

Universidade Presbiteriana Mackenzie

Disciplina: Classificação

XGBoost

Professor: Bruno Silva (ibm.biz/brunosilva)

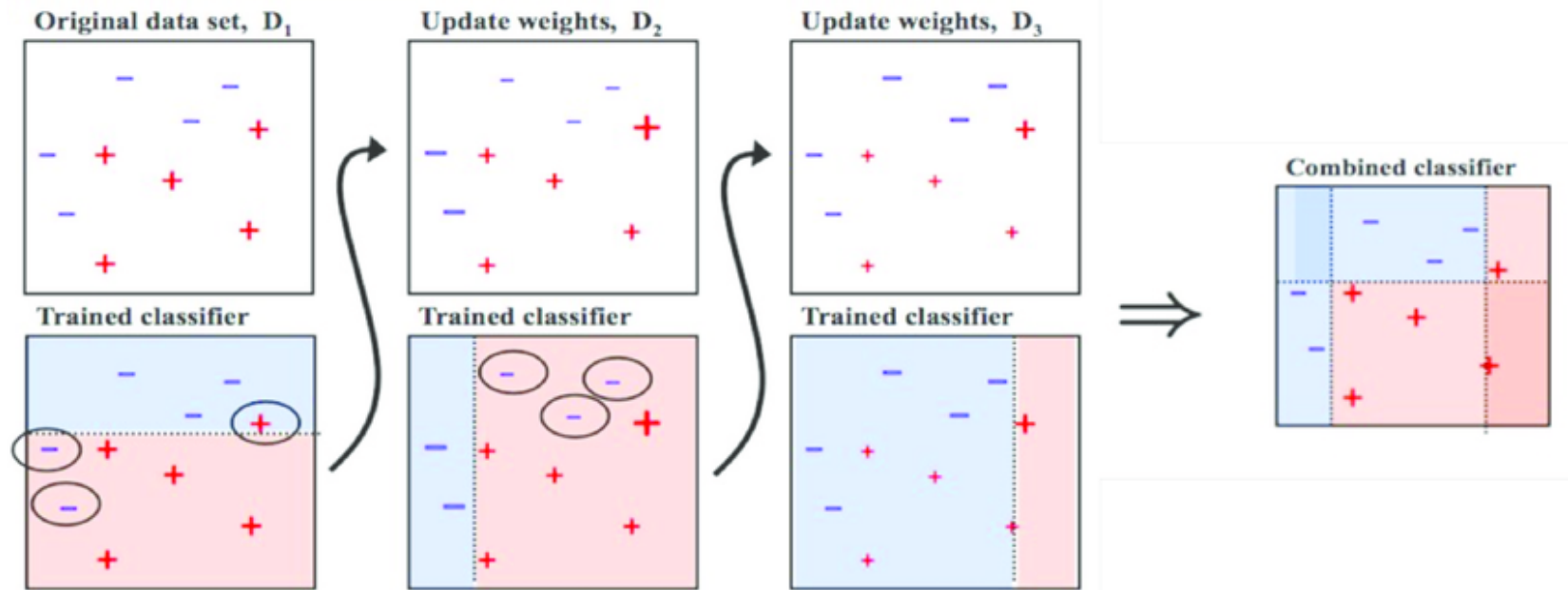
Variant of algorithms to learn Tree Ensembles

- Random Forest (Breiman 1997)
 - RandomForest packages in R and python
- Gradient Tree Boosting (Friedman 1999)
 - RGBM
 - `sklearn.ensemble.GradientBoostingClassifier`
- Gradient Tree Boosting with Regularization (variant of original GBM)
 - Regularized Greedy Forest (RGF) XGBoost\$

Learning Trees : Advantage and Challenges

- Advantages of tree-based methods
 - Highly accurate: almost half of data science challenges are won by tree based methods
 - Easy to use: invariant to input scale, get good performance with little tuning
 - Easy to interpret and control
- Challenges on learning tree(ensembles)
 - Control over-fitting
 - Improve training speed and scale up to larger dataset

Adaboosting or Adaptive Boosting



Marsh, Brendan. (2016). Multivariate Analysis of the Vector Boson Fusion Higgs Boson.

https://www.researchgate.net/figure/Training-of-an-AdaBoost-classifier-The-first-classifier-trains-on-unweighted-data-then_fig3_306054843

https://www.researchgate.net/figure/Training-of-an-AdaBoost-classifier-The-first-classifier-trains-on-unweighted-data-then_fig3_306054843

Gradient Boosting

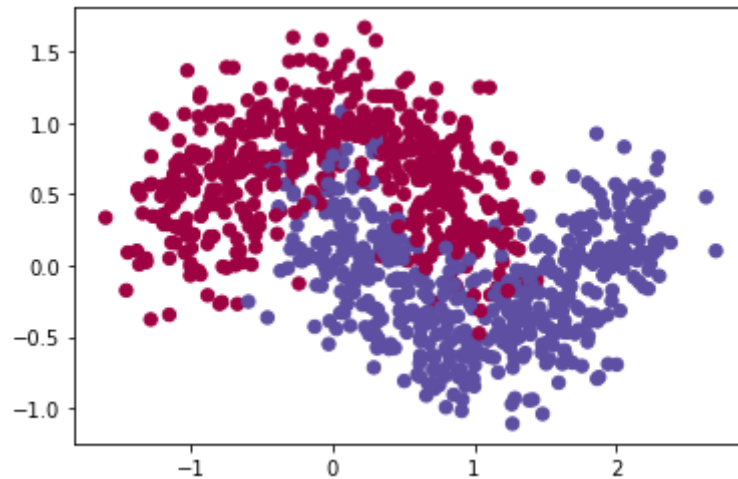
- It works by sequentiall adding predictors to an ensemnble
- Each new model corrects its predecessor
- Let's see the code.

Generate the data

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons

np.random.seed(10)
X, y = make_moons(1000, noise=0.25)
plt.scatter(X[:, 0], X[:, 1], s=40, c=y, cmap=plt.cm.Spectral)
```

```
Out[1]: <matplotlib.collections.PathCollection at 0x12f121f28>
```



Train the first DecisionTreeRegressor

```
In [2]: from sklearn.tree import DecisionTreeRegressor
tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

[illegible]

Now train a second DecisionTreeRegressor on the residual errors made by the first predictor:

```
In [3]: y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

[illegible]

Then we train a third regressor on the residual errors made by the second predictor:

```
In [4]: y3 = y2 - tree_reg2.predict(X)
        tree_reg3 = DecisionTreeRegressor(max_depth=2)
        _ = tree_reg3.fit(X, y3)
```

```
In [5]: y_pred = sum(tree.predict(X[:10]) for tree in (tree_reg1, tree_reg2, tree_reg3))
        print(y_pred)
        print(y[:10])
```

```
[0.87586386 0.94009311 0.21689014 0.08010433 0.87586386 0.87586386
 0.00936549 0.31653543 0.08010433 0.08010433]
[1 1 1 0 1 1 0 0 0 0]
```

Gradient Boosting from Sklearn

```
In [6]: from sklearn.ensemble import GradientBoostingRegressor
gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0)
gbrt.fit(X, y)
gbrt.predict(X[:10]), y_pred
```

```
Out[6]: (array([0.87586386, 0.94009311, 0.21689014, 0.08010433, 0.87586386,
                0.87586386, 0.00936549, 0.31653543, 0.08010433, 0.08010433]),
        array([0.87586386, 0.94009311, 0.21689014, 0.08010433, 0.87586386,
                0.87586386, 0.00936549, 0.31653543, 0.08010433, 0.08010433]))
```

**We can improve this algorithm even further with
XGBoost**

What is XGBoost

- A Scalable System for Learning Tree Ensemble
 - Model improvement
 - Regularized objective for better model
 - Systems optimizations
 - Out of core computing
 - Parallelizationn
 - Cache optimization
 - Distributed computing
 - Algorithm improvements
 - Sparse aware algorithm
 - Weighted approximate quantile sketch.
- In short, faster tool for learning better models

Machine Learning Challenge Winning Solutions

- The most frequently used tool by data science competition winners
 - 17 out of 29 winning solutions in kaggle (2015) used XGBoost
 - Solve wide range of problems: store sales prediction; high energy physics event classification; web text classification; customer behavior prediction; motion detection; ad click through rate prediction; malware classification; product categorization; hazard risk prediction; massive online course dropout rate prediction
- Present and Future of KDDCup. Ron Bekkerman (KDDCup 2015 chair):

"something dramatic happened in Machine Learning over the past couple of years. It is called XGBoost - a package implementing Gradient Boosted Decision Trees that works wonders in data classification. Apparently, every winning team used XGBoost, mostly in ensemble with other classifiers. Most surprisingly, the winning teams report very minor improvements that ensembles bring over a single well configured XGBoost ..."

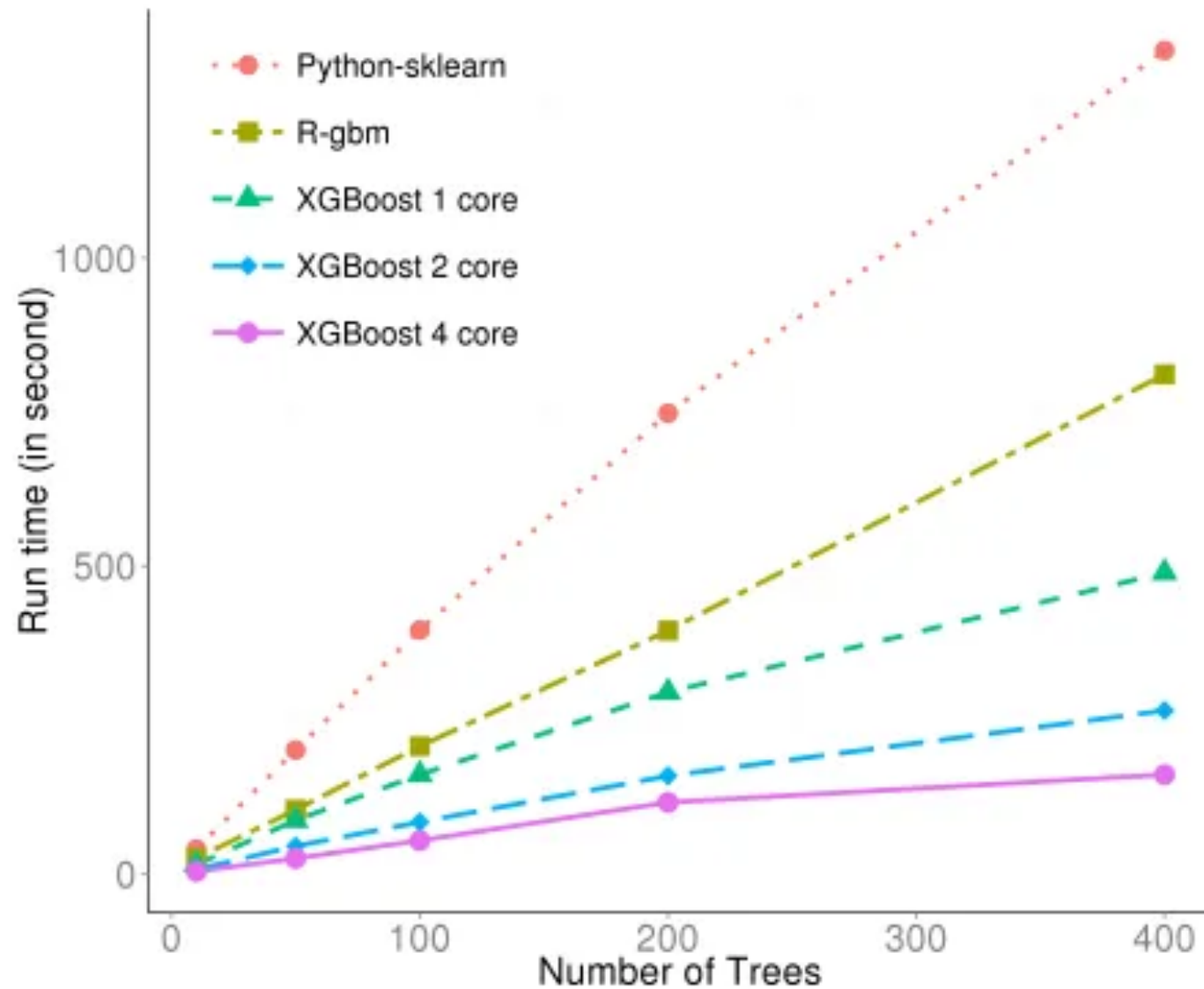
- A lot contributions from the kaggle community

Machine Learning Challenge Winning Solutions

XGBoost is extensively used by machine learning practitioners to create state of art data science solutions, this is a list of machine learning winning solutions with XGBoost. Please send pull requests if you find ones that are missing here.

- Maksims Volkovs, Guangwei Yu and Tomi Poutanen, 1st place of the [2017 ACM RecSys challenge](http://2017.recsyschallenge.com/) (<http://2017.recsyschallenge.com/>). Link to [paper](http://www.cs.toronto.edu/~mvolkovs/recsys2017_challenge.pdf) (http://www.cs.toronto.edu/~mvolkovs/recsys2017_challenge.pdf).
- Vlad Sandulescu, Mihai Chiru, 1st place of the [KDD Cup 2016 competition](https://kddcup2016.azurewebsites.net) (<https://kddcup2016.azurewebsites.net>). Link to [the arxiv paper](http://arxiv.org/abs/1609.02728) (<http://arxiv.org/abs/1609.02728>).
- Marios Michailidis, Mathias Müller and HJ van Veen, 1st place of the [Dato Truly Native? competition](https://www.kaggle.com/c/dato-native) (<https://www.kaggle.com/c/dato-native>). Link to [the Kaggle interview](http://blog.kaggle.com/2015/12/03/dato-winners-interview-1st-place-mad-professors/) (<http://blog.kaggle.com/2015/12/03/dato-winners-interview-1st-place-mad-professors/>).
- Vlad Mironov, Alexander Guschin, 1st place of the [CERN LHCb experiment Flavour of Physics competition](https://www.kaggle.com/c/flavours-of-physics) (<https://www.kaggle.com/c/flavours-of-physics>). Link to [the Kaggle interview](http://blog.kaggle.com/2015/11/30/flavour-of-physics-technical-write-up-1st-place-go-polar-bears/) (<http://blog.kaggle.com/2015/11/30/flavour-of-physics-technical-write-up-1st-place-go-polar-bears/>).
- Josef Slavicek, 3rd place of the [CERN LHCb experiment Flavour of Physics competition](https://www.kaggle.com/c/flavours-of-physics) (<https://www.kaggle.com/c/flavours-of-physics>). Link to [the Kaggle interview](http://blog.kaggle.com/2015/11/23/flavour-of-physics-winners-interview-3rd-place-josef-slavicek/) (<http://blog.kaggle.com/2015/11/23/flavour-of-physics-winners-interview-3rd-place-josef-slavicek/>).
- Mario Filho, Josef Feigl, Lucas, Gilberto, 1st place of the [Caterpillar Tube Pricing competition](https://www.kaggle.com/c/caterpillar-tube-pricing) (<https://www.kaggle.com/c/caterpillar-tube-pricing>). Link to [the Kaggle interview](http://blog.kaggle.com/2015/09/22/caterpillar-winners-interview-1st-place-gilberto-josef-lucas-mario/) (<http://blog.kaggle.com/2015/09/22/caterpillar-winners-interview-1st-place-gilberto-josef-lucas-mario/>).

XGBoost Performance



Reference: <https://www.r-bloggers.com/introduction-to-xgboost-r-package>
<https://github.com/melisgl/higgsml/blob/master/doc/model.md/>

What does XGBoost learn

- A self-contained derivation of general gradient boosting algorithm
- Resembles the original GBM derivation by Friedman
 - <https://statweb.stanford.edu/~jhf/ftp/trebst.pdf>
(<https://statweb.stanford.edu/~jhf/ftp/trebst.pdf>)
- Only preliminary of calculus is needed

Elements of Supervised Learning

- Model: how to make prediction $\hat{y}_i = f(x_i)$
 - Linear model: $\hat{y}_i = \sum_j w_j x_{ij}$
- Parameters: the things we need to learn from data
 - Linear model: $\Theta = \{w_j | j = 1, \dots, d\}$
- Objective Function: $Obj(\Theta) = L(\Theta) + \Omega(\Theta)$
 - Training Loss $L(\Theta)$ measures how well model fit on training data
 - Regularization $\Omega(\Theta)$ measures complexity of model
- Linear Model: $L(\Theta) = \sum_i (\hat{y}_i - y_i)^2$, $\Omega(\Theta) = \lambda \|w\|_2^2$

Elements of Tree Learning

- Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

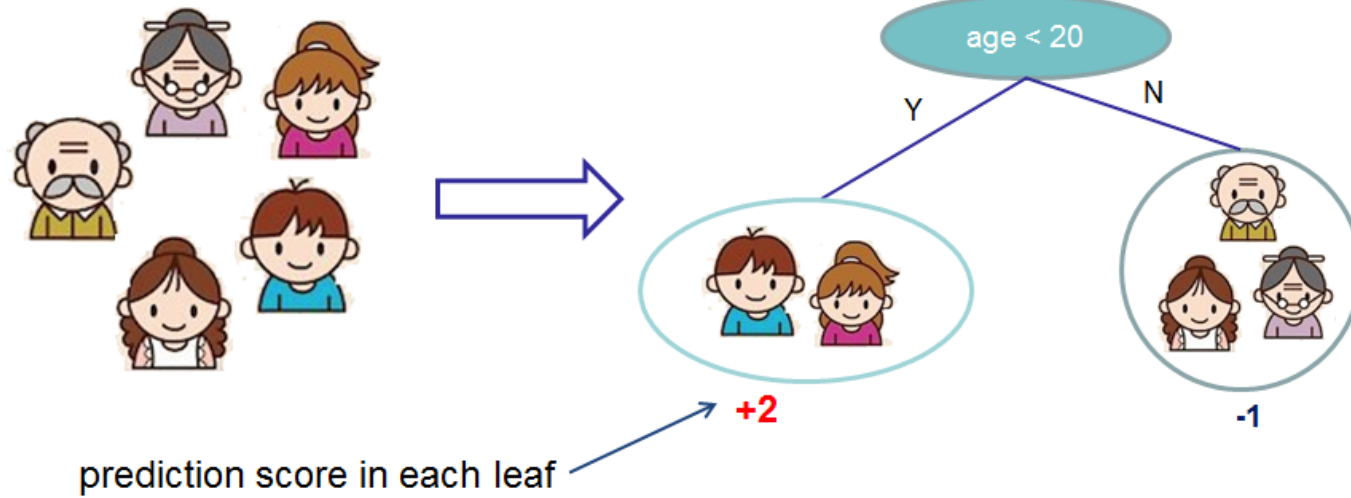
- \mathcal{F} corresponds to Space of Regression trees
- Objective is given by, where l is the loss and Ω is the regularization (complexity)

$$Obj = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

Decision Trees

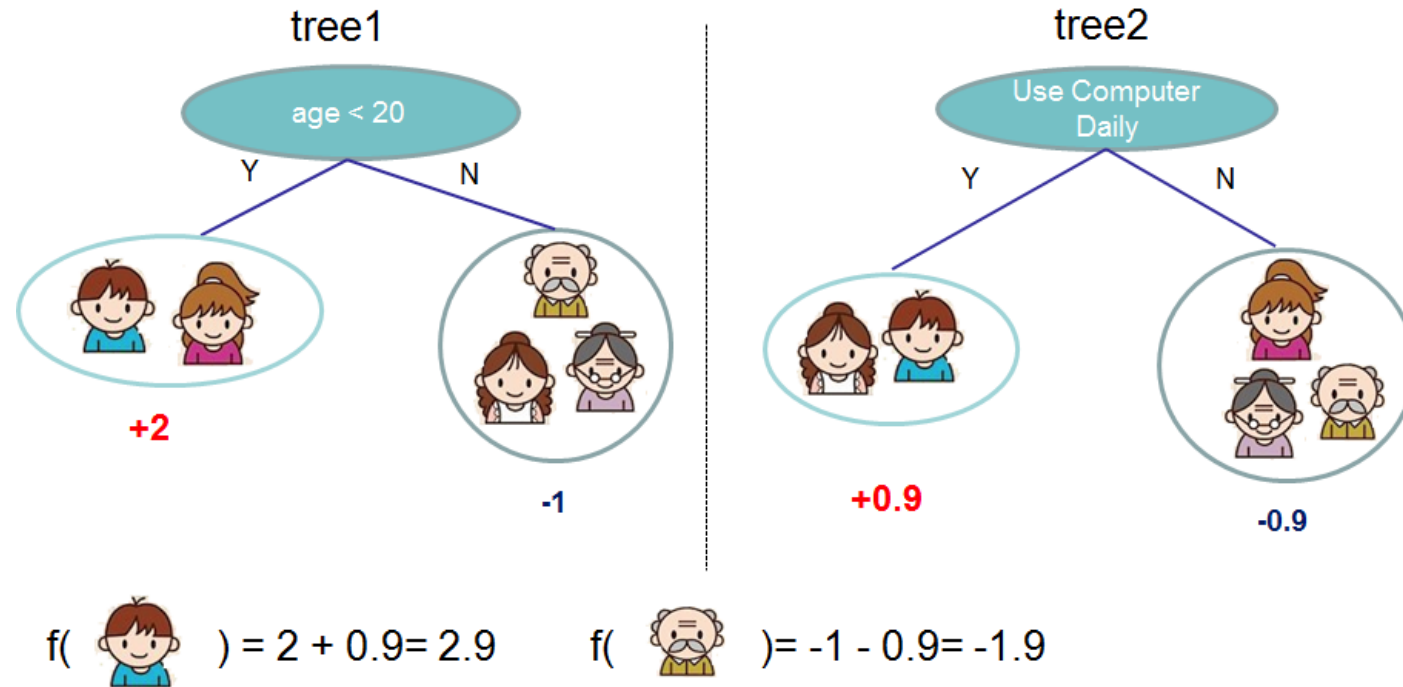
Input: age, gender, occupation, ...

Like the computer game X



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

Decision Trees



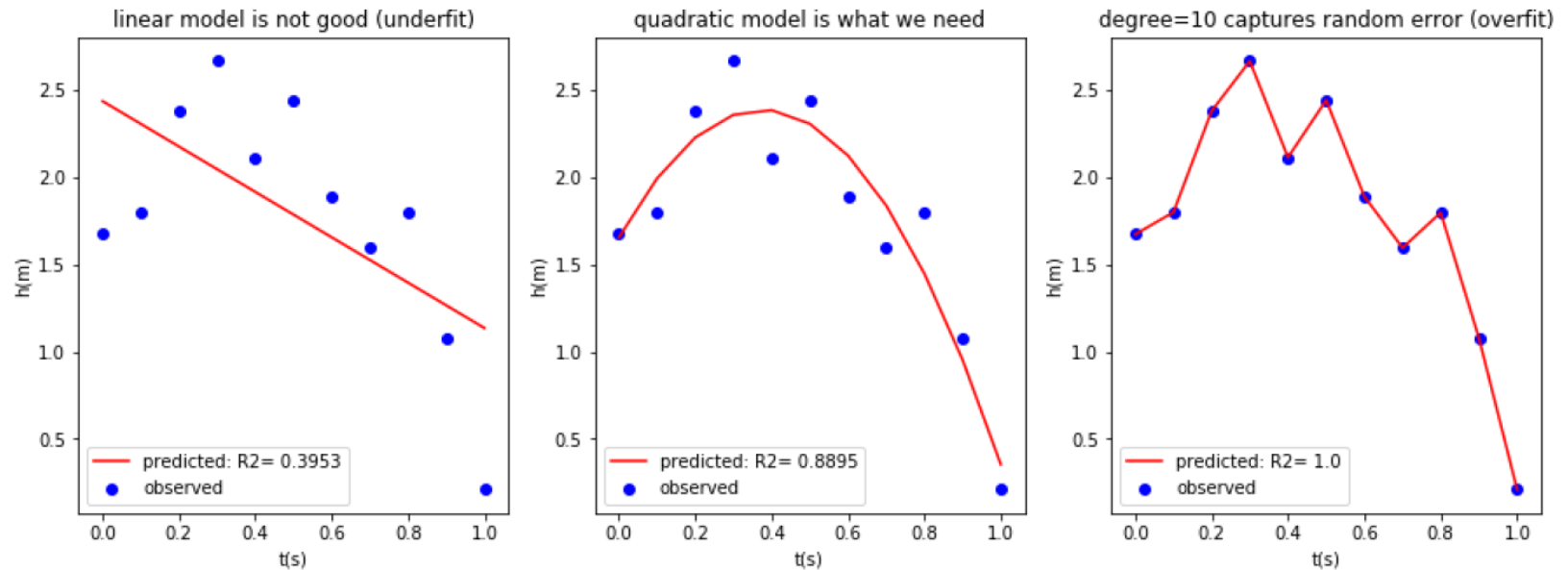
Reference: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

Trade off in learning algorithms

$$Obj = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

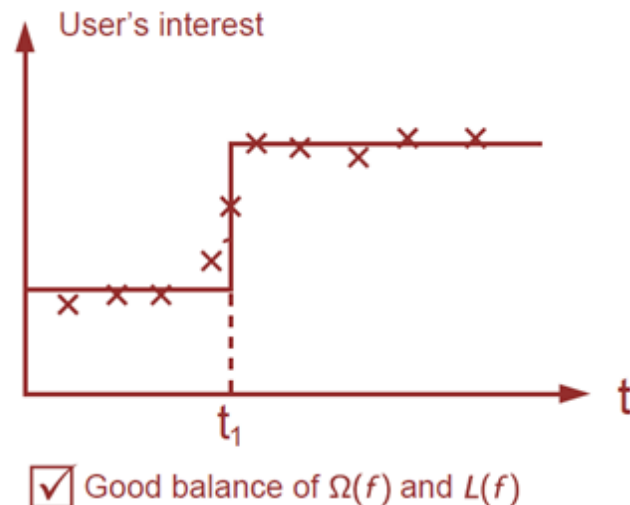
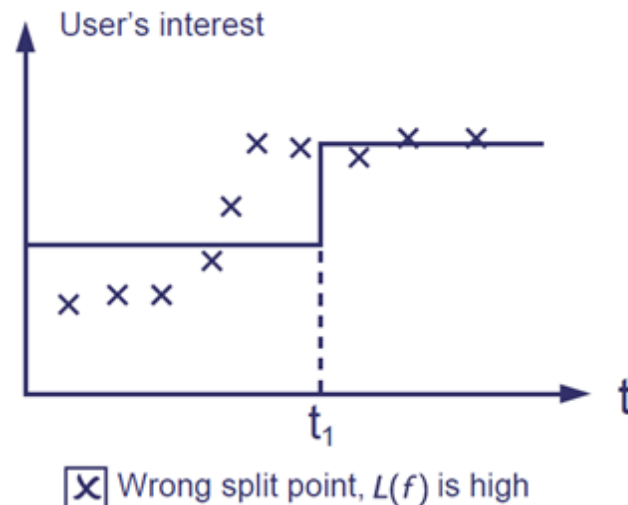
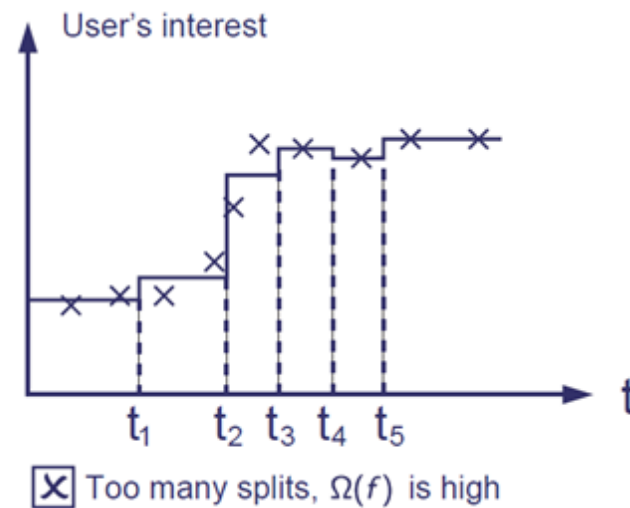
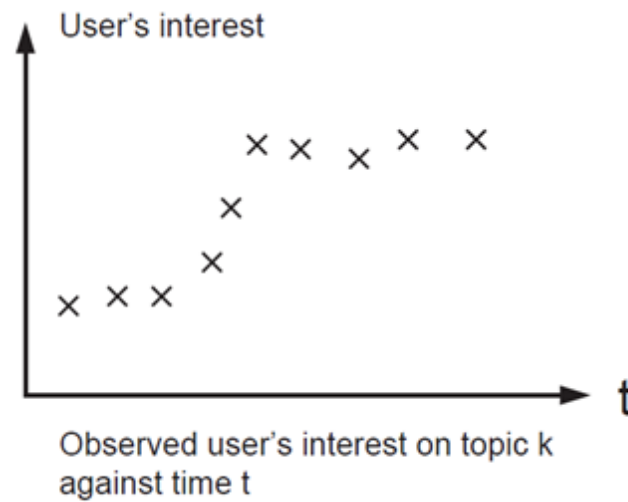
- Optimizing training loss encourages predictive models
 - Fitting well in training data at least get you close to training data which is hopefully close to the underlying distribution
- Optimizing regularization encourages simple models
 - Simpler models tends to have smaller variance in future predictions, making prediction stable

Bias \times Variance Trade-off



Reference: <https://medium.com/towards-artificial-intelligence/bias-variance-tradeoff-illustration-using-pylab-202943bf4c78> (<https://medium.com/towards-artificial-intelligence/bias-variance-tradeoff-illustration-using-pylab-202943bf4c78>)

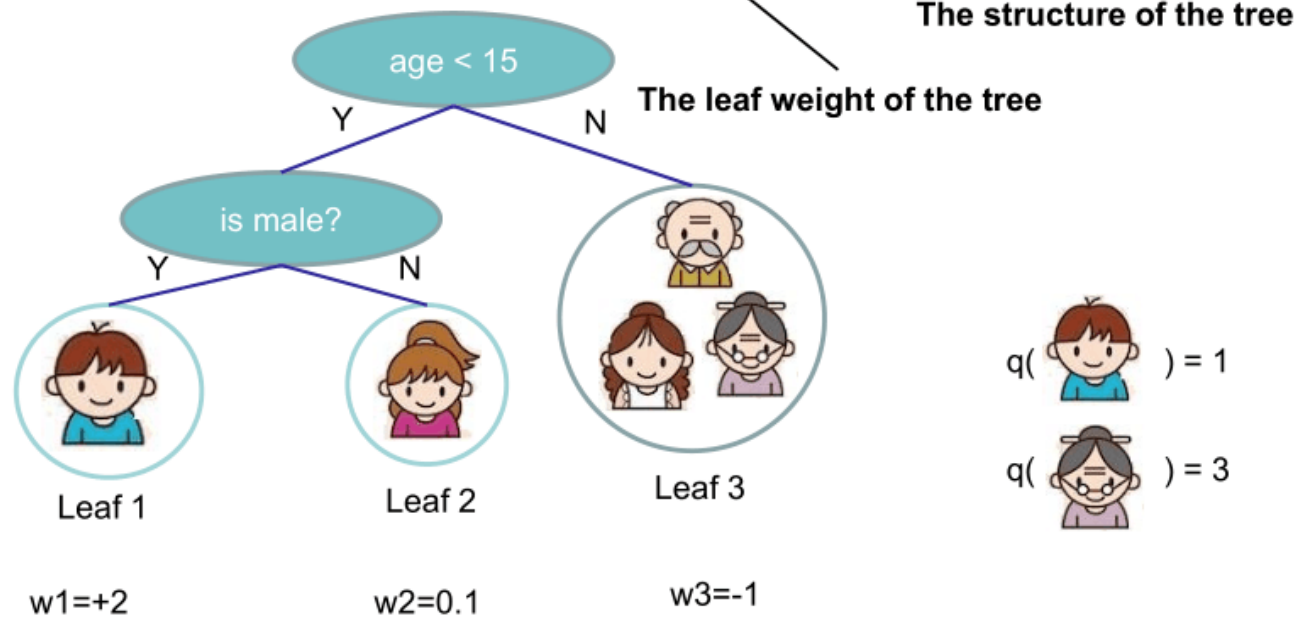
Regularization in Decision Trees



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

Complexity in Trees

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q: \mathbf{R}^d \rightarrow \{1, 2, \dots, T\}$$



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

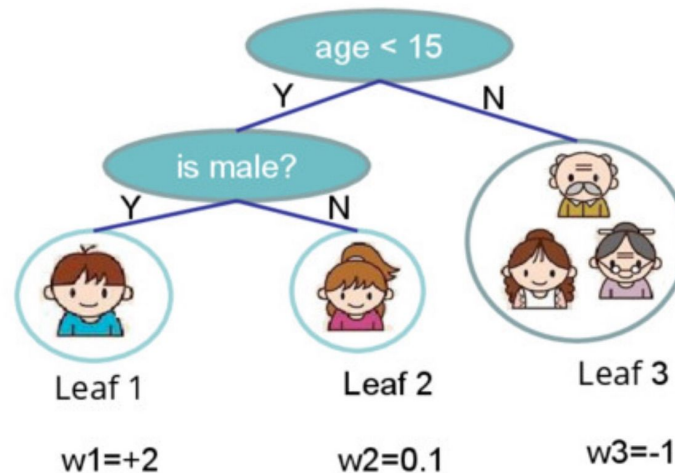
Complexity in Trees

Objective in XGBoost

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

Number of leaves

L2 norm of leaf scores



$$\Omega = \gamma 3 + \frac{1}{2} \lambda (4 + 0.01 + 1)$$

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

Learning the tree ensemble

- Mathematically, we can write our model in the form

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), f_k \in \mathcal{F}$$

- where K is the number of trees, f is a function in the functional space \mathcal{F} , and \mathcal{F} is the set of all possible CARTs. The objective function to be optimized is given by

$$\text{obj}(\theta) = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

Learning the tree ensemble

- Learning tree structure is much harder than traditional optimization problem where you can simply take the gradient.
- It is intractable to learn all the trees at once.
- Instead, we use an additive strategy: fix what we have learned, and add one new tree at a time.

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

...

$$\hat{y}_i^{(t)} = \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Taylor Expansion of Loss

- Goal $Obj^{(t)} =$

$$\sum_{i=1}^n l \left(y_i, \right. \\ \left. \hat{y}_i^{(t-1)} + f_t(x_i) \right) \\ + \Omega(f_t) \\ + \text{constant}$$

- Taylor expansion

- Recall $f(x + \Delta x)$
 $\simeq f(x)$
 $+ f'(x)\Delta x$
 $+ \frac{1}{2} f''(x)\Delta x^2$

- Define g_i
 $= \partial_{\hat{y}^{(t-1)}} l$
 $\left(y_i, \right.$
 $\left. \hat{y}^{(t-1)} \right),$
 h_i

After we remove all the constants, the specific objective at step t becomes

$$\begin{aligned}\text{obj}^{(t)} &\approx \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^n \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T\end{aligned}$$

where $I_j = \{i | q(x_i) = j\}$ is the set of indices of data points assigned to the j -th leaf

Final equation, finally 😊💧

We could further compress the expression by defining $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$:






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

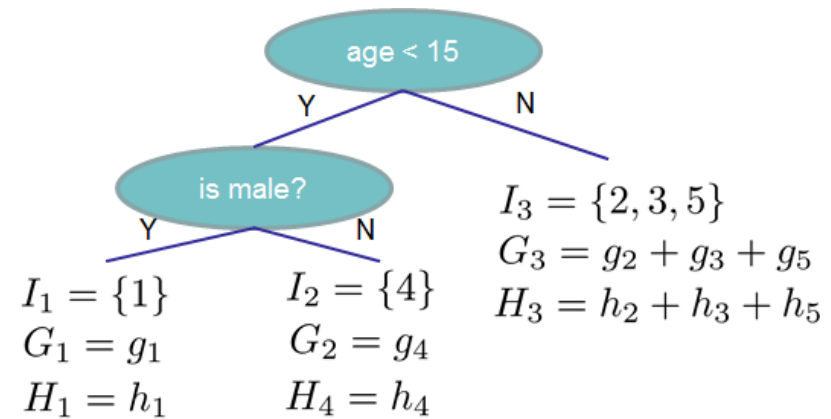
In this equation, w_j are independent with respect to each other, the form $G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2$ is quadratic and the best w_j for a given structure $q(x)$ and the best objective reduction we can get is:

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

Calculating costs

Instance index gradient statistics

1		g_1, h_1
2		g_2, h_2
3		g_3, h_3
4		g_4, h_4
5		g_5, h_5



$$Obj = - \sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

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

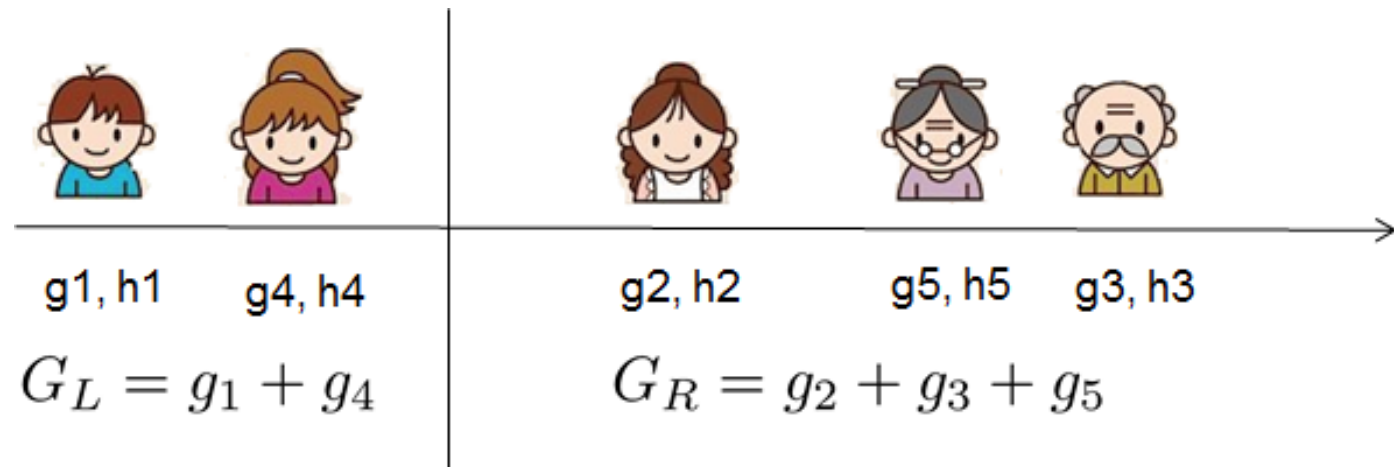
Learn the tree structure

- Now that we have a way to measure how good a tree is, ideally we would enumerate all possible trees and pick the best one.
- In practice this is intractable, so we will try to optimize one level of the tree at a time. Specifically we try to split a leaf into two leaves, and the score it gains is

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

1) the score on the new left leaf 2) the score on the new right leaf 3) The score on the original leaf 4) regularization on the additional leaf.

Learn the tree structure



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

Working on practice

```
In [7]: from sklearn import datasets
import xgboost as xgb

iris = datasets.load_iris()
X = iris.data
y = iris.target
```

Split data

```
In [32]: from sklearn.model_selection import train_test_split  
  
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3, random_state=43)
```

Setting up our data with XGBoost

```
In [33]: D_train = xgb.DMatrix(X_train, label=Y_train)
D_test = xgb.DMatrix(X_test, label=Y_test)
```

Defining an XGBoost model

```
In [34]: # set xgboost params
param = {
    'max_depth': 3, # the maximum depth of each tree
    'eta': 0.5, # the training step for each iteration
    'silent': 1, # logging mode - quiet
    'objective': 'multi:softprob', # error evaluation for multiclass training
    'num_class': 3} # the number of classes that exist in this dataset
steps = 20 # the number of training iterations
```

```
In [35]: model = xgb.train(param, D_train, steps)
```

Let's now run an evaluation

```
In [36]: import numpy as np
from sklearn.metrics import precision_score, recall_score, accuracy_score

preds = model.predict(D_test)
best_preds = np.asarray([np.argmax(line) for line in preds])

print("Precision = {}".format(precision_score(Y_test, best_preds, average='macro'
')))
print("Recall = {}".format(recall_score(Y_test, best_preds, average='macro')))
print("Accuracy = {}".format(accuracy_score(Y_test, best_preds)))
```

Precision = 0.9351851851851851

Recall = 0.9291101055806937

Accuracy = 0.9333333333333333

Improving Evaluation

Test Model

```
In [38]: preds = grid.predict(X_test)

print("Precision = {}".format(precision_score(Y_test, preds, average='macro')))
print("Recall = {}".format(recall_score(Y_test, preds, average='macro')))
print("Accuracy = {}".format(accuracy_score(Y_test, preds)))
```

Precision = 0.9547511312217195

Recall = 0.9547511312217195

Accuracy = 0.9555555555555556

```
In [41]: from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC(probability=True)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='soft'
)

_ = voting_clf.fit(X_train, Y_train)
```

```
In [42]: preds = voting_clf.predict(X_test)

print("Precision = {}".format(precision_score(Y_test, preds, average='macro')))
print("Recall = {}".format(recall_score(Y_test, preds, average='macro')))
print("Accuracy = {}".format(accuracy_score(Y_test, preds)))
```

Precision = 0.9547511312217195

Recall = 0.9547511312217195

Accuracy = 0.9555555555555556

Further Reading

1. Tianqi Chen - <https://tqchen.com/> (<https://tqchen.com/>).
2. XGBoost: A Scalable Tree Boosting System - <https://arxiv.org/pdf/1603.02754.pdf> (<https://arxiv.org/pdf/1603.02754.pdf>).
3. A Gentle Introduction to XGBoost for Applied Machine Learning - <https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/> (<https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>).
4. Introduction to XGBoost: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html> (<https://xgboost.readthedocs.io/en/latest/tutorials/model.html>).