TARAN

ADVISORY IN DATA & ANALYTICS

Data Science 2 - Trees

February 8, 2025

Data Science 2 - Trees

Faculty of Mathematics and Physics

Agenda

Decision tree based algorithms
Model and Parameters
Decision Trees
Tree ensemles
Random Forest
Gradient Boosting
Extreme Gradient Boosting
Model quality
Overfitting
Hyper-parameters optimization algorithms
Model interpretation
Other boosting implementations
Feature Engineering
Reject inference
Uplift modelling

Decision tree based algorithms

Decision tree based algorithms

- See tutorial to one of the popular machine learning models, XGBoost `https://xgboost.readthedocs.io/en/latest/tutorials/model.html`
- We will introduce the commonly known regression problems in an expanded form and notation usually used in machine learning articles

Basic principles of ML algorithms:

- Process huge amounts of data (in terms of both number of observations and attributes)
- High accuracy of the resulting models
- Full automation of the training process which should discover all relationships from the data

Model and Parameters

TARAN

ADVISORY IN DATA & ANALYTICS

Supervised learning

- *Model* refers to the mathematical structure of by which the prediction y_i is made from input $\boldsymbol{x}_i$.
- Linear model gives $\hat{y}_i = \sum_j \theta_j x_{ij}$, a linear combination of weighted input features.
- The prediction value can have different interpretations, depending on the task, i.e., regression or classification.
  - It can be logistic transformed to get the probability of positive class in logistic regression
  - It can also be used as a ranking score when we want to rank the outputs
- *parameters* or coefficients $\boldsymbol{\theta}$ are the undetermined part that we need to learn from data.
- For ML, there are often some settings or parameters of the model training itself = hyperparameters.

Training the model amounts to finding the best parameters $\boldsymbol{\theta}$ that best fit the training data $\boldsymbol{x}_i$ and labels y_i. We need to define the *objective function* to measure how well the model fit the training data. It should consist two parts: *training loss* L and *regularization term* Ω :

$$Q(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \Omega(\boldsymbol{\theta})$$

Loss functions

A common choice of L is the mean squared error, which is given by

$$L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

In logistic regression, commonly used loss function is logistic loss:

$$L(\boldsymbol{\theta}) = \sum_i -y_i \ln(\pi(\boldsymbol{x}_i)) - (1-y_i)\ln(1-\pi(\boldsymbol{x}_i)) \tag{1}$$

Another possible loss fuction could be Hinge loss:

$$L(\boldsymbol{\theta}) = \sum_i y_i \max(0, 1-\pi(\boldsymbol{x}_i)) + (1-y_i)\max(0, \pi(\boldsymbol{x}_i)) \tag{2}$$

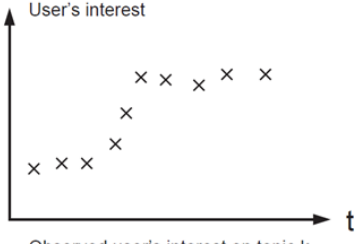
Note

Sometimes, binary target y_i is encoded as $\{-1, 1\}$, which would simplify Hinge loss equation:

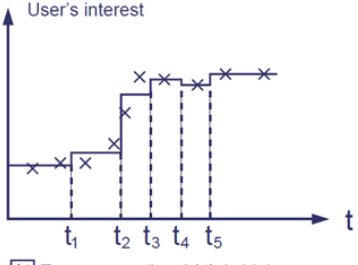
$$L(\boldsymbol{\theta}) = \sum_i \max(0, 1 - y_i\pi(\boldsymbol{x}_i))$$

Bias-Variance Tradeof

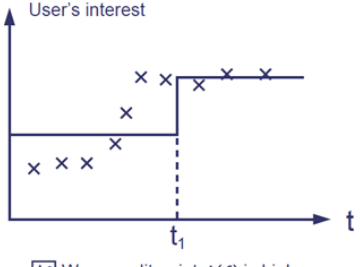
What is regularization term good for?



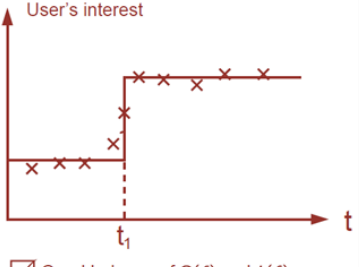
Observed user's interest on topic k against time t



☒ Too many splits, $\Omega(f)$ is high



☒ Wrong split point, $L(f)$ is high



☑ Good balance of $\Omega(f)$ and $L(f)$

Regularization
Dealing with correlation

With correlated predictors, regression coefficients can be unstable. Suppose we have two predictors x_1 and x_2 with high correlation close to equality. Let's denote logistic function with σ. Two following models can provide similar predictions and performance metrics

$$\begin{aligned} y &\sim \sigma(1x_1 + 3x_2 + \ldots) \\ y &\sim \sigma(2x_1 + 2x_2 + \ldots) \end{aligned} \tag{3}$$

Predictors' strength is distributed equally since both hold almost the same information about the target. This can be achieved by employing either $L1$ or $L2$ regularization.

$$\begin{aligned} \min_{\theta} \; & L(\boldsymbol{\theta}) + \alpha\Omega(\boldsymbol{\theta}) \\ & \Omega_{L1}(\boldsymbol{\theta}) = \sum_{i=1}^{n} |\beta_i| \\ & \Omega_{L2}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \beta_i^2 \end{aligned} \tag{4}$$

Regularization
Probabilistic derivation

Let's suppose a prior information that weights follow normal distribution. Using Bayes theorem:

$$\mathbb{P}\left[\boldsymbol{\theta}|\boldsymbol{x}, y\right] = \frac{\mathbb{P}\left[\boldsymbol{x}, y|\boldsymbol{\theta}\right]\mathbb{P}\left[\boldsymbol{\theta}\right]}{\mathbb{P}\left[\boldsymbol{x}, y\right]}$$

Now when we try to maximize a posteriori probability estimate, we can ignore the part $\mathbb{P}\left[\boldsymbol{x}, y\right]$ since it is constant with respect to $\boldsymbol{\theta}$. Let's take the prior $\boldsymbol{\theta} \sim N(0, \sigma^2 I)$:

$$\begin{aligned} &\max_{\boldsymbol{\beta}} \mathbb{P}\left[\boldsymbol{x}, y|\boldsymbol{\theta}\right]\mathbb{P}\left[\boldsymbol{\theta}\right] \\ &\max_{\boldsymbol{\beta}} \log\left\{\mathbb{P}\left[\boldsymbol{x}, y|\boldsymbol{\theta}\right]\mathbb{P}\left[\boldsymbol{\theta}\right]\right\} \\ &\max_{\boldsymbol{\beta}} \log\left\{\mathbb{P}\left[\boldsymbol{x}, y|\boldsymbol{\theta}\right]\right\} + \log\left\{\mathbb{P}\left[\boldsymbol{\theta}\right]\right\} \end{aligned} \tag{5}$$

Regularization
Probabilistic derivation

The first term in the equation is a clasicall loss part. For the second term, we arrive at $L2$ regularization:

$$\begin{aligned} \log\left\{\mathbb{P}\left[\boldsymbol{\theta}\right]\right\} &= \log\left\{\frac{\exp\left\{-\frac{1}{2}\boldsymbol{\beta}^{\top} I \boldsymbol{\beta}\right\}}{\sqrt{\left(2\pi\right)^{k}}}\right\} \\ \log\left\{\mathbb{P}\left[\boldsymbol{\theta}\right]\right\} &= \alpha \sum_{i=1}^{N} \beta_i^2 + const. \end{aligned} \tag{6}$$

If we would consider Laplacean prior, we would derive $L1$ regularization in a same way.

Regularization
Most common models

In Model training task, we have a loss part which makes model universal and regularizer with coefficient $\alpha > 0$ which fights complexity and overtraining:

$$\min_{\theta} L(\theta) + \alpha\Omega(\theta)$$

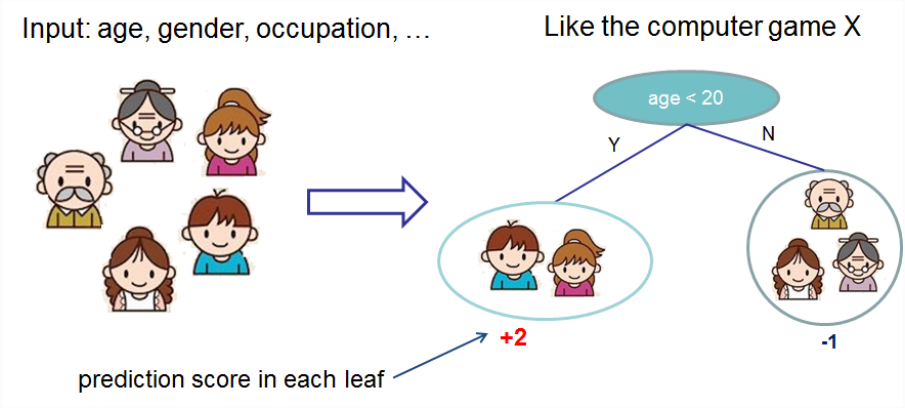
Loss	Regularizer	Model	Task
Logloss	-	Classical logistic regression	Binary classification
Logloss	L2	L2 logistic regression	Binary classification
Logloss	L1	L1 logistic regression	Binary classification
Hinge loss	L2	support vector machine	Binary classification
Squared error	L2	Ridge	Regression
Squared error	L1	Lasso	Regression
Squared error	L1+L2	Elasticnet	Regression

Decision Trees

Decision trees

classification and regression trees (CART)

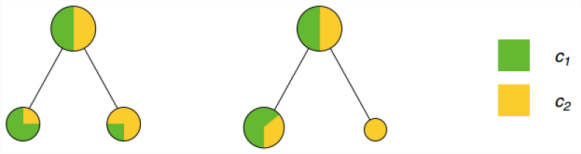
Here's a simple example of a CART that classifies whether someone will like a hypothetical computer game X.



We classify the members of a family into different leaves, and assign them the score on the corresponding leaf.

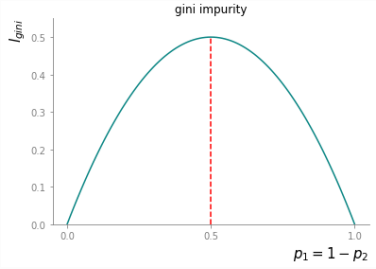
Decision Trees
Optimal split

Which split is better?



Impurity function:

- Becomes minimum at $p_1 = 0, p_2 = 1$ and $p_2 = 0, p_1 = 1$.
- Is symmetric: $l(p_1, p_2) = l(p_2, p_1)$.
- Becomes maximum at $p_1 = 0.5, p_2 = 0.5$.
- Entropy impurity function: $l_{entropy}(p_1, p_2) = -p_1 \log p_1 - p_2 \log p_2$
- Gini impurity function $l_{gini}(p_1, p_2) = 1 - p_1^2 - p_2^2 = 2p_1p_2$.



Decision trees
Summary

Iterate over all possible variables and splits, maximize gain:

$$\text{gain} = I(\text{root}) - \frac{n_{\text{left}}}{n} I(\text{left}) - \frac{n_{\text{right}}}{n} I(\text{right})$$

- Tend to underfit or overfit
- Can model non-linear dependencies
- Not sensitive to monotonic transformation of variables
- Small decision trees can be also interpreted and well understood

Tree ensemles

TARAN

ADVISORY IN DATA & ANALYTICS

Tree ensemles

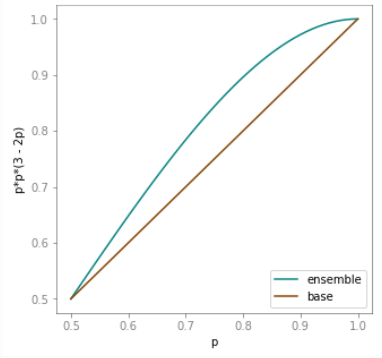
Motivation

Model ensembling is a technique used in machine learning to improve predictive power of the final model.

Let us assume that we have 3 non-correlated binary classifiers. Probability of correct answer of each of them is p.

What is the probability of correct answer of "voting" classifier?

$$p^3 + 3p^2(1-p) = p^2(3-2p)$$



Tree ensemles
Examples

- Usually, a single tree is not strong enough to be used in practice.
- Ensemble model sums the prediction of multiple trees together.
- Trees should try to complement each other

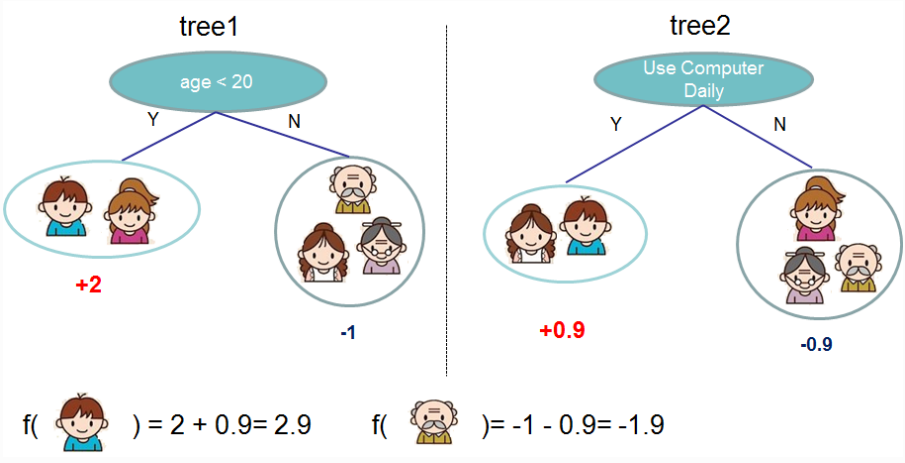


Figure: a toy example for tree ensemble, consisting of two CARTs

Tree ensemles
Notation

Mathematically, we can write our model in the form

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

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

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

Random Forest

TARAN

ADVISORY IN DATA & ANALYTICS

Random Forest
Introduction

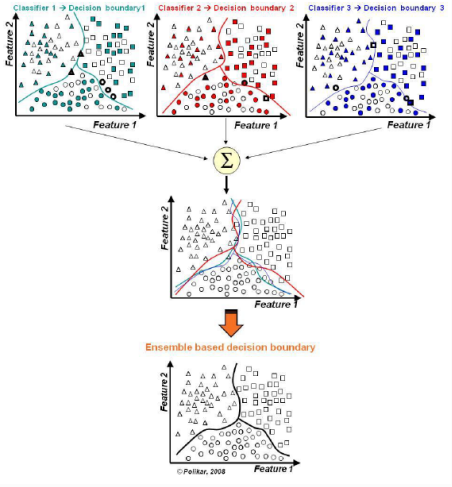
- Training multiple trees independently
- To make the trees more independent, two techniques are used: bagging and random subspaces.
- Averaging of probabilities or voting of class is used to combine decisions of all the trees
- Trees are usually deep (not restricted by depth and minimum number of samples in leave).
- Often used for their good precision and robustness with respect to outliers.
- Training can be easily parallelized and increasing the number of trees does not mean overfitting.
- Easy to use, since there are only few hyperparameters (fraction of attributes used or subsampling ratio).

Random Forest

Bagging

Bagging is a technique to prevent overfitting:

- In each iteration random subsample of given size is selected (Usually 60% for random forest).
- A new decision tree (base classifier) is trained on the subsample only.
- Influence of outliers is mitigated since they are not present in every subsample - approach emphasizes main patterns.

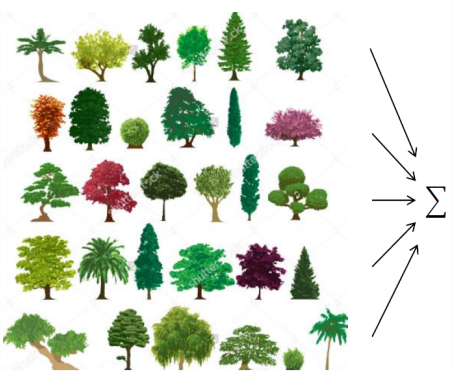


Random Forest

Random subspaces

Sampling approach applied in random subspaces for selection of the attributes:

- In each iteration randomly select attributes to be used for base classifier training.
- More attributes should be used in a final ensemble.
- Base classifiers are less correlated and ensemble should be more efficient.



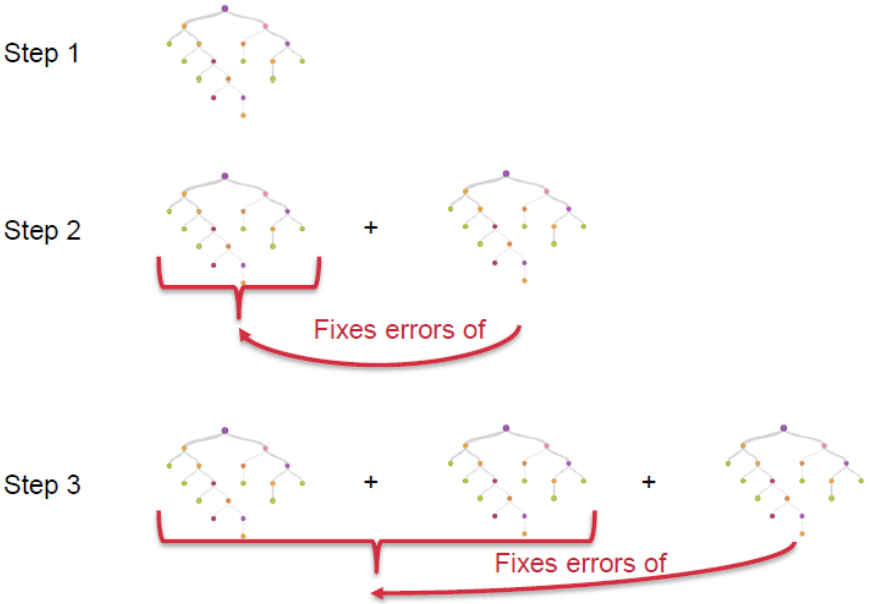
Gradient Boosting

TARAN

ADVISORY IN DATA & ANALYTICS

Gradient Boosting

Idea



Gradient Boosting
Introduction

- Trains a new decision tree on each step with respect to the already existing ensemble.
- Probabilities (or predictions) from activated leaves are summed up for all trees.
- Adding a new tree to the sum makes entire model more precise each step of the training process.
- Usually more accurate than random forest and need less trees which are not so deep.
- Training process is not so intuitively parallelizable as random forest
- Number of trees should be accurately selected to prevent overfitting.
- Multiple hyper-parameters should be also carefully selected

Gradient Boosting
Example

Let's start with an example. We have following dataset and would like to predict target variable weight:

Height(m)	Favourite Color	Gender	Weight(kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73

Before we start gradient boosting, we usually consider average of all obervations as the first prediction. Let's take 73.5 as the average and calculate residuals:

Height(m)	Favourite Color	Gender	Weight(kg)	Prediction 1	Residual 1
1.6	Blue	Male	88	73.5	14.5
1.6	Green	Female	76	73.5	2.5
1.5	Blue	Female	56	73.5	-17.5
1.8	Red	Male	73	73.5	-0.5

Gradient Boosting
Example

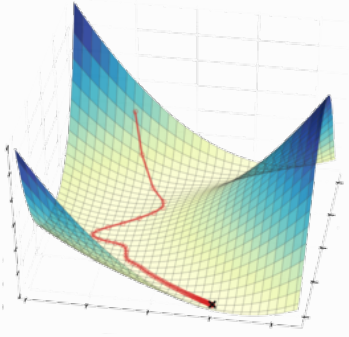
- Residuals are the new target variable which we want to explain using a decison tree.
- Consider simple decision tree with one split: if height > 1.5 then assign 5.5, else -17.5.
- The second prediction tries to fix the error of all previous predictions
- New residuals can be computed as *weight - (prediction 1 + prediciton 2)*

Height(m)	Favourite Color	Gender	Weight(kg)	Prediction 1	Prediction 2	Residual 2
1.6	Blue	Male	88	73.5	5.5	9
1.6	Green	Female	76	73.5	5.5	-3
1.5	Blue	Female	56	73.5	-17.5	0
1.8	Red	Male	73	73.5	5.5	6

- After first prediction, we had sum of absolute residuals of 35 while after the second one only 18, so the ensemble is improving.
- We can now take *residual 2* as target variable and build another decision tree and so on.

Gradient descent

Gradient descent is a method for finding minimum of complex functions.



- In each iteration we calculate the derivatives of the objective function with respect to model weights (gradient).
- Usually, learning rate lower than 1 is used. The gradient is multiplied by learning rate to be more conservative.
- Neural networks utilizes more advanced algorithms to define learning rate, that can be variable in time.

Gradient Boosting

Gradient descent

Using residuals as a target variable is feasible only for continuous regression problems. In logistic regression, it would be tricky to even calculate residual, therefore we focus on general approach:

- Minimize the loss function $Q = \sum_i^n l(y_i, \hat{y}_i)$.
- Construct a series of predictions $\hat{y}_i^{(m)}, n = 1, \ldots, M$.

In m-th iteration of gradient descent, we have:

$$\hat{y}_i^{(m)} = \hat{y}_i^{(m-1)} - b_m \nabla Q$$

$$\nabla Q = \left[\frac{\partial Q}{\hat{y}_i^{(m)-1}} (\mathbf{x}_i) \right]_{i=1}^{N} = \left[\frac{\partial l \left(y_i, \hat{y}_i^{(m-1)} \right)}{\partial \hat{y}_i^{(m)-1}} (\mathbf{x}_i) \right]_{i=1}^{N} \tag{7}$$

Gradient Boosting

Gradient descent

Similarly to our example above, we cannot use the gradient of the loss function directly. When using residuals, we have used them to define the next decision tree, which can be then used generally to any observations outside of the training set:

- ∇Q applies only to the observations in our training set.
- We need regression model which can generalize to new observations.
- Let's train a model $f_m(\boldsymbol{x}_i)$ with ∇Q as a target

Now, let's use a fastest descent by minimization of l via linear search:

$$b_m = \underset{b \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{N} l\left(y_i, \hat{y}_i^{(m-1)} - b f_m(\boldsymbol{x}_i)\right)$$

Usually, to prevent bouncing, shrinkage parameter $\nu \in (0, 1]$ is added:

$$\hat{y}_i^{(m)} = \hat{y}_i^{(m-1)} - \nu b_m f_m(\boldsymbol{x}_i)$$

Gradient Boosting
Algortihms

Gradient boosting of decision trees (GBDT) are widely used, functions f_m are decison trees. Usual hyper-parameters of each tree are: max tree depth, max children in leaf, etc.

Few of common boosting algoritms:

- LS-boosting (regression): $l(y_i, \hat{y}_i) = \frac{(y_i - \hat{y}_i)^2}{2}$
- LAD-boosting (regression): $l(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$
- AdaBoost (classification): $l(y_i, \hat{y}_i) = \exp\{-y_i\hat{y}_i - (1 - y_i)\hat{y}_i\}$
- LogitBoost (classification): $l(y_i, \hat{y}_i) = \log\{1 + \exp\{-2y_i\hat{y}_i - 2(1 - y_i)\hat{y}_i\}\}$

Let's now calculate the gradient for LS-boosting. General approach with loss function simplifies to the procedure with residuals shown in our simple example:

$$\frac{\partial l}{\partial \hat{y}_i} = \frac{\partial \frac{(y_i - \hat{y}_i)^2}{2}}{\partial \hat{y}_i} = y_i - \hat{y}_i$$

Extreme Gradient Boosting

TARAN

ADVISORY IN DATA & ANALYTICS

XGBoost
Intro

- One of mostly used boosting implementations today (the other very commonly used implementation is LightGBM).
- Many regularization features (bagging, random subspaces, $L2$-regularization, etc.).
- Popular in Kaggle competitions.
- Using model ensembles to achieve best performance.
- Base classifiers can be either tree based models or linear models.
- XGBoost can be used to predict probability of event, number of events (Poisson), continuous target.
- By our experience, usually it is the best algorithm for structured (tabular-like) data.

Nowadays, Python is the most favourite programming language for data science. XGBoost has API in several other languages like R, C++ or Julia, but we will focus on working with XGBoost in Python.

Documentation: https://xgboost.readthedocs.io/en/latest/

XGBoost
Model training

Objective function (needs to contain training loss and regularization):

$$Q = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

- Parameters are the functions f_i, containing structure of the tree and leaf scores.
- 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.
- We use an additive strategy: fix what we have learned, and add one new tree at a time.
- We write the prediction value at step t as $\hat{y}_i^{(t)}$.

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(\boldsymbol{x}_i) = \hat{y}_i^{(0)} + f_1(\boldsymbol{x}_i) \\
\hat{y}_i^{(2)} &= f_1(\boldsymbol{x}_i) + f_2(\boldsymbol{x}_i) = \hat{y}_i^{(1)} + f_2(\boldsymbol{x}_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(\boldsymbol{x}_i) = \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)
\end{aligned} \tag{8}$$

XGBoost
Training loss

$$
\begin{aligned}
Q^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)) + \Omega(f_t) + \text{constant}
\end{aligned}
\tag{9}
$$

If we consider using mean squared error (MSE) as our loss function, the objective becomes

$$
\begin{aligned}
\text{obj}^{(t)} &= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)))^2 + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(\boldsymbol{x}_i) + f_t(\boldsymbol{x}_i)^2] + \Omega(f_t) + \text{constant}
\end{aligned}
\tag{10}
$$

The form of MSE is friendly, with a first order term (usually called the residual) and a quadratic term.

XGBoost
Training loss

For other losses of interest (for example, logistic loss), it is not so easy to get such a nice form. So in the general case, we take the Taylor expansion of the loss function up to the second order:

$$Q^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i)] + \Omega(f_t) + \text{constant}$$

where the g_i and h_i are defined as

$$\begin{aligned} g_i &= \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \\ h_i &= \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \end{aligned} \tag{11}$$

XGBoost
Training loss

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

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

- This becomes our optimization goal for the new tree.
- Value of the objective function only depends on g_i and h_i, which is how XGBoost supports custom loss functions.

XGBoost
Model complexity

We need to define the complexity of the tree $\Omega(f)$.

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

Here w is the vector of scores on leaves, q is a function assigning each data point to the corresponding leaf, and T is the number of leaves.
In XGBoost, we define the complexity as

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

- There is more than one way to define the complexity, but this one works well in practice.
- Traditional treatment of tree learning only emphasized improving impurity, while the complexity control was left to heuristics.
- With regularization we obtain models that perform better in the wild.

XGBoost
Structure score

We can now write the objective value with the t-th tree as:

$$\begin{aligned} Q^{(t)} &\approx \sum_{i=1}^{n} [g_i w_{q(\mathbf{x}_i)} + \frac{1}{2} h_i w_{q(\mathbf{x}_i)}^2] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \end{aligned} \tag{12}$$

where $I_j = \{i | q(\mathbf{x}_i) = j\}$ is the set of indices of data points assigned to the j-th leaf. Notice that in the second line we have changed the index of the summation because all the data points on the same leaf get the same score.
We could further compress the expression by defining $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$:

$$Q^{(t)} = \sum_{j=1}^{T} [G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2] + \gamma T \tag{13}$$

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.

XGBoost
Structure score

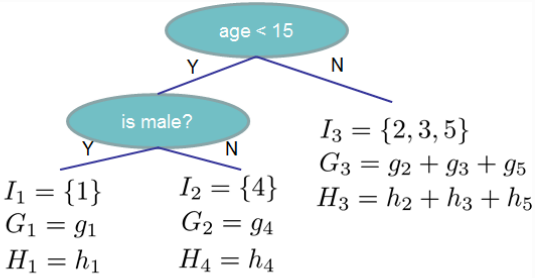
The best w_j for a given structure $q(x)$ and the best objective reduction we can get is:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$
$$Q^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T \tag{14}$$

- The last equation measures how good a tree structure $q(x)$ is.
- We push the statistics g_i and h_i 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.

XGBoost
Structure score

Instance index	gradient statistics
1 	g1, h1
2 	g2, h2
3 	g3, h3
4 	g4, h4
5 	g5, h5



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

The smaller the score is, the better the structure is

XGBoost

Learn the tree structure

- 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

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

This formula can be decomposed as:

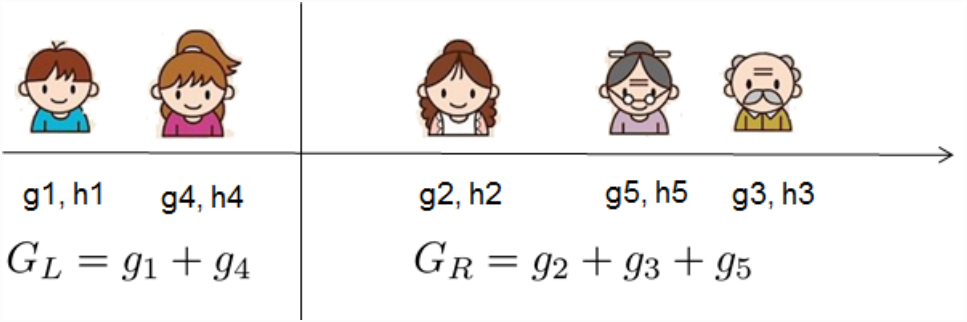
1. The score on the new left leaf
2. The score on the new right leaf
3. The score on the original leaf
4. Regularization on the additional leaf.

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!

XGBoost

Learn the tree structure

For real valued data, we usually want to search for an optimal split. To efficiently do so, we place all the instances in sorted order:



A left to right scan is sufficient to calculate the structure score of all possible split solutions. Since it is intractable to enumerate all possible tree structures, we add one split at a time.

Categorical variables

a/ Label encoding

Obs ID	Color
1	Red
2	Green
3	Blue
4	Red

Label encoding →

Obs ID	Color
1	1
2	2
3	3
4	1

b/ Dummy encoding

Obs ID	Color
1	Red
2	Green
3	Blue
4	Red

Dummy encoding →

Obs ID	Red	Green	Blue
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0

c/ Mean target encoding

$$\frac{N \cdot avg_{cat}(target) + \alpha \cdot avg(target)}{N + \alpha}$$

Missing values

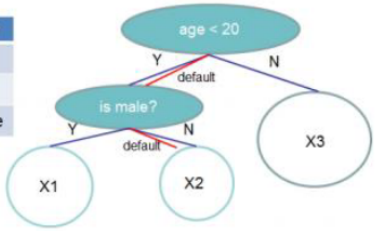
Missing values are treated automatically:

1. Make a split using non-missing values only.
2. Calculate target rate in right and left branch using non-missing values only.
3. Calculate target rate for population with missing values.
4. Assign missing values population to right or left based on lower difference in target rates.

In case of linear base model, missing values are replaced with zeros!

Data

Example	Age	Gender
X1	?	male
X2	15	?
X3	25	female



Model quality

TARAN

ADVISORY IN DATA & ANALYTICS

Continuous target

Mean squared error

Classical measures are applied also in case of ML:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2} \tag{15}$$

Log RMSE is useful to reduce the influence of outliers. But can generated problems with negative predictions (e.g. -1):

$$\text{LRMSE} = \sqrt{\frac{1}{n}\sum_{i}^{n}\left(log(\hat{y}_i + 1) - log(y_i + 1)\right)^2} \tag{16}$$

Continuous target

Mean absolute error

Classical measures are applied also in case of ML:

$$\text{MAE} = \frac{1}{n}\sum_{i}^{n}|y_i - \hat{y}_i| \tag{17}$$

And same relative metric with weighted variant:

$$\text{MAPE} = \frac{1}{n}\sum_{i}^{n}\frac{|y_i - \hat{y}_i|}{y_i} \tag{18}$$

$$\text{WMAPE} = \frac{\sum_i^n |y_i - \hat{y}_i|}{\sum_i^n |y_i|} \tag{19}$$

Classification tasks
Logistic loss

Negative log likelihood (in order to minimize it) is usually defined as logistic loss:

$$\text{logloss} = -\frac{1}{n}\sum_{1}^{n} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \tag{20}$$

- There are also versions for Poisson, Cox, Gamma, Tweedie regression, etc.

For multiclass problems:

$$\text{mlogloss} = -\frac{1}{n}\sum_{1}^{n}\sum_{1}^{K} y_{ik} \log(\hat{y}_{ik}) \tag{21}$$

Standard goodness of fit tests are not suitable for use in practice:

- For some data sets, constant prediction using average event rate can give pretty nice value of the goodness of fit
- Nice fit does not guarantee strong differentiation between target variables (0/1)
- Our predictions will be of a good quality if they differentiate well between classes of the target

Diversification power
Introduction

Denote

- K random observastion in the data set
- $\boldsymbol{x}_K$ information which is known about the observation at the time of prediction
- Y_K a random variable, which determines if the observation is good ($Y_K = 0$) or bad ($Y_K = 1$).
- $p_G = \mathbb{P}[Y_K = 0]$ and $p_B = \mathbb{P}[Y_K = 1]$.
- $s(\boldsymbol{x})$ decisioning function with range of $s(\boldsymbol{x})$ is $[0, 1]$. Greater values mean worse observation.

To assess the discriminating power of the function, we need to analyse its distribution for good and bad clients. The cumulative distribution function of good clients is called True Positive Rate (TPR, sensitivity, recall or hit rate):

$$\mathrm{TPR}(p) = F^G(p) = \mathbb{P}[s(\boldsymbol{x}) < p | Y_K = 0]. \tag{22}$$

Similarly, cumulative distribution function for the bad clients is called False Positive Rate (FPR, fall-out):

$$\mathrm{FPR}(p) = F^B(p) = \mathbb{P}[s(\boldsymbol{x}) < p | Y_K = 1]. \tag{23}$$

Diversification power

Confusion matrix

		Predicted condition		Sources: [8][9][10][11][12][13][14][15] view · talk · edit	
	Total population $= P + N$	**Predicted positive**	**Predicted negative**	Informedness, bookmaker informedness (BM) $= TPR + TNR - 1$	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
Actual condition	**Positive (P)** [a]	**True positive** (TP), hit[b]	**False negative** (FN), miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate type II error [c] $= \frac{FN}{P} = 1 - TPR$
	Negative (N) [d]	**False positive** (FP), false alarm, overestimation	**True negative** (TN), correct rejection[e]	False positive rate (FPR), probability of false alarm, fall-out type I error [f] $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$

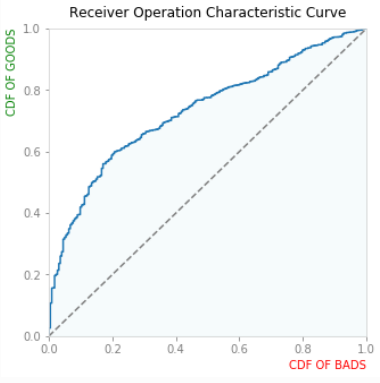
Figure: Confusion matrix with basic metrics

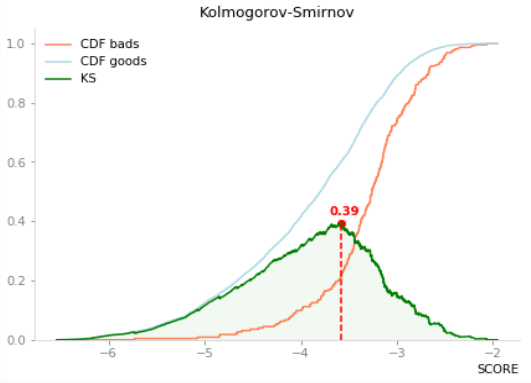
Diversification power

ROC curve

If we put values $[\text{FPR}(p), \text{TPR}(p)]$, $p \in [0, 1]$ into a chart, we get so-called Receiver operating characteristic curve (ROC curve).

- if the ROC curve would be close to identity, our discriminating power between good and bad observations is not very efficient.
- if the ROC curve would be very close to the upper left angle (maximum of x and y axis), our decisions would be strong.





Diversification power

Commonly used metrics

Definition

Integral criterion of decisioning function quality is defined by:

$$\mathrm{IK}(s) = \int_0^1 |F^G(p) - F^B(p)| \,\mathrm{d}p. \tag{24}$$

Kolmogorov-Smirnov statistic of decisioning function quality is defined by:

$$\mathrm{KS}(s) = \sup_{p \in [0,1]} |F^G(p) - F^B(p)|. \tag{25}$$

Area Under ROC curve (AUC *s*) *is given by:*

$$\mathrm{AUC}(s) = \int_0^1 \mathrm{TPR}(p) \,\mathrm{d}\,\mathrm{FPR}(p) = \int_0^1 F^G(p) \,\mathrm{d}F^B(p). \tag{26}$$

Diversification power

Commonly used metrics

Definition

Gini coefficient is defined by:

$$\begin{aligned} \mathrm{Gini}(s) &= 2\int_0^1 (\mathrm{TPR}(p) - \mathrm{FPR}(p))\,\mathrm{d}\,\mathrm{FPR}(p) \\ &= 2\int_0^1 \left(F^G(p) - F^B(p)\right)\mathrm{d}F^B(p). \end{aligned} \tag{27}$$

Let K_1 and K_2 be independent, c-statistic is the value of

$$\mathrm{c}(s) = \mathbb{P}\left[s(\boldsymbol{x}_{K_1}) \leq s(\boldsymbol{x}_{K_2}) | Y_{K_1} = 0,\ Y_{K_2} = 1\right] \tag{28}$$

Diversification power

Relationship beween metrics

Lemma

Let F^G, F^B be continuous distribution functions and let $F^G(p) \geq F^B(p)$ for all $p \in [0,1]$. Then it holds:

1. $\text{Gini} = 2\,\text{AUC} - 1$
2. $\text{AUC} = 1 - \int_0^1 F^B(p)\,\mathrm{d}F^G(p)$
3. $\text{c} = \text{AUC}$

Note

Originally Gini coefficient was defined to measure wealth inequality in population using area above so called Lorenz curve. However, Lorenz curve uses distribution of whole population on the x axis, not only distribution of bads or goods as we do in ROC curve. We therefore suggest to use ROC curve only, since by using Lorenz curve the resulting Gini coefficient might have slightly different value.

Diversification power

Proofs

1. With $\int_0^1 F^B(p)\mathrm{d}F^B(p) = \left[F^B(p)F^B(p)\right]_0^1 - \int_0^1 F^B(p)\mathrm{d}F^B(p) = \frac{1}{2}$:

$$\begin{aligned}\mathrm{Gini}(s) &= 2\int_0^1 \left(F^G(p) - F^B(p)\right)\mathrm{d}F^B(p) = 2\int_0^1 F^G(p)\mathrm{d}F^B(p) - 2\int_0^1 F^B(p)\mathrm{d}F^B(p)\\ &= 2\,\mathrm{AUC} - 1\end{aligned}$$

2. Using integration by parts:

$$\begin{aligned}\mathrm{AUC} &= \int_0^1 F^G(p)\,\mathrm{d}F^B(p) = \left[F^G(p)F^B(p)\right]_0^1 - \int_0^1 F^B(p)\,\mathrm{d}F^G(p)\\ &= 1 - \int_0^1 F^B(p)\,\mathrm{d}F^G(p)\end{aligned}$$

3. Due to the independence we have:

$$\begin{aligned}\mathrm{c} &= \mathbb{P}\left[s(\boldsymbol{x}_{\boldsymbol{K}_1}) \leq s(\boldsymbol{x}_{\boldsymbol{K}_2}) | Y_{K_1} = 0,\ Y_{K_2} = 1\right] = \int_0^1 \int_p^1 \mathrm{d}F^B(t)\,\mathrm{d}F^G(p) =\\ &= \int_0^1 \left(1 - F^B(p)\right)\mathrm{d}F^G(p) = 1 - \int_0^1 F^B(p)\mathrm{d}F^G(p) =\\ &= 1 + \mathrm{AUC} - 1 = \mathrm{AUC}\end{aligned}$$

Diversification power

Gini in practice

Take $K_1, \ldots, K_n$ randomly chosen observations with $s(\boldsymbol{x}_i),\ i = 1, \ldots, n$. We will estimate the distribution functions F^G and F^B by the empirical distribution functions. Suppose now that the observations are ordered in a way that first n_G observations are good and the rest $n_B = n - n_G$ are bad:

$$\begin{aligned} \hat{F}^G(p) &= \frac{|i : i \in \{1, \ldots, n_G\},\ s(\boldsymbol{x}_i) \leq p|}{n_G} \\ \hat{F}^B(p) &= \frac{|i : i \in \{n_G + 1, \ldots, n\},\ s(\boldsymbol{x}_i) \leq p|}{n_B} \end{aligned} \tag{29}$$

- ROC curve can be constructed by plotting $\hat{F}^B(p)$ on x axis and $\hat{F}^G(p)$ on y axis.
- The computation of Gini coefficient or other criteria can be then performed using the values from this piece-wise linear curve.
- The computation of the c-statistic can be run directly by combinatorial argument: every chosen couple of a bad and good observation has the same probability. We compute the number of couples where $s(\boldsymbol{x}_{K_G}) \leq s(\boldsymbol{x}_{K_B})$ and estimate c by the ratio of this number to the number of all couples.

Diversification power

Gini in practice

Suppose again we have scores $s(\boldsymbol{x}_i)$, $i = 1, \ldots, n$ of n observations. Denote:

- a the number of couples (i, j), $i < j$ for which $\operatorname{sgn}\left(s(\boldsymbol{x}_i) - s(\boldsymbol{x}_j)\right) = \operatorname{sgn}\left(Y_i - Y_j\right) \neq 0$, the number of cases when the good observation received better score than the bad observation.
- b the number of couples (i, j), $i < j$ for which $\operatorname{sgn}\left(s(\boldsymbol{x}_i) - s(\boldsymbol{x}_j)\right) = -\operatorname{sgn}\left(Y_i - Y_j\right) \neq 0$, the number of cases when the good observation received worse score than the bad observation.
- c the number of couples (i, j), $i < j$ for which $s(\boldsymbol{x}_i) = s(\boldsymbol{x}_j)$ and $Y_i \neq Y_j$, e.g. the number of cases when the good observation received same score as the bad observation.

It can be shown that following statistic called Somers-d equals to the Gini coefficient:

$$\mathrm{d} = \frac{a - b}{a + b + c} \tag{30}$$

Sometimes, Gini coefficient is also calculated based on the number of interchanges needed to reach the ideal state, when all the bad observations have worse score than the good observations. In random model, we have to do $c_r = \frac{1}{2} n_G n_B$ interchanges to reach the optimal ordering. Suppose number of interchanges for our model is c_m, then:

$$\mathrm{Gini} = \frac{c_r - c_m}{c_r} \tag{31}$$

Diversification power
Lift

Another useful statistic is Lift:

- indicates the ratio of the proportion of bad observations with a score of greater than p, $p \in [0,1]$, to the proportion of bad observations in the general population (or vice versa for the good observations).
- Usually, lift is computed for the worst/best α-percent of population, where it indicates how many times the prediction performs better than just random selection.

Suppose that the observations are ordered by score, e.g. $s(\boldsymbol{x}_1) \leq \ldots \leq s(\boldsymbol{x}_n)$, then:

$$\text{Lift}_\alpha = \frac{\frac{1}{\lfloor \alpha n \rfloor} \sum_{i=n-\lfloor \alpha n \rfloor}^{n} Y_i}{\frac{1}{n} \sum_{i=1}^{n} Y_i} \tag{32}$$

The level α should be set to estimated percentage of prediction cutoff.

Ranking problems
Model quality

Suppose that we convert the true values y_i into the ranks $R(y_i)$ and predictions $\hat{y}_i$ into $R(\hat{y}_i)$. Spearman's rank correlation coefficient is then defined as Pearson correlation coefficient over the ranks:

$$r_s = \rho\left(R(Y), R(\hat{Y})\right) \tag{33}$$

Kendall's tau is an extension to our calculation of Gini coefficient. We say that any pair $(y_i, \hat{y}_i)$ is concordant with $(y_j, \hat{y}_j)$ if $y_i > y_j \,\&\, \hat{y}_i > \hat{y}_j$ or $y_i < y_j \,\&\, \hat{y}_i < \hat{y}_j$. Otherwise, the pair is discordant:

$$\tau = \frac{\#\,\text{concordant} - \#\,\text{discordant}}{\binom{n}{2}} \tag{34}$$

Ranking problems
Learning to rank

Learning to rank:

- Query y: enter "ranking ml" into Google
- Document d: all the links that Google displays
- Task: Calculate relevance score of each available document d for each possible query y
- Relevance is often defined depending on the fact if user clicks or not clicks on the link shown

Precision can be then defined as:

$$\mathrm{P} = \frac{\#\text{ relevant documents retrieved}}{\#\text{ documents retrieved}} \tag{35}$$

Ranking problems
Model quality

Denote $\mathrm{P}(K)$ precision for the first K documents shown and $\mathrm{rel}(k)$ indicator if the document at rank k is relevant. Then summing over relevant documents only, with non-relevant documents retrieved having precision equal to zero:

$$\mathrm{AveP} = \frac{\sum_{k=1}^{K} \mathrm{P}(k)\,\mathrm{rel}(k)}{\#\text{ relevant documents}} \tag{36}$$

Mean average precision is calculated over all queries:

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q} \mathrm{AveP}(q)}{Q} \tag{37}$$

Ranking problems

Model quality

Suppose that rel_k denotes some grade of relevance for the document presented at position k. Then **discounted cumulative gain** for first K documents retrieved:

$$\text{DCG}_K = \sum_{k=1}^{K} \frac{\text{rel}_k}{\log_2(k+1)} \tag{38}$$

Ideal algorithm would find all the relevant documents and correctly sort them, thus arriving at **Ideal Discounted Cumulative Gain**:

$$\text{IDCG}_K = \sum_{\text{K documents with highest relevance}} \left(\frac{\text{rel}_k}{\log_2(k+1)} \right) \tag{39}$$

Finally, **Normalized Discounted Cumulative Gain**:

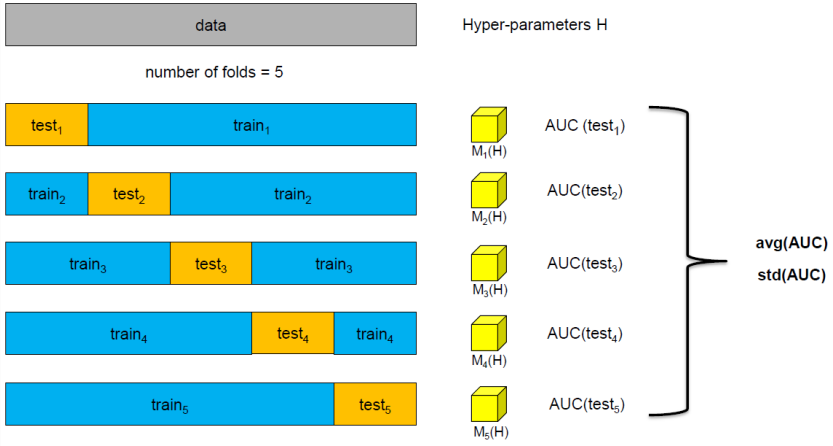
$$\text{nDCG}_K = \frac{\text{DCG}_K}{\text{IDCG}_K} \tag{40}$$

Relevance has to be usually approximated from the user interactions (clicks, etc.).

Overfitting

Cross-validation

Cross-validation is a technique that allows us to use all the data for training. This is especially useful in case of small datasets.



Cross-validation

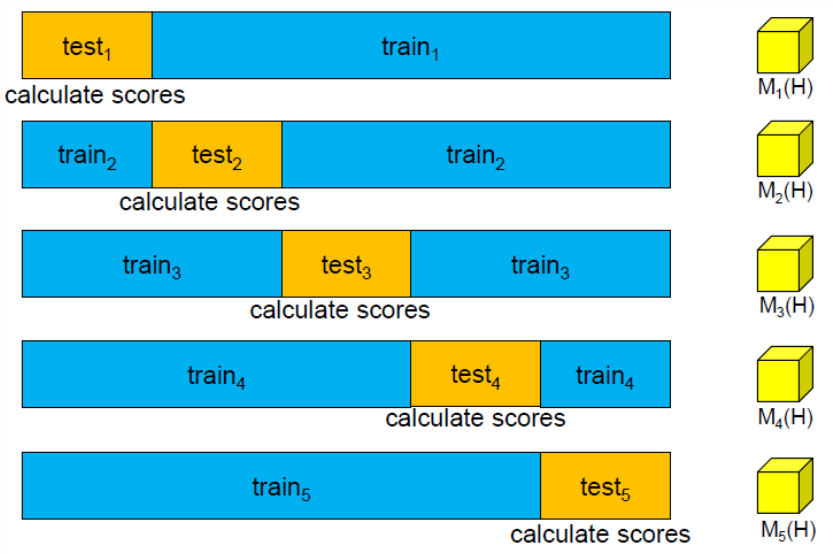
N-fold cross-validation trains N-models and returns performance on test fold on each of them.

- Average performance is estimation of the overall performance.
- Standard deviation of performance gives use information about possible variance.

Question: How should we build the final model?

Cross-validation score

Cross-validation score is a way how to extract not overfitted prediction for the whole dataset (even though the data was used for training).



Bootstrapping

Bootstrapping is an alternative way to obtain confidence interval for performance metric.

Bootstrapping uses random sampling with replacement. Let us assume that our data sample has N observations.

1. Split data for training and testing samples.
2. Re-sample training dataset using random sample with replacement.
3. Train a model on re-sampled training dataset.
4. Measure performance on testing sample.
5. Repeat the procedure M-times.

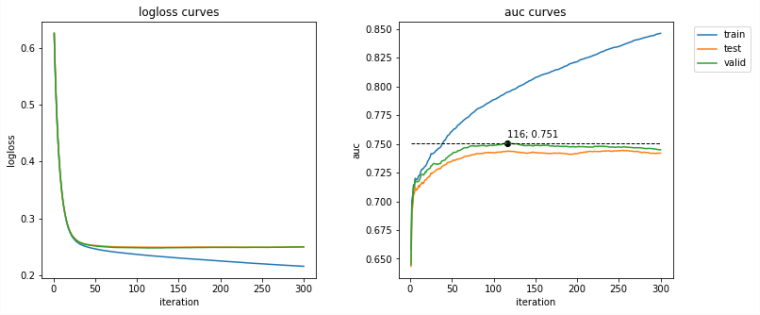
Now we can estimate average performance on test sample with confidential intervals. Such a procedure helps us to understand how the expected model performance is sensitive for selection of data sample used for training.

Bootstrapping

Monte Carlo sampling is generally useful in machine learning:

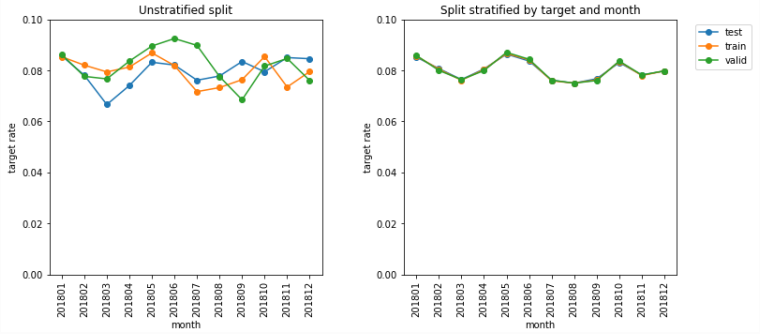
- Produce confidence intervals of model performance
- Produce confidence intervals of model predictions (subject to assumptions)
- Compare two models: instead of simple AUC comparison, bootstrap many samples and perform t-test of model A is better than model B
- Analyze predictor importance: measure marginal contribution of the predictor on bootstrapped samples
- In is used in modern ML algorithms for training the model (bagging, ensembles)

Overfitting
Training curves



- Optimal strategy is to stop training once performance on validation sample is not improving.
- Since validation sample was used for deciding when to stop training, it is not an independent sample. Therefore another sample (test) is often used in ML models.

Overfitting
Sample stratification



Stratification ensures that the data in each sample have similar distribution.

Objective functions
Binary classification

- binary:logistic

$$obj = -\frac{1}{N}\sum_{i=1}^{N}\left[label_i \ln(pred_i) + (1 - label_i)\ln(1 - pred_i)\right] + \Omega$$

prediction: probability of target

- binary:logitraw

$$obj = -\frac{1}{N}\sum_{i=1}^{N}\left[label_i \ln(pred_i) + (1 - label_i)\ln(1 - pred_i)\right] + \Omega$$

prediction: logit of probability of target (score)

- binary:hinge

$$obj = \sum_{i=1}^{N} max(0, 1 - label_i \cdot pred_i) + \Omega$$

prediction: binary classification

Objective functions
Regression

- reg:squaredederror

$$obj = \frac{1}{\sqrt{N}} \sqrt{\sum_{i=1}^{N} (pred_i - label_i)^2 + \Omega}$$

- reg:squaredlogerror

$$obj = \frac{1}{\sqrt{N}} \sqrt{\sum_{i=1}^{N} \left(\ln(pred_i + 1) - \ln(label_i + 1)\right)^2 + \Omega}$$

All input labels must be greater then -1.

Objective functions
Multiclass classification

- multi:softmax

$$obj = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} label_{ij} \cdot \ln pred_{ij} + \Omega$$

prediction: class with highest probability
note: M models with binary target (denoting each class) are fitted.

- multi:softprob

$$obj = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} label_{ij} \cdot \ln pred_{ij} + \Omega$$

prediction: probability of each class
note: M models with binary target (denoting each class) are fitted.

Objective functions
Rank

- rank:pairwise
- rank:ndcg
- count:poisson

$$obj = \frac{1}{N}\sum_{i=1}^{N}\left(\ln\Gamma(label_i + 1) + pred_i + label_i \cdot \ln pred_i\right) + \Omega$$

XGBoost parameters

General parameters

- booster [default=gbtree]
  - Which booster to use. Can be gbtree, gblinear or dart; gbtree and dart use tree based models while gblinear uses linear functions.
- verbosity [default=1]
  - Verbosity of printing messages. Valid values are 0 (silent), 1 (warning), 2 (info), 3 (debug).
- nthread [default to maximum number of threads available if not set]
  - Number of parallel threads used to run XGBoost.

XGBoost parameters

Tree booster

- eta [default=0.3, alias: learning_rate]
  - Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.
  - range: $[0, 1]$
- gamma [default=0, alias: min_split_loss]
  - Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be.
  - range: $[0, \infty)$
- max_depth [default=6]
  - Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 is only accepted in lossguided growing policy when tree_method is set as hist or gpu_hist and it indicates no limit on depth. Beware that XGBoost aggressively consumes memory when training a deep tree.
  - range: $[0, \infty)$

XGBoost parameters

Tree booster

- 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 task, this simply corresponds to minimum number of instances needed to be in each node. The larger min_child_weight is, the more conservative the algorithm will be.
  - range: $[0, \infty)$
- max_delta_step [default=0]
  - Maximum delta step we allow each leaf output to be. If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative. Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced. Set it to value of 1-10 might help control the update.
  - range: $[0, \infty)$
- subsample [default=0]
  - Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting. Subsampling will occur once in every boosting iteration.
  - range: $(0, 1]$

XGBoost parameters
Tree booster

- colsample_bytree [default=1]
  - Subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.
  - range: (0, 1]
- colsample_bylevel [default=1]
  - Subsample ratio of columns for each level. Subsampling occurs once for every new depth level reached in a tree. Columns are subsampled from the set of columns chosen for the current tree.
  - range: (0, 1]
- colsample_bynode [default=1]
  - Subsample ratio of columns for each node (split). Subsampling occurs once every time a new split is evaluated. Columns are subsampled from the set of columns chosen for the current level.
  - range: (0, 1]

XGBoost parameters
Tree booster

- lambda [default=1, alias: reg_lambda]
  - L2 regularization term on weights. Increasing this value will make model more conservative.
- alpha [default=1, alias: reg_alpha]
  - L1 regularization term on weights. Increasing this value will make model more conservative.
  - range: $[0, \infty)$
- tree_method [default=auto]
  - The tree construction algorithm used in XGBoost.
  - available values: [auto, exact, approx, hist, gpu_hist]
- monotone_constraint [default=auto]
  - Constraint of variable monotonicity.

XGBoost parameters

Learning task parameters

- objective [default=reg:squarederror]
  - Objective function to be minimized.
- base_score [default=0.5]
  - The initial prediction score of all instances, global bias
- eval_metric [default according to objective]
  - Evaluation metrics for validation data, a default metric will be assigned according to objective (rmse for regression, and logloss for classification, mean average precision for ranking).
  - User can add multiple evaluation metrics.
- seed [default=0]
  - Random number seed.

Additional parameters

Early stopping

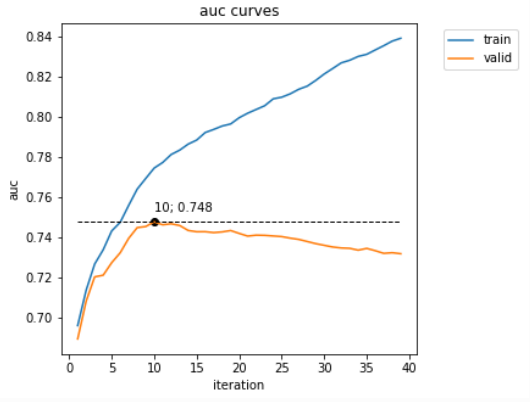
Objective function on training set improves with each iteration. When should we stop training?

Training is stopped once performance on validation sample does not improve anymore!

```
[0]     train-auc:0.69626       valid-auc:0.68963
Multiple eval metrics have been passed: 'valid-auc' will be used for early stopping.

Will train until valid-auc hasn't improved in 20 rounds.
[1]     train-auc:0.71388       valid-auc:0.70841
[2]     train-auc:0.72682       valid-auc:0.72845
[3]     train-auc:0.73391       valid-auc:0.72128
[4]     train-auc:0.74326       valid-auc:0.72749
[5]     train-auc:0.74754       valid-auc:0.73241
[6]     train-auc:0.75598       valid-auc:0.73959
[7]     train-auc:0.76404       valid-auc:0.74491
[8]     train-auc:0.76931       valid-auc:0.74544
[9]     train-auc:0.77450       valid-auc:0.74776
[10]    train-auc:0.77728       valid-auc:0.74632
[11]    train-auc:0.78124       valid-auc:0.74682
[12]    train-auc:0.78341       valid-auc:0.74596
[13]    train-auc:0.78640       valid-auc:0.74350
[14]    train-auc:0.78842       valid-auc:0.74287
[15]    train-auc:0.79215       valid-auc:0.74290
[16]    train-auc:0.79370       valid-auc:0.74244
[17]    train-auc:0.79539       valid-auc:0.74279
[18]    train-auc:0.79643       valid-auc:0.74347
[19]    train-auc:0.79952       valid-auc:0.74203
[20]    train-auc:0.80176       valid-auc:0.74070
[21]    train-auc:0.80361       valid-auc:0.74118
[22]    train-auc:0.80547       valid-auc:0.74108
[23]    train-auc:0.80889       valid-auc:0.74076
[24]    train-auc:0.80971       valid-auc:0.74046
[25]    train-auc:0.81137       valid-auc:0.73969
[26]    train-auc:0.81363       valid-auc:0.73906
[27]    train-auc:0.81520       valid-auc:0.73802
[28]    train-auc:0.81804       valid-auc:0.73696
[29]    train-auc:0.82124       valid-auc:0.73612
Stopping. Best iteration:
[9]     train-auc:0.77450       valid-auc:0.74776
```

no improvement after 20 iterations



Hyper-parameters optimization algorithms

TARAN

ADVISORY IN DATA & ANALYTICS

Hyper-parameters
Tuning algorithms

Following algorithm can be used for hyper-parameters tuning:

- Grid search.
- Randomized grid search.
- Grid search / randomized grid search with halving.
- Bayesian optimization.
- Gradient based optimization.
- Evolutionary optimization.

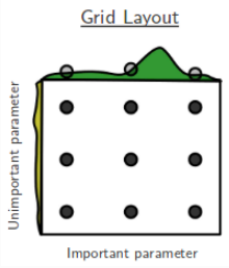
Cross validation is recommended to be used during optimization process.

Grid search

Search parameter space in selected points.

Advantages

+ Non-convex cases are handled well - no convergence to local minima.



Disadvantages

- Time consuming.
- Number of combinations grow exponentially with increasing number of parameters.
- Cannot allocate amount of resources.
- Unimportant parameters consume same amount of resources as important parameters.
- Does not reflect already learnt information.

Randomized Grid search

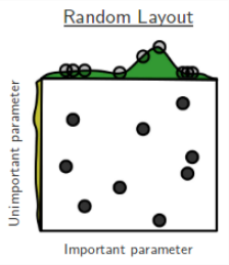
Randomly select points from search space to be investigated.

Advantages

+ Non-convex cases are handled well - no convergence to local minima.
+ Budget can be chosen independently on number of parameters.
+ Unimportant parameters do not negatively effect search process.

Disadvantages

- Time consuming.
- Does not reflect already learnt information.



Grid search with halving

Halving tackles the problem of big computation cost of hyper-parameters optimization process.

Algorithm:

1. Select initial candidates (hyper-parameter combinations) and use limited resources (number of observations) to evaluate objective function.
2. Reduce number of candidates by *factor* and increase number of resources by *factor*.
3. Continue process until all resources are used or only one candidate remains.

Resource can be number of observations, but also for instance number of decision trees used in random forest model.

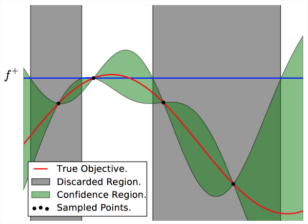
Bayesian optimization

Our prior believes about an unknown objective function are updated based on observed data. Proposition of next test point is trade off between **exploration** and **exploitation**.

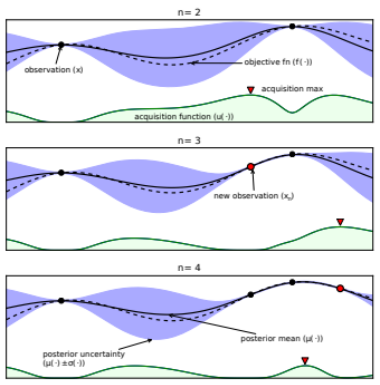
- Exploitation seeks to sample where the surrogate model predicts a good objective.
- Exploration seeks to sample in locations where the uncertainty is high.

Next test point is given by **acquisition function**. The simplest acquisition function is probability of improvement (PI):

$$PI(\boldsymbol{x}) = \phi\left(\frac{\mu(\boldsymbol{x}) - \mu^{+}}{\sigma(\boldsymbol{x})}\right)$$



Bayesian optimization



Evolutionary optimization

Generative algorithms can be used for hyper-parameters optimization.

1. Initialization - create initial population with N parameter vectors. Each vector parameter is selected randomly from search space. For each vector calculate objective function.
2. For each vector in population:
   - Mutation
     $$p_i^{mut} = p_i^{best} + F \cdot (p_i^{r_1} - p_i^{r_2})$$
     where F is a **mutation rate** and $p_i^{r_1}$ and $p_i^{r_2}$ are parameter values from randomly chosen vectors r_1 and r_2.
   - Recombination - new generation vector is created by selecting each of its items as either value from current vector or value from mutated vector. To choose between current or mutated value, random number is drawn from $[0; 1]$. If this number is lower than **recombination rate**, then current value is used, otherwise mutated value is selected.
   - Replacement - if objective function for new generation vector is lower then for current vector, then replace current vector with new generation vector.
3. Stop condition - stop evolution if standard deviation of population is lower then tolerance. Maximum number of iterations can be also selected.

General notes

Algorithm summary:

- Grid search is probably most common technique and will usually work fine.
- Randomized grid search allows you to control time of evaluation.
- Bayesian optimization can reduce consumed resources.

Always think about the problem at hand. Try to avoid meaningless combinations. Few examples:

- Target rate is 1% - restricting decision tree to have 100 observations in each leaf does not make sense.
- Having low amount of observations, decision trees with large depth will overfit.
- Make sure, you are using proper metric. For instance, accuracy is not the best choice for binary target problems with unbalanced sample.

Other notes:

- Use training sample to search for hyper-parameters. Otherwise your test set won't be independent.

Model interpretation

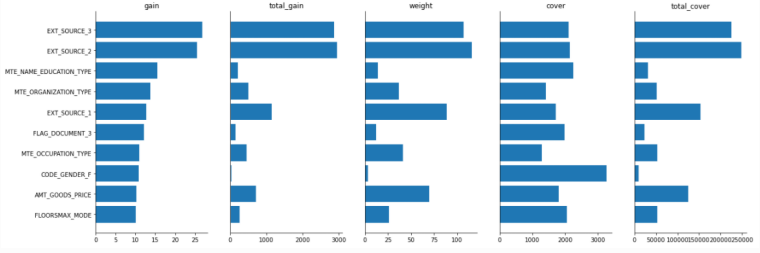
TARAN

ADVISORY IN DATA & ANALYTICS

Feature importance

XGBoost calculates several metrics that can be used to assess feature importance.

- weight: The number of times a feature is used to split the data across all trees.
- gain: The average gain across all splits the feature is used in.
- cover: The average coverage across all splits the feature is used in. Cover is defined in xgboost as the sum of second order gradient of training data classified to the leaf. If it is square loss, this simply corresponds to the number of instances in that branch. Deeper in the tree a node is, lower this metric will be
- total_gain: The total gain across all splits the feature is used in.
- total_cover: The total coverage across all splits the feature is used in.



Marginal contribution

Marginal contribution works the same for simple models and ML models.

- Assume we have a model with given strength and K predictors.
- For each predictor:
  - Fit new model with given predictor excluded.
  - Measure performance of new model.
  - Drop in performance is called marginal contribution.

Marginal contribution requires fitting of K models. If we use cross validation, we must fit $K \times n\_folds$ models.

Permutation importance

Permutation importance (PI) assess what will happen if the predictor would be broken (start returning random values).

Computation:

- For each predictor repeat n-times:
  - Randomly shuffle predictor values.
  - Calculate model performance using data with modified predictor.
  - Compare measured model strength with original model.

Permutation importance

Permutation importance (PI) assess what will happen if the predictor would be broken (start returning random values).

Computation:

- For each predictor repeat n-times:
  - Randomly shuffle predictor values.
  - Calculate model performance using data with modified predictor.
  - Compare measured model strength with original model.

Comparison with marginal contribution:

- No need to fit new models.
- PI asses predictors importance in given model, not in general.

Feature importance

Partial dependence

Partial dependence describes marginal effect of a feature on the predicted outcome.

$$f_{x_S}(x_S) = \mathbb{E}_{x_C}\left[f(x_S, x_C)\right] = \int f(x_S, x_C)\mathrm{d}\mathbb{P}(x_C)$$

where f is a machine learning model, x_s is feature for which we calculate partial dependence and x_c are other feature included in model f. Function f can be estimated by calculating averages in training data:

$$f_{x_S}(x_S) = \frac{1}{N}\sum_{i=1}^{N} f(x_S, x_C^{(i)})$$

Feature importance

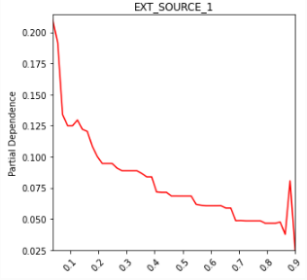
Partial dependence

Partial dependence describes marginal effect of a feature on the predicted outcome.

$$f_{x_S}(x_S) = \mathbb{E}_{x_C}\left[f(x_S, x_C)\right] = \int f(x_S, x_C)\mathrm{d}\mathbb{P}(x_C)$$

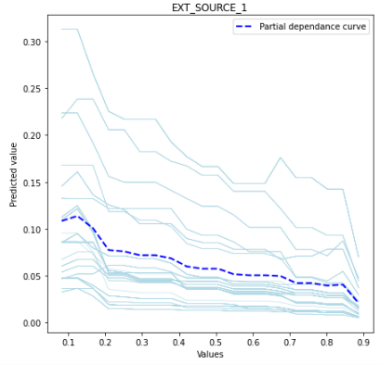
where f is a machine learning model, x_s is feature for which we calculate partial dependence and x_c are other feature included in model f. Function f can be estimated by calculating averages in training data:

$$f_{x_S}(x_S) = \frac{1}{N}\sum_{i=1}^{N} f(x_S, x_C^{(i)})$$



Feature importance
Individual Conditional Expectation

Individual Conditional Expectation (ICE) plots display one line per instance that shows how the instance's prediction changes when a feature changes.



Partial dependence plots can obscure a heterogeneous relationship created by interactions. ICE plot provides more insight.

Shapley Value

What is the feature effect on model prediction in case of linear regression?

Model prediction is given by:

$$f(x) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

Shapley Value

What is the feature effect on model prediction in case of linear regression?

Model prediction is given by:

$$f(x) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

Contribution of j-th feature is:

$$\phi_j(f) = \beta_j x_j - \mathbb{E}(\beta_j X_j)$$

where $\mathbb{E}(\beta_j X_j)$ is the mean effect estimate for feature j.

Shapley Value

In case of ML model we can calculate feature effect using Shapley value. Shapley value concept has its origin in cooperative game theory.

Let $X_1, \ldots, X_p$ be predictors of ML model f and S is a subset of features used in the model. We define value function *val* as:

$$val_x(S) = \int f(x_1, \ldots, x_p) \mathrm{d}\mathbb{P}_{x \notin S} - \mathbb{E}_X(f(X))$$

For instance, if ML model includes four features x_1, x_2, x_3 and x_4, then value function of coalition S consisting of feature values x_1 and x_3 is:

$$val_x(S) = val_x(\{x_1, x_3\}) = \int_{\mathbb{R}} \int_{\mathbb{R}} f(x_1, X_2, x_3, X_4) \mathrm{d}\mathbb{P}_{X_2 X_4} - \mathbb{E}_X(f(X))$$

The Shapley value of a feature value is:

$$\phi_j(val) = \sum_{S \subseteq \{x_1, \ldots, x_p\} \setminus \{x_j\}} \frac{|S|!(p - |S| - 1)!}{p!} \left(val(S \cup \{x_j\}) - val(S)\right)$$

Shapley Value
Properties

Shapley value satisfies following properties:

- Efficiency
- Symmetry
- Dummy
- Additivity

Efficiency

The feature contributions must add up to the difference of prediction for x and the average.

$$\sum_{j=1}^{p} \phi_j = f(x) - \mathbb{E}_X(f(x))$$

Shapley Value
Properties

Symmetry

The contributions of two feature values j and k should be the same if they contribute equally to all possible coalitions. If

$$val(S \cup \{x_j\}) = val(S \cup \{x_k\})$$

for all $S \subseteq \{x_1, \ldots, x_p\} \setminus \{x_j, x_k\}$, then

$$\phi_j = \phi_k$$

Dummy

A feature j that does not change the predicted value – regardless of which coalition of feature values it is added to – should have a Shapley value of 0. If

$$val(S \cup \{x_j\}) = val(S)$$

for all $S \subseteq \{x_1, \ldots, x_p\}$, then

$$\phi_j = 0$$

Shapley Value
Properties

Additivity

For a game with combined payouts $val + val^+$ the respective Shapley values are as follows:

$$\phi_j + \phi_j^+$$

Shapley Value
SHAP estimation

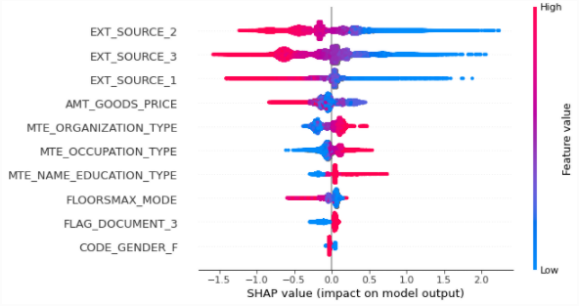
Number of possible coalition grows exponentially with number of features. Thus Shapley values are estimated using following approximation:

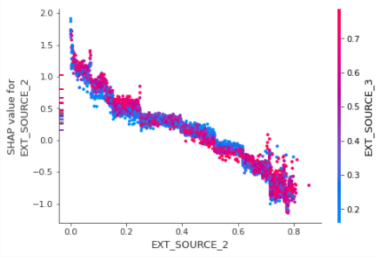
$$\hat{\phi}_j = \frac{1}{M} \sum_{m=1}^{M} \left(f(x_{+j}^m) - f(x_{-j}^m) \right)$$

where $f(x_{+j}^m)$ is the prediction for x with random number of feature values replaced by feature values from random data sample. Feature x_j is not replaced though. x_{-j}^m is the same as x_{+j}^m, but with x_j also replaced with randomized data.

For more details check:
https://christophm.github.io/interpretable-ml-book/shapley.html

Shapley Value
SHAP plots





Other boosting implementations

TARAN

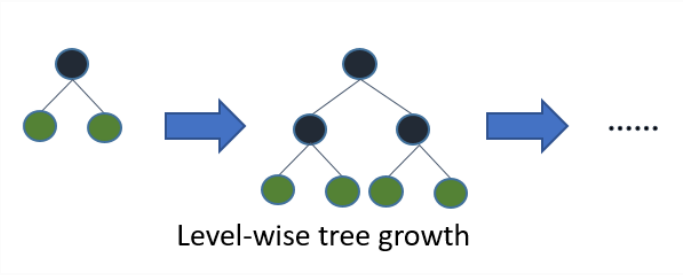
ADVISORY IN DATA & ANALYTICS

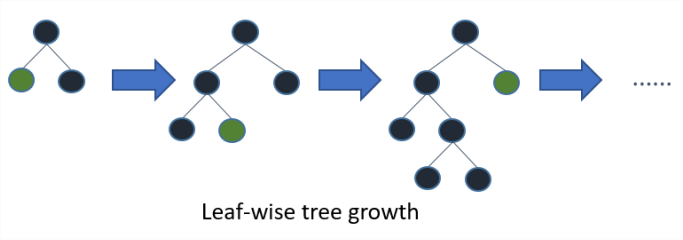
LightGBM

- Uses leaf-wise splitting instead of level-wise splitting
  - Needs less trees for the same accuracy
  - Keep eye on hyperparameters to reduce overfitting
- Support for categorical variables
  - Sort the categories according to its training objective (sum_gradient / sum_hessian) and then finds the best split on the sorted histogram
- More efficient implementation (GPU utilization, usage of histogram for splitting)
  - In XGBoost, similar method hast been implemented (Tree method: 'hist')
- Gradient-Based One-Side Sampling (GOSS)
  - To improve the accuracy, GOSS retains instances with larger gradients and performs random sampling on instances with smaller gradients.
- Exclusive Feature Bundling (EFB)
  - Many features rarely take non-zero values simultaneously (e.g. one-hot encoding). EFB bundles these features to improve efficiency.

XGBoost vs LightGBM

Tree growth approach





LightGBM parameters

Main hyperparameters

- max_depth [default=-1]
  - Maximum depth of a tree, can be unlimited (numbers below 0). Can be used to prevent overfitting.
- num_leaves [default=31]
  - Important parameter in terms of controlling the complexity of the tree.
- min_data_in_leaf [default=20]
  - Minimal number of observations in a leaf, again, approximated by Hessian. As rule of thumb, use hundreds or thousands.
- feature_fraction [default=1]
  - Subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.
  - range: $(0, 1]$
- bagging_fraction [default=1]
  - Subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.
  - range: $(0, 1]$

CatBoost

- Uses symmetric (oblivious) trees instead of assymetric ones
  - Ensures further regularization, while it might take more trees to fit
- Ordered boosting
  - Permutation-driven approach to train model on a subset of data while calculating residuals on another subset, thus preventing target leakage and overfitting
- Support for categorical variables
  - One-hot encoding for features with less than one_hot_max_size categories (parameter to be setup)
  - Mean target encoding (with random permutation)
  - Greedy seach for combinations (combines $2 - 3$ features usually)
- Text features with preprocessing (Bad of Words, Naive Bayes, etc.)
- Extended support for ranking objectives

XGBoost vs. CatBoost

Tree growth approach

Figure: Classical trees

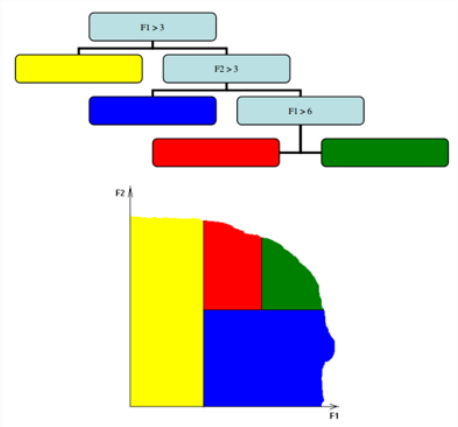
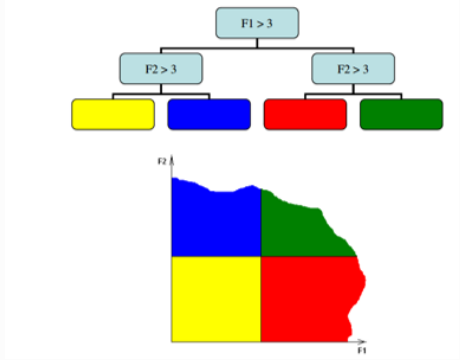


Figure: Oblivious trees



Feature Engineering

Overview

Feature engineering is an attempt to make data better suited to the problem at hand.

- increase model predictive strength
- reduce computational expenses
- improve interpretability

Techniques:

- feature transformations
- extraction of specific information (year from date)
- group transform
- feature combinations (interactions)
- clustering
- PCA
- social network

Univariate feature performance

At the beginning of FE process, measure feature univariate strength:

- correlation with target
- mutual information
- other metrics such as AUC coefficient

The actual usefulness of a feature depends on the model you use it with.

Feature preparation

Raw data must be aggregated to features.

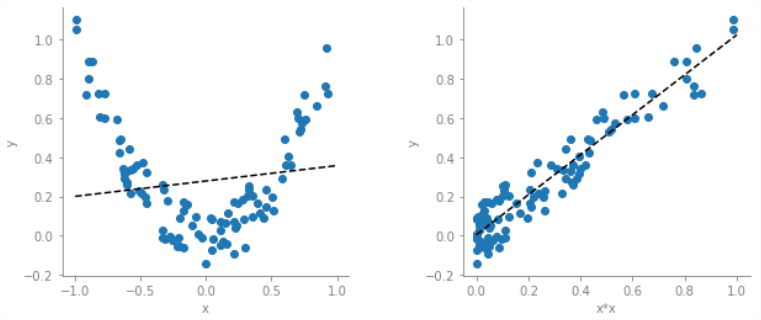
- Different aggregation functions.
- Different time windows.
- Different segmentations.

Few tips:

- Brute force feature preparation can lead to overwhelming the model. Feature selection might be used to tackle the problem.
- Try to create reasonable features only.
- Think about the problem at hand and derive features that might be suitable.

Feature transformation

Features should be transformed to the shape suitable for model.



Transformation examples:

- Linearization for linear models.
- Logarithmic transformation to deal with skewed distributions.
- Relativisation (by time window for instance).
- Conversion of date to duration.

Feature extraction

New features can be created by extracting part of information from existing predictor.

- From timestamp extract hour.
- From date extract day of week.
- From email extract email domain.
- From phone number extract dialling code.

Group transform

Group transforms calculates a value for a group. For instance, average income in region.

- Income 30k can be above average in one region, while below average in different region.
- Credit amount 75k can be above average for credit card and below average for cash loan.

Time relativisation might be also useful.

- Interest rate can be compared with average rate during relevant time period.

Feature combinations

It makes sense to interact features:

- meaningful relationship
  - education x income
  - age x worklength
  - number of applications x time since last application
- correlated predictors
  - max dpd 30d x max dpd 180d
  - number of applications x number of rejected application
- same origin predictors
  - socdem predictors
  - previous applications
- predictors with high interaction potential according to algorithm (SHAP for instance)

How to create interaction:

- manual interaction
- fraction / multiplication
- clustering
- PCA

Clustering

Clustering can be used to detect groups with similar characteristics.

- Locations with increased risk.
- Clients with similar spending strategy.
- Clients with similar products.

Few technical notes:

- Clustering can be supervised of unsupervised.
- Normalize values before clustering.
- Use single digit number of features for clustering.

Principal component analysis

PCA transforms set of feature into new set of uncorrelated features.

- Dimensionality reduction
- Decorrelation
- New features are linear combinations of old features.
- Magnitudes and signs of coefficients in linear combination describes what the new feature consists of.

Technical note:

- PCA is typically applied to standardized data.

Reject inference

TARAN

ADVISORY IN DATA & ANALYTICS

Reject inference

Introduction

What is Reject Inference? Application of some predictive model or rules can lead to censoring of the target variable:

- In credit risk, you can observe repayment of the loan only on granted/approved loans
- In CRM, you observe customer reaction to the offer only when you make the offer (make a call, send SMS, etc.)
- In AntiFraud, if you ban someone, you do not know how much loss he would generate

Effects of reject inference

- Development sample contains only censored observations
- Filtering out "bad" observations can affect predictor behaviour
- New model reflects biased patterns

How to evaluate impact of Reject Inference?

- Explore predictor binning on censored population
- Use current score to estimate probability on censored population
- Compare transition matrices for development and censored population

Reject inference
Techniques

How to deal with Reject inference?

1. Fantomas
   - Do not censor small share of observations selected as "bad" by the model or process
   - Reweight “fantomased” observations to simulate censored population
2. Artificial target
   - Generate artificial target based on prediction of current score
   - Generate artificial target based on prediction of score trained on development sample only
   - Use proxy data to define target variable (for example Credit Bureau in credit risk, response on other products in CRM, etc.)

Uplift modelling

Uplift modelling
Introduction

Aims to model true effect of some treatment:

- Sending marketing SMS to open savings account
- Reminder to do not forget about loan repayment
- Discounted rate offered on mortgage

Usually, 4 groups of customers are distinguished:

- The Persuadables: customers who only respond to the marketing action because they were targeted
- The Sure Things: customers who would have responded whether they were targeted or not
- The Lost Causes: customers who will not respond irrespective of whether or not they are targeted
- The Do Not Disturbs: customers who are less likely to respond because they were targeted

Uplift modelling
Performance metrics

Standard measure determines performance of the score on first k observations. Suppose we have observations sorted by the uplift score and determine T_K those of the first K belonging in the treatment group and C_K respectively:

$$\text{Uplift}_K = \frac{\sum_{k \in T_K} y_k}{|T_K|} - \frac{\sum_{k \in C_K} y_k}{|C_K|} \tag{41}$$

- Now put $K = \lfloor \alpha * N \rfloor$ and define $\text{Uplift}_\alpha = \text{Uplift}_{\lfloor \alpha * N \rfloor}$
- Uplift curve can be defined as points $[\alpha, \text{Uplift}_\alpha]$, $\alpha \in [0, 1]$.
- When we put $\alpha = 1$, we receive the total difference in conversion rate between treatment and control group: $t_{eff} = \frac{\sum_{k \in T} y_k}{|T|} - \frac{\sum_{k \in C} y_k}{|C|}$
- For random model, uplift curve is a straight line between $[0, 0]$ and $[1, t_{eff}]$

Uplift modelling
Performance metrics

We can define Qini coefficient Q using the uplift curve. Sometimes, this is called AUUC, area under uplift curve:

$$Q = \int_0^1 \text{Uplift}_\alpha - \alpha * t_{eff} \, \mathrm{d}\alpha \tag{42}$$

- The area under this curve generally depends on the size of the treatment effect in the data set
- Various normalizations can be defined based on the area under uplift curve for ideal model
- Other metrics like Cumulative Gain or Adjusted Qini are also used to overcome possible overfitting issues

$$\text{CumulativeGain}_K = \left(\frac{\sum_{k \in T_K} y_k}{|T_K|} - \frac{\sum_{k \in C_K} y_k}{|C_K|} \right) (|T_K| + |C_K|) \tag{43}$$

Uplift modelling
Standard approach

Include the treatment flag in the set of predictors for a predictive model:

- do not use linear model!

Build two models:

- $M_{\text{treatment}}$ predicts the event rate (conversion, default, etc.) on the treatment group
- M_{control} predicts the event rate on control group

The effect of the treatment is then estimated as $M_{\text{treatment}} - M_{\text{control}}$

However, this approach has some pitfalls:

- Models aim to predict the event, not the effect of treatment
- Overfitting can cause misleading results (each client is in one of the groups)
- Multiple models are harder to manage

Uplift modelling
Proxy approach

- Denote target variable Y and $G = T$ for treatment group, $G = C$ for control group
- Denote $\mathbb{P}^T$ probability of event if the treatment is applied, $\mathbb{P}^C$ probability in control group
- Goal is to predict $\mathbb{P}^T[Y = 1] - \mathbb{P}^C[Y = 1]$
- Define proxy variable Z: $Z = 1$ if $G = T$ & $Y = 1$ or $G = C$ & $Y = 0$. $Z = 0$ otherwise

$$\begin{aligned}
\mathbb{P}[Z = 1|X_1, \ldots, X_n] &= \mathbb{P}[Z = 1|X_1, \ldots, X_n, G = T]\,\mathbb{P}[G = T|X_1, \ldots, X_n] \\
&\quad + \mathbb{P}[Z = 1|X_1, \ldots, X_n, G = C]\,\mathbb{P}[G = C|X_1, \ldots, X_n] \\
&= \mathbb{P}[Y = 1|X_1, \ldots, X_n, G = T]\,\mathbb{P}[G = T] + \mathbb{P}[Y = 0|X_1, \ldots, X_n, G = C]\,\mathbb{P}[G = C] \\
&= \mathbb{P}^T[Y = 1|X_1, \ldots, X_n]\,\mathbb{P}[G = T] + \mathbb{P}^C[Y = 0|X_1, \ldots, X_n]\,\mathbb{P}[G = C]
\end{aligned}$$

Uplift modelling
Proxy approach

Assume $\mathbb{P}[G = T] = \mathbb{P}[G = C] = \frac{1}{2}$:

$$\begin{aligned} 2\mathbb{P}[Z = 1|X_1, \ldots, X_n] &= \mathbb{P}^T[Y = 1|X_1, \ldots, X_n] + \mathbb{P}^C[Y = 0|X_1, \ldots, X_n] \\ &= \mathbb{P}^T[Y = 1|X_1, \ldots, X_n] + 1 - \mathbb{P}^C[Y = 1|X_1, \ldots, X_n] \end{aligned}$$

Which leads to $\mathbb{P}^T[Y = 1|X_1, \ldots, X_n] - \mathbb{P}^C[Y = 1|X_1, \ldots, X_n] = 2\mathbb{P}[Z = 1|X_1, \ldots, X_n] - 1$

- Balance of the groups can be achieved by oversampling
- Single model approach which finds the treatment effect rather then event prediction

Uplift modelling

Uplift decision tree

We can use special splitting criteria when constructing our decision tree:

- If probability $\mathbb{P}^T$ and $\mathbb{P}^C$ are the same in the leaves, then effect of splitting is zero
- We can use divergence metrics to define split quality

Suppose two distributions $P = (p_1, \ldots, p_N)$ and $Q = (q_1, \ldots, q_N)$:

$$\mathrm{KL}(P, Q) = \sum_{i=1}^{N} p_i \log \frac{p_i}{q_i}$$

$$\mathrm{E}(P, Q) = \sum_{i=1}^{N} (p_i - q_i)^2$$

$$\chi^2(P, Q) = \sum_{i=1}^{N} \frac{(p_i - q_i)^2}{q_i}$$

Uplift modelling

Uplift decision tree

In our case, we have two classes, $y_i = 1$ and $y_i = 0$, so for E measure:

$$E(\mathbb{P}^T, \mathbb{P}^C) = (\mathbb{P}[Y = 1|G = T] - \mathbb{P}[Y = 1|G = C])^2 + (\mathbb{P}[Y = 0|G = T] - \mathbb{P}[Y = 0|G = C])^2$$

Now we can define gain similarly as in decision trees, for instance with KL measure:

$$\text{gain} = KL(\mathbb{P}^T, \mathbb{P}^C|\text{root}) - \frac{n_{\text{left}}}{n} KL(\mathbb{P}^T, \mathbb{P}^C|\text{left}) - \frac{n_{\text{right}}}{n} KL(\mathbb{P}^T, \mathbb{P}^C|\text{right})$$

- If these measures lead to leaves with large imbalance between treatment and control groups (possible overfitting), we can enhance our fuction to include penalty for uneven splitting using Gini criterion or Euclidian one.

Thank you!

TARAN

ADVISORY IN DATA & ANALYTICS