QBUS6850 Week 9

Ensemble Methods 3 - Gradient Boosting

Dr Stephen Tierney

The University of Sydney Business School

The Plan

- Part 1 (Week 6): Understand decision trees and How to make an ensemble (forests)
- Part 2 (Week 8): Advanced ensembles 1 (boosting)
- **Part 3 (Week 9): Advanced ensembles 2 (gradient boosting)**

Reading

- Sections 1, 2, 3, 4.1-4.3 and 4.5 from Greedy Function Approximation: A Gradient Boosting Machine, Jerome H. Friedman
- Chapter 10.9 - 10.13, The Elements of Statistical Learning, Hastie et al.

Gradient Boosting - Concept

Gradient Boosting takes a different strategy to AdaBoost and related techniques.

Instead of re-weighting the data to fix mistakes, gradient boosting operates directly on the mistakes!

Gradient Boosting - Concept

Suppose we have trained a regression model $f(\mathrm{x})$. Our regression model is good but not perfect.

This means we have a prediction error i.e.

$$f(\mathrm{x}) - \mathrm{y} = \mathrm{e}$$

where e is the vector of residuals.

Gradient Boosting - Concept

Re-arranging we have

$$f(x) - y = e$$
$$f(x) - e = y$$

This means that we could get to y by adding something to $f(x)$!

Gradient Boosting - Concept

Re-arranging we have

$$f(x) - y = e$$
$$f(x) - e = y$$

This means that we could get to y by adding something to $f(x)$![1]

$$f(x) + h(x) = y$$

[1] this thing might be positive or negative depending on whether we over or under estimated y

Gradient Boosting - Concept

We train $h(x)$ so that it predicts e, the residuals, from our first model.

We use a model $h(x)$ (not just a fixed value) because e changes depending on x.

Since our extra model $h(x)$ isn't perfect either, we end up with

$$(f(x) + h(x)) - y = e_1$$

where e_1 are the new residuals.

Gradient Boosting - Concept

We can continue to add models to the ensemble $F(x)$

$$F_1(\mathrm{x}) = F_0(\mathrm{x}) + h_1(\mathrm{x})$$
$$F_2(\mathrm{x}) = F_1(\mathrm{x}) + h_2(\mathrm{x})$$
$$\dots$$
$$F_M(\mathrm{x}) = F_{M-1}(\mathrm{x}) + h_M(\mathrm{x})$$

up to M additional models and F_0 is just a single model, while F_i for $i \geq 1$ are ensembles.

Each additional model $h_m(\mathrm{x})$ is trained to predict the residuals e_{m-1} of the previous ensemble $F_{m-1}(\mathrm{x})$.

The Details

The Details

This section focuses on the details for regression problems since it greatly simplifies the mathematics.

For details with particular loss functions you are expected to do the first reading.

Gradient Descent

Typically when building a model we choose a particular model f with parameters θ and then set

$$\theta^* = \arg\min_{\theta} \sum_{i}^{n} L(y_i, f(x_i))$$

where θ^* is our estimate of θ and L is a loss function e.g. squared error or binary cross entropy.

Gradient Descent

Solving

$$\arg\min_{\theta} \sum_{i}^{n} L(y_i, f(x_i))$$

is commonly done using Gradient Descent (also known as Steepest Descent).

Functional Gradient Descent

Instead of choosing a parameterised function we could optimise over the choice of functions themselves i.e.

$$f^* = \arg\min_f \sum_i^n L(y_i, f(\mathrm{x}_i))$$

Functional Gradient Descent

Applying gradient descent to this function optimisation problem gives

$$F_m(\mathrm{x}) = F_{m-1}(\mathrm{x}) - \alpha_m \frac{\partial L}{\partial F_{m-1}}$$

Residuals

Since we are evaluating over a fixed dataset the gradient is defined element wise for each data point x_i

$$\frac{\partial L(y_i, F_{m-1}(\mathrm{x}_i))}{\partial F_{m-1}(\mathrm{x}_i)} = g_m(\mathrm{x}_i)$$

Usually all $g_m(\mathrm{x}_i)$ are combined into a vector $g_m(\mathrm{x}_i)$ and called "pseudo-residuals".

Residuals

In the case of a squared error loss $L(y, F(\mathrm{x})) = \frac{1}{2}(y - F(\mathrm{x}))^2$ the residuals are:

$$\begin{aligned} \frac{\partial L(y_i, F_{m-1}(\mathrm{x}_i))}{\partial F_{m-1}(\mathrm{x}_i)} &= \frac{\partial \frac{1}{2}(y - F(\mathrm{x}))^2}{\partial F_{m-1}(\mathrm{x}_i)} \\ &= -y_i + F_{m-1}(\mathrm{x}_i) \end{aligned}$$

Motivation

This idea is motivated by the fact that we can express the additive process as

$$F_m(\mathrm{x}) = F_{m-1}(\mathrm{x}) + \arg\min_{h,\alpha} \left[\sum_i^n L(y_i, F_{m-1}(\mathrm{x}_i) + \alpha h(\mathrm{x}_i)) \right]$$

Which can be simplified to

$$F_m(\mathrm{x}) = F_{m-1}(\mathrm{x}) - \alpha_m g_m(\mathrm{x})$$

The Problem

Using the pseudo-residuals to directly step towards the solution will almost guarantee loss of generalisation.

The learnt function is only be useful to predict the specific training points used.

We won't learn a function that generalises to unseen data!

The Solution

To get around this we fit a model h_m to the residuals g_m.

Mathematically we find h_m by solving

$$h_m = \arg\min_h \sum_i^n (g_m(x_i) - h_m(x_i))^2$$

Normally for gradient boosting we will use regression trees or stumps for h_m.

In other words we find the tree that fits best to $g_m(x_i)$.

Step Size

After finding $h_m(x_i)$ we then can then find α_m by solving

$$\alpha_m = \arg\min_{\alpha} \left[\sum_{i}^{n} L(y_i, F_{m-1}(x_i) - \alpha h_m(x_i)) \right]$$

Algorithm

Input: training data and a loss function L

1. Initialise first model

$$F_0(\mathrm{x}) = \arg\min_p \sum_i^n L(y_i, p)$$

2. Repeat until stopping criteria reached (e.g. maximum iterations)

2.1 Compute residuals

$$g_m(\mathrm{x}) = \frac{\partial \sum_i^n L(y_i, F_{m-1}(\mathrm{x}_i))}{\partial F_{m-1}(\mathrm{x}_i)}$$

2.2 Fit h_m

$$h_m = \arg\min_h \sum_i^n (g_m(\mathrm{x}_i) - h(\mathrm{x}_i))^2$$

2.3 Update step size (optional, can use fixed size instead)

$$\alpha_m = \arg\min_\alpha \left[\sum_i^n L(y_i, F_{m-1}(\mathrm{x}_i) - \alpha h_m(\mathrm{x}_i)) \right]$$

2.4 Update model

$$F_m(\mathrm{x}) = F_{m-1}(\mathrm{x}) - \alpha_m h_m(\mathrm{x})$$

Why Is It Called Gradient Boosting?

In essence we are using gradient descent to update our model.

Except we are searching in "function space", rather than "parameter space" for the next best position.

Now you might be able to see that the name comes from the fact that we are boosting (additive ensemble) and finding the next model via gradient descent.

Gradient Boosting = Gradient Descent + Boosting

Regression and Classification

Regression and Classification

So far we have defined a very generic algorithm.

The gradient boosting algorithm can be used for both regression and classification.

All we need to do is **change the loss function** L depending on the problem!

Regression

For regression problems we often use the least squares loss

$$L(y, F(\mathrm{x})) = \frac{1}{2}\sum_{i}^{n}(y_i - F(\mathrm{x}_i))^2$$

and the residuals are

$$\frac{\partial \sum_i^n L(y_i, F_{m-1}(\mathrm{x}_i))}{\partial F_{m-1}(\mathrm{x}_i)} = -y_i + F_{m-1}(\mathrm{x}_i)$$

Classification

For classification problems we often use the cross entropy (log loss). For binary classification this is

$$L(y, F(\mathrm{x})) = -\sum_{i}^{n} y_i \log(F(\mathrm{x}_i)) + (1 - y_i) \log(1 - F(\mathrm{x}_i))$$

where $F(\mathrm{x}_i) = P(y = 1|\mathrm{x}_i)$ i.e. our model gives the probability of class 1.

The residuals are

$$\frac{\partial \sum_i^n L(y_i, F_{m-1}(\mathrm{x}_i))}{\partial F_{m-1}(\mathrm{x}_i)} = \frac{y_i}{F_{m-1}(\mathrm{x}_i)} + \frac{1 - y_i}{1 - F_{m-1}(\mathrm{x}_i)}$$

Regression and Classification

Notice that in both cases we fit each h_m to a vector of residuals

$$\frac{\partial \sum_i^n L(y_i, F_{m-1}(\mathrm{x}_i))}{\partial F_{m-1}(\mathrm{x}_i)} = [e_1, e_2, \cdots, e_n]^T$$

So this means that the type of model we use for h_m just needs to be able to predict a continuous value since the residuals are continuous.

Therefore no matter if the problem is classification or regression, we can use the same base model h_m!

Base Model - Regression Tree

For the base models h_m we normally use a **Regression Tree**.

This is for two main reasons:

1. trees are simple and fast to train
2. shallow trees will prevent overfitting (small steps at each iteration)

Regularisation

To reduce overfitting even further some variations add a regularisation term when fitting h_m to penalise the tree complexity.

This means changing the objective from

$$h_m = \arg\min_h \sum_i^n (g_m(x_i) - h(x_i))^2$$

to

$$h_m = \arg\min_h \Omega(h) + \sum_i^n (g_m(x_i) - h(x_i))^2$$

where Ω is the penalty term, which for example might count either the number of leaf nodes or the depth of the tree.

Implementations

XGBoost

XGBoost (supposedly short for "eXtreme Gradient Boosting") is a popular implementation of Gradient Boosting.

The first innovation of XGBoost is the method for finding feature splits when training each tree. Instead of iterating over all possible feature splits, it uses an approximate quantile search algorithm.

The second innovation is computational. XGBoost exploits sparsity in features by ignoring the empty values and divides the data up into "blocks". Searching for the optimal feature splits can be done on a per-block basis and each block can be distributed.

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

LightGBM

LightGBM (Light Gradient Boosting Machine) promises improvements in training/prediction speed and lower memory requirements.

The idea of LightGBM is to use histogram representations of each feature. For each feature a histogram is built and each training point is assigned to the respective histogram bin.

This reduces the number of feature splits that need to be evaluated to the number of histogram bins. It should also reduce **overfitting**!

Memory requirements are lower because each feature value can only take on a small range of values. Normally we use 32 or 64 bit floats. For example if we bin a feature with 256 bins we can use an 8 bit integer instead.

https://github.com/microsoft/LightGBM

CatBoost

CatBoost is a new algorithm which promises improvements in prediction accuracy training/prediction speed.

We won't cover the details of CatBoost in QBUS6850 but you can read the details for yourself and try it out!

https://github.com/catboost/catboost

Discussion

Bias vs Variance

Since boosting adds models sequentially it sequentially forms a better approximation of the data generating process.

However since the models are not independent there is no guaranteed variance reduction like with bagging

Bias vs Variance

BUT since we are trying to correct the mistakes of the past it is unlikely that the weak learners are highly correlated.

> *In general, a greedy search strategy will not successively add highly correlated terms, since that would not produce very much local reduction in the criterion being minimized.*

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)."

Bias vs Variance

In other words boosting avoids specific forms of correlation such as adding a tree that is identical to the tree we already learnt. Which is avoided in a random forest by bagging and feature randomisation.

In summary we can roughly say that boosting reduces bias at the expensive of slight or limited increases in variance.

Advantages and Disadvantages

Disadvantages:

- very easy to overfit, since we clamp down on error at every iteration
- can't parallelise at the model level due to sequential training of models
- potentially large number of hyper-parameters
- harder to interpret than simple models like a single tree

Advantages:

- empirically one of the best performing models
- can parallelise and distribute the training inside each model

Interpretability

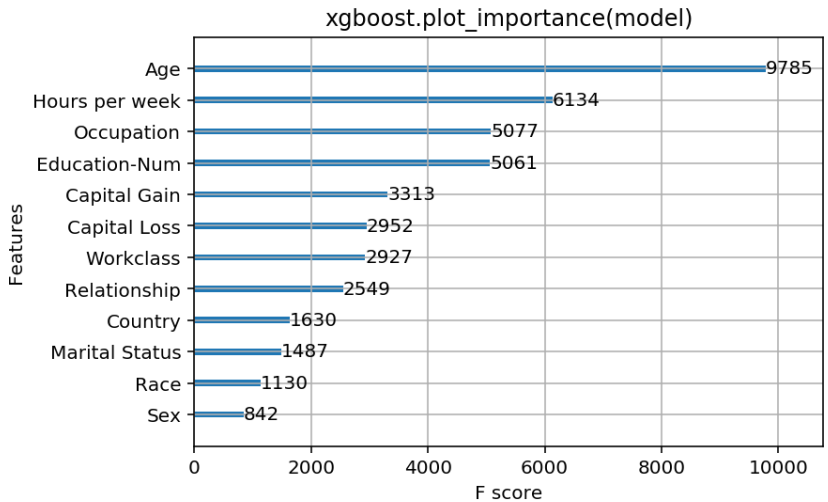
Although the general interpretability of ensemble methods is lower than single model methods there are ways to know which features are “important”.

The simplest method is to count the number of times each feature is used as a decision node. Or use a version weighted by the information gain of the feature.

Interpretability

Although the general interpretability of ensemble methods is lower than single model methods there are ways to know which features are "important".

The simplest method is to count the number of times each feature is used as a decision node. Or use a version weighted by the information gain of the feature.



Interpretability

However these methods are heuristic and you will get very different results depending on which method you use.

Instead it is recommended to use **SHAP feature importances**.

This is not an assessable part of QBUS6850, but please do some reading

- https://towardsdatascience.com/interpretable-machine-learning-with-xgboost-9ec80d148d27
- https://github.com/slundberg/shap
- https://christophm.github.io/interpretable-ml-book/shap.html#shap-feature-importance