XGBoost (Boosting)

Machine Learning II

December 23, 2020

1 Description

Boosting is another ensemble technique in with we use several estimators but not at the same time. We start by training a simple model, and upon the errors of this first estimator, we train another more complex model that puts more weight into those observations with higher errors. Thus we train the estimators in a sequence similar to that of a neural network.

We can apply boosting with Decision Trees by using several different available algorithms like `GradientBoosting` or `AdaBoost` in scikit-learn, or `XGBoost`.

2 Key concepts

- Ensemble: using several models and average their predictions.
- Boosting: Ensemble of models trained sequentially giving more weight to the observations with higher residuals
- Cost Function: function of the features that we want to minimize to achieve a good performance
- Regularization: method to avoid overfitting by adding a penalty to the cost function on the features

3 How it works?

Boosting works by starting with a "dumb" initial estimator, and sequentially adding more and more estimators that train on the previous estimator's errors.

In order to define the Boosting algorith we need to define first 3 parameters:

- Number of trees, M
- Shrinkage parameter, λ: rate at which the boosting learns. Very small shrinkage will require a very high number of trees, since each of them will learn very little

- Number of splits in each tree, d: if d=1 it's called a "stump". This value also refers to the number of terminal nodes of each tree (d + 1).

For a dataset consisting of N observations and P features, we can use boosting to train with the following algorithm:

```
# boosting
1. Initialize the model
    1.1. Set initial model to zero: f = 0
    1.2. Set weights to their y value: w_i = y_i
2. for each tree (m) in M:
    2.1. fit a tree f_m with d splits
    2.2. update f by adding shrinkage to the new tree:
        f = f + lambda * f_m
    2.3. update the residuals:
        w_i = w_i - lambda * f_m(x_i)
3. Output the boosted model:
    f(x) = lambda * f_1(x) + ... + lambda * f_m(x)
```

4 What could go wrong?

Unlike Random Forest, by adding too many trees to our boosted model we can fall into overfitting because we have added trees that have learned too much.

We can control this by tuning the shrinking parameter λ and the numbers of trees M. we should always keep in mind that the more trees (M) we include in our model, the less they should be learning in each round (λ).

Also, XGBoost comes with a handy feature that allows us to use as many trees as we want, and by using a validation set we can check when the validation error has reached a minimum and stop our boosting at that many trees. This technique is also used when training neural networks to avoid overfitting.

5 When to use it?

- One of the best options for big models in which we need good performance at the cost of computation.
- Will overfit if too many trees are added, needs to balance this with the shrinkage factor
- Explainability of these models is quite low, since its a bit of a black box in which we tune different knobs. Feature Importance improves this, but not comparable to Decision Tree.

6 Summary

Type	Regression, Classification
Tolerates many features?	Strong
Parametrization	Simple, intuitive
Memory needed	Very Large
Data requirements	Large
Overfitting	Medium
Difficulty	Average
Training time	Costly
Prediction time	Weak
Interpretability	Average