p_i + (1-y_i)\log(1-p_i)]$.
  - The lower loss, the better
- Output: **SoftProb**
  - **SoftMax** function
  - If K=2, it is nothing but sigmoid function $\sigma(\mathbf{z})_j = \frac{e^{\mathbf{z}_j}}{\sum_{k=1}^{K} e^{\mathbf{z}_k}}$

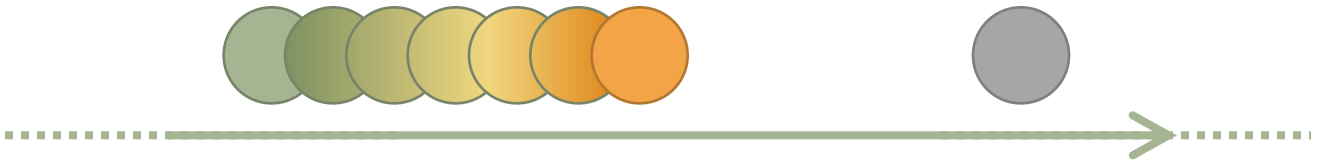
```
> head(te_PredProbM)
          [,1]        [,2]       [,3]        [,4]        [,5]         [,6]         [,7]         [,8]
[1,] 0.8816421 0.005944975 0.02508449 0.007792724 0.006807924 0.0303574912 0.0300322436 0.012338050
[2,] 0.9806254 0.004665819 0.01031600 0.001372232 0.001031863 0.0008126805 0.0004851386 0.000690924
[3,] 0.9455202 0.003513946 0.02261764 0.005038270 0.007133683 0.0093963090 0.0046692370 0.002110715
[4,] 0.9118249 0.004694429 0.06721365 0.002572345 0.004350583 0.0016389237 0.0037722199 0.003932943
[5,] 0.9267581 0.007896024 0.03787285 0.006578859 0.006483393 0.0048993947 0.0062650349 0.003246349
[6,] 0.7595973 0.021684406 0.16005373 0.014128289 0.016255667 0.0073420894 0.0124910185 0.008447539
> apply(head(te_PredProbM), 1, which.max)
[1] 1 1 1 1 1 1
```

- SoftMax yields label 1 for all six samples as max prob
- SoftProb maintain the probability vector

What does black-box give us? (cont)

- Model labelling is performed in probabilistic perspective
- Cell differentiation stage (label) is continuous
- Model interpretation is consistent to biological processing
- Cell could be predicted at 0.5h, 6.23h, etc. Rather than fixed time labels

```
> head(te_PredProbM)
          [,1]        [,2]       [,3]        [,4]        [,5]         [,6]         [,7]         [,8]
[1,] 0.8816421 0.005944975 0.02508449 0.007792724 0.006807924 0.0303574912 0.0300322436 0.012338050
[2,] 0.9806254 0.004665819 0.01031600 0.001372232 0.001031863 0.0008126805 0.0004851386 0.000690924
[3,] 0.9455202 0.003513946 0.02261764 0.005038270 0.007133683 0.0093963090 0.0046692370 0.002110715
[4,] 0.9118249 0.004694429 0.06721365 0.002572345 0.004350583 0.0016389237 0.0037722199 0.003932943
[5,] 0.9267581 0.007896024 0.03787285 0.006578859 0.006483393 0.0048993947 0.0062650349 0.003246349
[6,] 0.7595973 0.021684406 0.16005373 0.014128289 0.016255667 0.0073420894 0.0124910185 0.008447539
> rowSums(head(te_PredProbM))
[1] 1 1 1 1 1 1
> apply(head(te_PredProbM), 1, function(x) crossprod(x, 1:8))
[1] 1.525071 1.045353 1.182171 1.222600 1.214123 1.619989
```



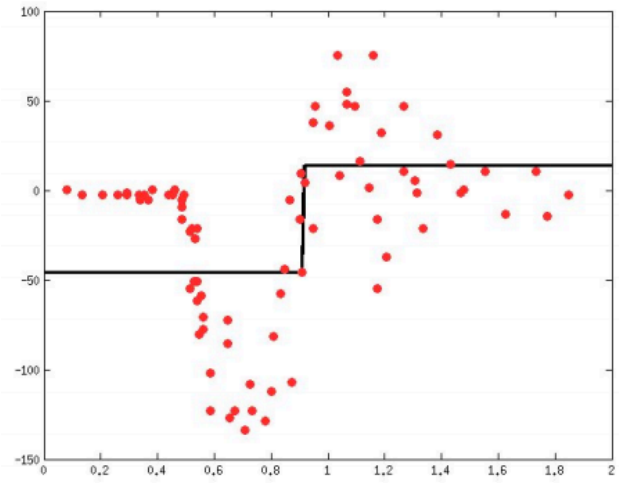
Black-box: Gradient Boosting

- Additive training
  1. Train a weak learner or aggregate weak learners
  2. Compute the error residual
  3. Train another weak learner to minimize the residual
  4. Back to Step 1
- Intuition: Given making choice at time T, there is no regret any more, but rather fixing the issue at time T+1 to make it better

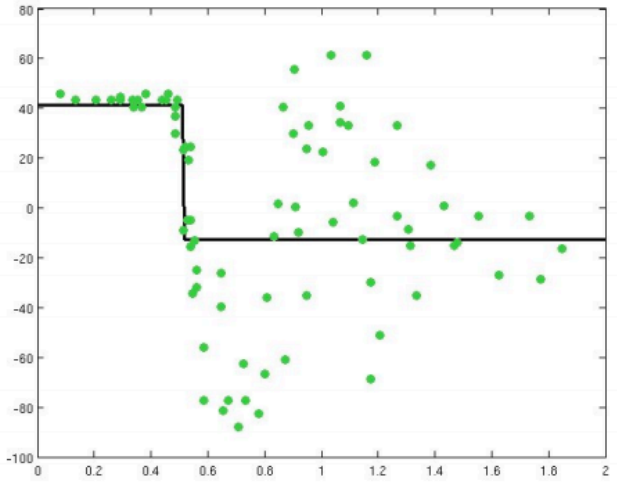
Example of Gradient Boosting

- Additive training with example from Ref 1.
  1. Train a weak learner or aggregate weak learners
  2. Compute the error residual
  3. Train another weak learner to minimize the residual
  4. Back to Step 1

Learn a simple predictor…

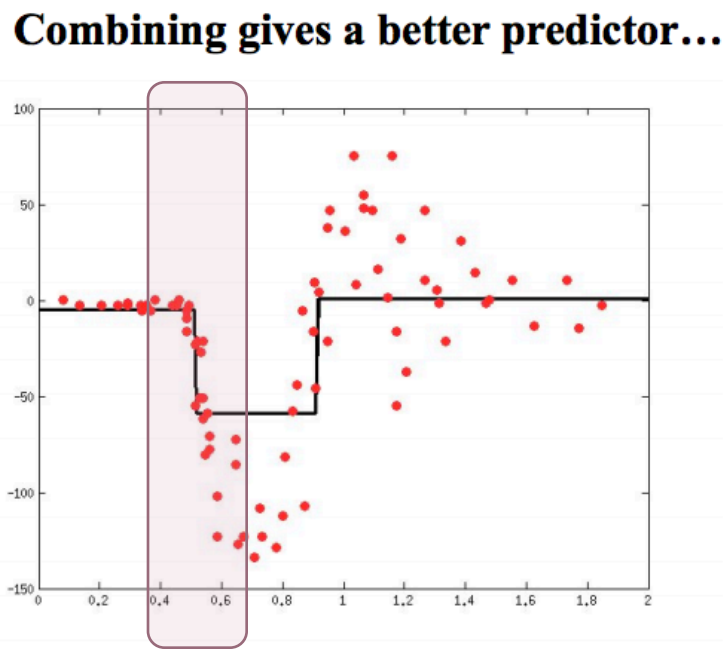


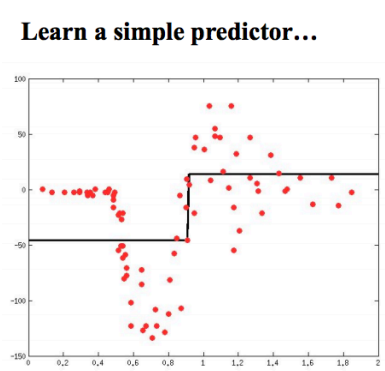
Then try to correct its errors

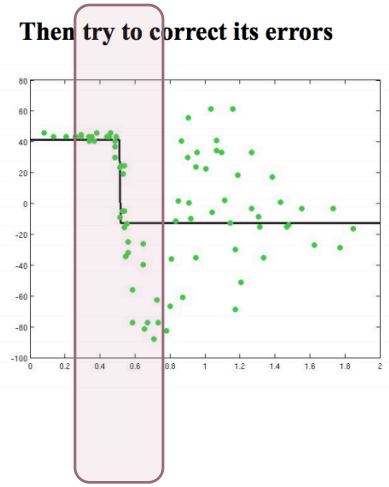


Example of Gradient Boosting (cont)

- Additive training with example from Ref 1.
  1. Train a weak learner or aggregate weak learners
  2. Compute the error residual
  3. Train another weak learner to minimize the residual
  4. Back to Step 1



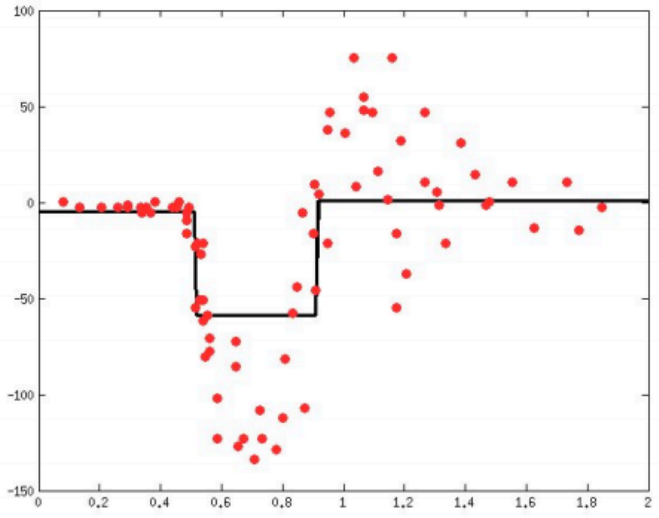




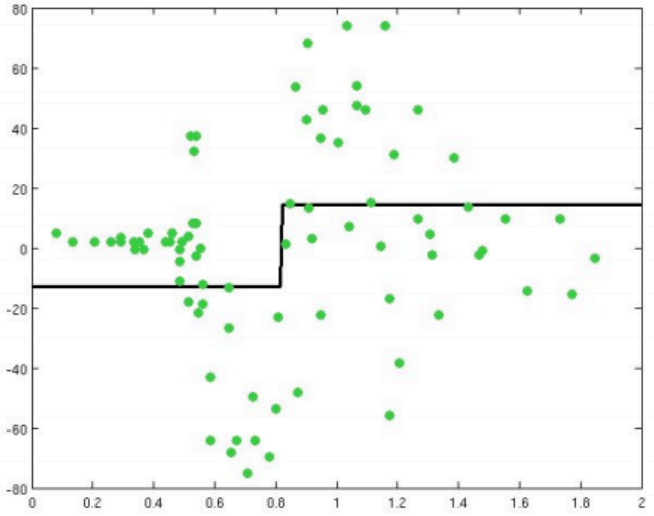
Example of Gradient Boosting (cont)

- Additive training with example from Ref 1.
  1. Train a weak learner or aggregate weak learners
  2. Compute the error residual
  3. Train another weak learner to minimize the residual
  4. Back to Step 1

Combining gives a better predictor…

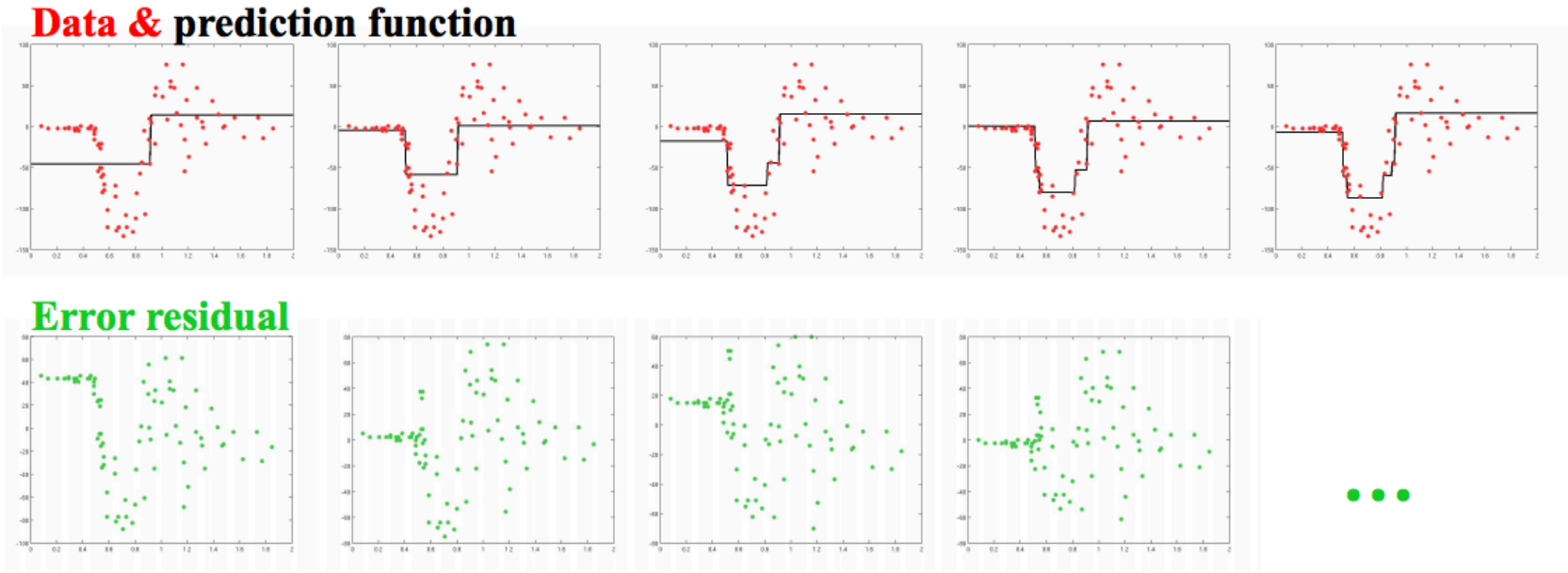


Can try to correct its errors also, & repeat



Example of Gradient Boosting (cont)

- Learn sequence of predictors
- Sum of predictions is increasingly accurate
- Predictive function is increasingly complex

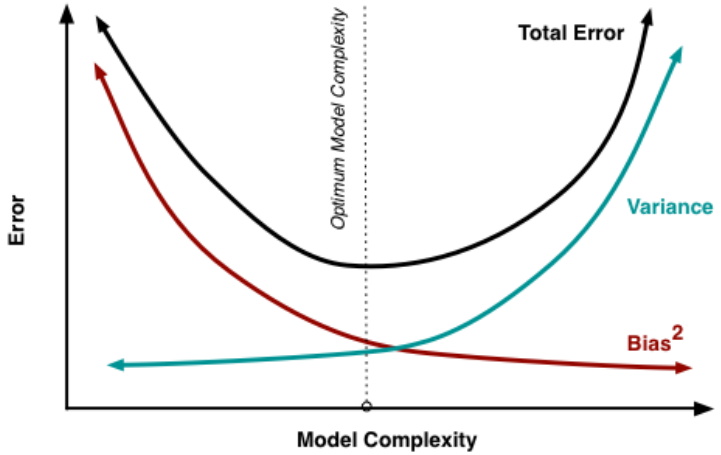


Model training in xgboost

- See src/model_xgboost.R source codes
- 4 key parameters to be tuned:
  - **`nround`**: (0, +Inf] Number of weak leaners. Greater, more weaker learner, there might be higher risk of overfitting;
  - **`subsample`**: (0, 1] Ratio of subpopulation of training dataset for a tree training
  - **`eta`**: (0, 1] Learning rate. Smaller, the model is more conservative;
  - **`max_depth`**: (0, +Inf] Depth of tree structure

Tuning **nround** – Overfitting?

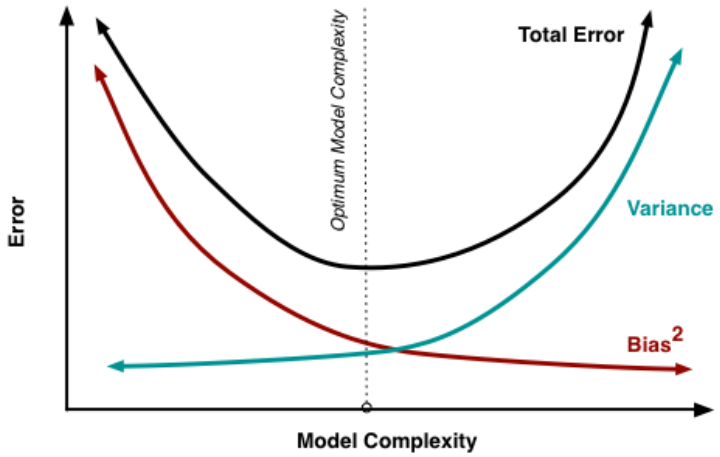
- Bias v.s. Variance
- Cross-Validation: Training dataset is split into actual training and testing dataset
- Function is **xgb.cv**
- Enable early-stop mode for CV




```
bst_cv <- xgb.cv(params = xbt_param,
                 data = tr_x,
                 label = tr_yInt - 1,
                 nfold = cv_fold,
                 nrounds =  cv_nround,
                 nthread = num_threads,
                 print.every.n = ceiling(cv_nround / 10),
                 prediction = TRUE,
                 maximize = FALSE,
                 early.stop.round = rounds_after_x)
```

Tuning **nround** – Overfitting? (cont)

- Bias v.s. Variance
- Cross-Validation: Training dataset is split into actual training and testing dataset
- Enable early-stop mode: loss on testing data is about to increase




```
[0]     train-mlogloss:1.812076+0.003431      test-mlogloss:1.973003+0.027654
[100]   train-mlogloss:0.018502+0.000147      test-mlogloss:0.867796+0.202143
[200]   train-mlogloss:0.013755+0.000138      test-mlogloss:0.842194+0.221320
[300]   train-mlogloss:0.013752+0.000139      test-mlogloss:0.841738+0.221003
Stopping. Best iteration: 369
```

Tuning tree growth

- Prepare parameter grid

```
## tunning grid
xbt_param_grid <- expand.grid(subsample = c(0.5, 0.75, 1),
                              eta = c(0.1),
                              max_depth = c(16, 32, 64))
```

- Choose the parameter set which achieves lowest loss value on test data during CV
  - More weak learners
  - Simpler structure

	subsample	eta	max_depth	mlogloss	nrounds
1	0.50	0.1	16	0.794045	127
2	0.75	0.1	16	0.772358	169
3	1.00	0.1	16	0.841707	369
4	0.50	0.1	32	0.794045	127
5	0.75	0.1	32	0.772358	169
6	1.00	0.1	32	0.841707	369
7	0.50	0.1	64	0.794045	127
8	0.75	0.1	64	0.772358	169
9	1.00	0.1	64	0.841707	369

Evaluation

- **`io.R`** splites raw data into training and testing dataset (8:2)
  - NOT the testing data during cross-validation phase
- Evaluate model performance by applying model on testing data

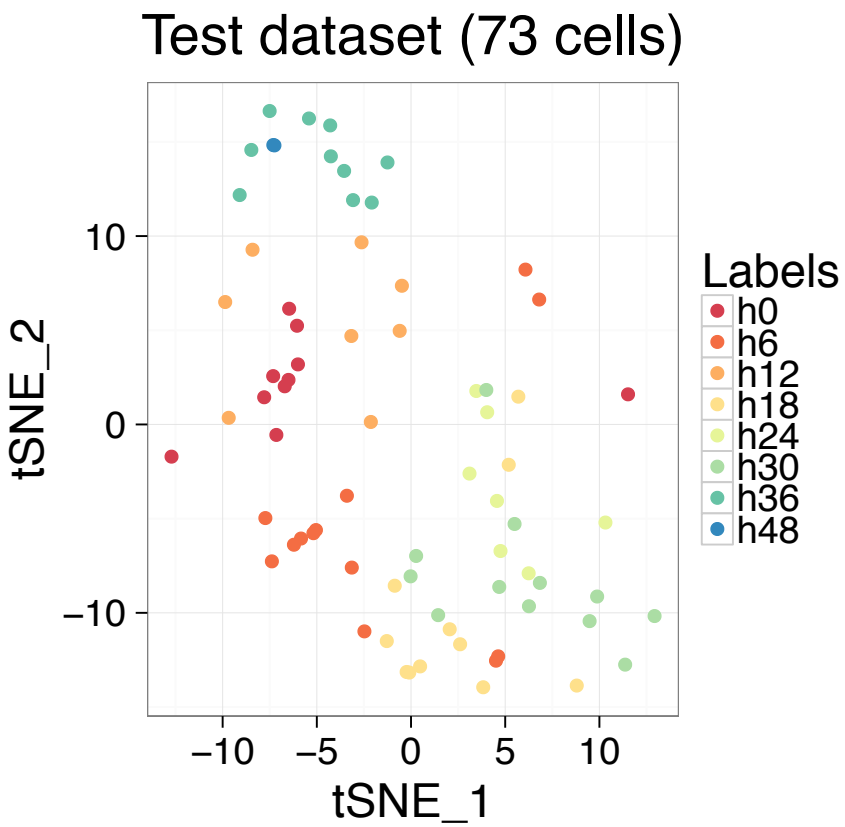
$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log p_{ij}$$

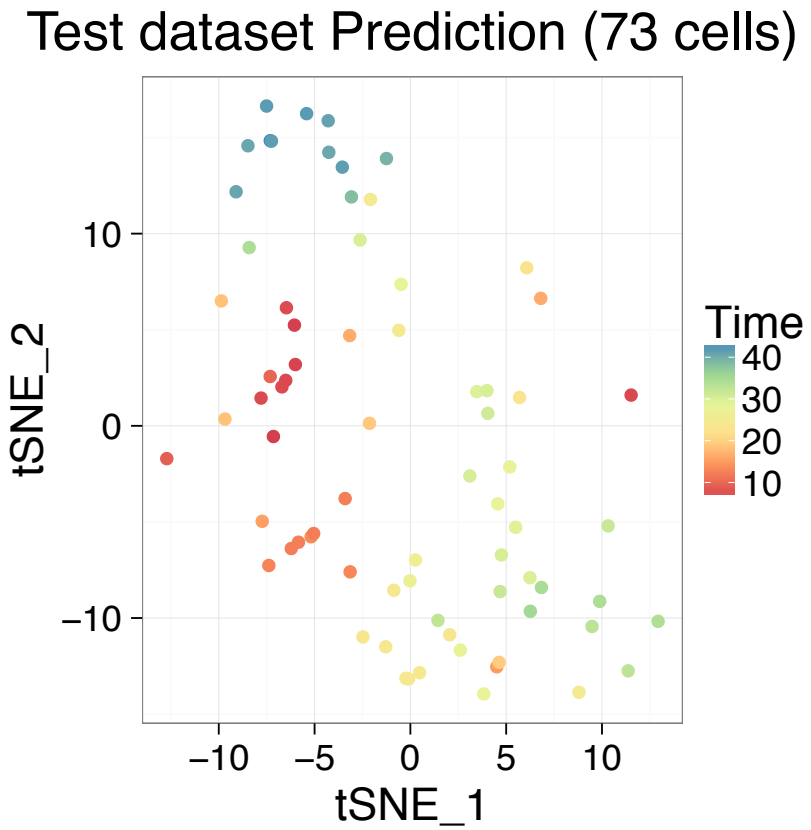
- Results: 0.61
- On average model predicts: there is probability of 0.54 that sample with true label 1 falls into label 1.

```
> logLoss(1, 0.54)
[1] 0.6161861
> logLoss(1, 1)
[1] 0
> logLoss(1, 0.9)
[1] 0.1053605
```

> True label is 1; Predicted label is also 1 at 54%
Loss is 0.6
> True label is 1; Predicted label is also 1 at 100%
Loss is 0
> True label is 1; Predicted label is also 1 at 90%
Loss is ~0.1

Time-track


Test dataset (73 cells)
tSNE_2
tSNE_1
10
0
−10
−10
−5
0
5
10
Labels
h0
h6
h12
h18
h24
h30
h36
h48


Test dataset Prediction (73 cells)
tSNE_2
tSNE_1
10
0
−10
−10
−5
0
5
10
Time
40
30
20
10

Section Summary

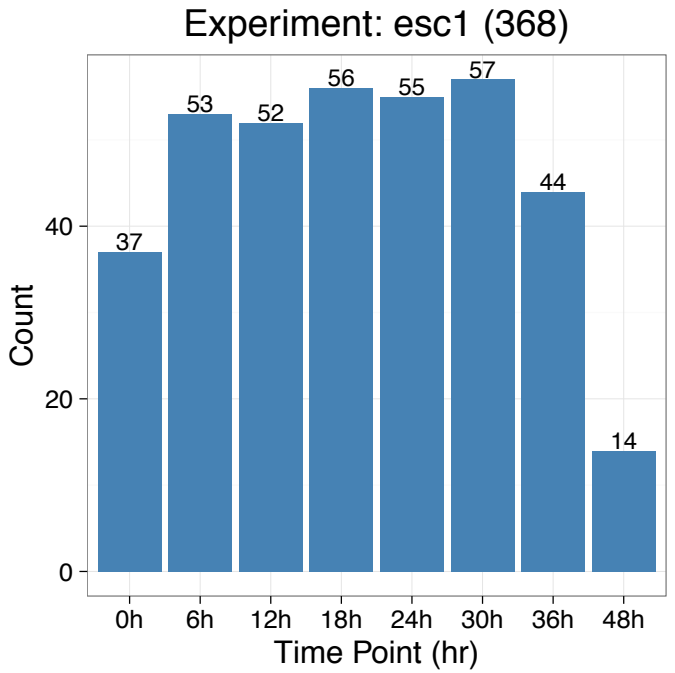
- Establish xgboost pipeline to train GBT
  - Parameter tunings
- Evaluation using multi-class log loss function
- Prediction in probabilistic perspective
- Time-track

Outline

- Data science problem beneath our project
  - Project analogy: Otto Product Classification Challenge
- Perform data learning via xgboost
  - Classification in probabilistic perspective
  - Gradient boost tree model
  - Result
- Why need more data
- Planned Enhancement
  - Dimension reduction
  - Deal with entity resolution
- General Discussion
  - Drawback/blind area of model

Samples per class

- #Features >> #Samples
  - # of genes: 17627
  - # of samples on training data: 295
- Data imbalance
  - Statistically up/down sampling



Outline

- Data science problem beneath our project
  - Project analogy: Otto Product Classification Challenge
- Perform data learning via xgboost
  - Classification in probabilistic perspective
  - Gradient boost tree model
  - Result
- Why need more data
- Planned Enhancement
  - Dimension reduction
  - Deal with entity resolution
- General Discussion
  - Drawback/blind area of model

Data pre-processing

1. Apply dimension reduction first
   - tSNE to 2D
   - PCA
   - EFA (exploratory factor analysis)
     - Note: #Samples ≥ 5 * #Features (see Ref 3)
2. Perform Clustering/Similarity Search to exclude ambiguous samples
3. Perform GBT on samples projected to reduced dimensions space

Outline

- Data science problem beneath our project
  - Project analogy: Otto Product Classification Challenge
- Perform data learning via xgboost
  - Classification in probabilistic perspective
  - Gradient boost tree model
  - Result
- Why need more data
- Planned Enhancement
  - Dimension reduction
  - Deal with entity resolution
- General Discussion
  - Drawback/blind area of model

General representation of unseen facts

- 0, 6, 12, …, 48 hours are project-specific
- Others 6h v.s. Our 6h?
- Lack of general index to indicate cell differentiation level
  - Our model cannot extend to support predicting others' data
  - Others' data cannot be used to enhance our model

Summary

- GBT is suitable for the black-box
- Establish pipeline via xgboost framework
- More data is needed
- Pre-processing is doable for potential enhancement
- Open question about universe mature level index

References

1. http://sli.ics.uci.edu/Classes/2016W-178
2. Trevor Hastie, “Trees, Bagging, Random Forests and Boosting”, Page 17, http://jessica2.msri.org/attachments/10778/10778-boost.pdf
3. Chapter 14, “R in Action, Data analysis and graphics with R”, 1st Edition