2.1 Boosting - wisdom of the crowd (theory)

February 22, 2017

1 Boosting - Wisdom of the Crowd (theory)

What you will learn: - What is the idea of boosting - Why use tree as a weak classifier - What are some common boosting implementations - How XGBoost helps

1.0.1 Idea of boosting

Let's start with intuitive definition of the concept: > **Boosting** (*Freud and Shapire, 1996*) - algorithm allowing to fit **many** weak classifiers to **reweighted** versions of the training data. Classify final examples by majority voting.

When using boosting techinque all instance in dataset are assigned a score that tells *how difficult to classify* they are. In each following iteration the algorithm pays more attention (assign bigger weights) to instances that were wrongly classified previously.

In the first iteration all instance weights are equal.

Ensemble parameters are optimized in **stagewise way** which means that we are calculating optimal parameters for the next classifier holding fixed what was already calculated. This might sound like a limitation but turns out it's a very resonable way of regularizing the model.

1.0.2 Weak classifier - why tree?

First what is a weak classifier? > **Weak classifier** - an algorithm **slightly better** than random guessing.

Every algorithm can be used as a base for boosting techinique, but trees have some nice properties that makes them more suitable candidates.

Pro's

- computational scalability,
- handling missing values,
- robust to outliers,
- does not require feature scalling,
- can deal with irrelevant inputs,
- interpretable (if small),
- can handle mixed predictors (quantitive and qualitative)

Con's

- can't extract linear combination of features
- small predictive power (high variance)

Boosting techinque can try to reduce the variance by **averaging** many **different** trees (where each one is solving the same problem)

1.0.3 Common Algorithms (warning MATH INCLUDED)

In every machine learning model the training objective is a sum of a loss function L and regularization Ω:

$$obj = L + \Omega$$

The loss function controls the predictive power of an algorithm and regularization term controls it's simplicity.

AdaBoost (Adaptive Boosting) The implementation of boosting technique using decision tress (it's a *meta-estimator* which means you can fit any classifier in). The intuitive recipie is presented below:

Algorithm:

Assume that the number of training samples is denoted by N, and the number of iterations (created trees) is M. Notice that possible class outputs are $Y = \{-1, 1\}$

1. Initialize the observation weights $w_i = \frac{1}{N}$ where $i = 1, 2, \ldots, N$
2. For $m = 1$ to M:
   - fit a classifier $G_m(x)$ to the training data using weights w_i,
   - compute $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x))}{\sum_{i=1}^{N} w_i}$,
   - compute $\alpha_m = \log((1 - err_m)/err_m)$,
   - set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x)]$, where $i = 1, 2, \ldots, N$
3. Output $G_m(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$

Generalized Boosted Models We can take advantage of the fact that the loss function can be represented with a form suitable for optimalization (due to the stage-wise additivity). This creates a class of general boosting algorithms named simply **generalized boosted model (GBM)**.

An example of a GBM is **Gradient Boosted Tree** which uses decision tree as an estimator. It can work with different loss functions (regression, classification, risk modeling etc.), evaluate it's gradient and approximates it with a simple tree (stage-wisely, that minimizes the overall error).

AdaBoost is a special case of Gradient Boosted Tree that uses exponential loss function. You can learn more about GBM in this video.

1.0.4 How XGBoost helps

The problem with most tree packages is that they don't take regularization issues very seriously - they allow to grow many very similar trees that can be also sometimes quite bushy.

GBT tries to approach this problem by adding some regularization parameters. We can: - control tree structure (maximum depth, minimum samples per leaf), - control learning rate (shrinkage), - reduce variance by introducing randomness (stochastic gradient boosting - using random subsamples of instances and features)

But it could be improved even further. Enter XGBoost.

> **XGBoost** (*extreme gradient boosting*) is a **more regularized** version of Gradient Boosted Trees.

It was develop by Tianqi Chen in C++ but also enables interfaces for Python, R, Julia. Used for supervised learning problem gave win to many Kaggle competitions.

The main advantages: - good bias-variance (simple-predictive) trade-off "out of the box", - great computation speed, - package is evolving (author is willing to accept many PR from community)

XGBoost's objective function is a sum of a specific loss function evaluated over all predictions and a sum of regularization term for all predictors (K trees). In the formula f_k means a prediction coming from k-th tree.

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

Loss function depends on the task being performed (classification, regression, etc.) and a regularization term is described by the following equation:

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

First part (γT) is responsible for controlling the overall number of created leaves, and the second term ($\frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$) watches over the their's scores.

To optimize the objective a gradient descent is used, this leads to a problem of finding an optimal structure of the successive tree. More mathematics about the algorithm is not included in the scope of this course, but pretty decent informations can be found on the package docs page and in this presentation.