DSO530 Statistical Learning Methods

Lecture 7b : Bagging, Random Forests and Boosting

Dr. Xin Tong
Department of Data Sciences and Operations
Marshall School of Business
University of Southern California

Review: Advantages and disadvantages of trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Why? No mathematical formula needed in the communication
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous lectures.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Unfortunately, trees generally do not have the great predictive accuracy.
- This disadvantage motivates the ensemble methods such as *bagging*, *random forests* and *boosting*.

Bagging

- The decision trees typically suffer from high variance.
- *Bootstrap aggregation*, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- In this approach we generate B different *bootstrapped* training data sets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$ (not pruned), and average all the predictions:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)\,.$$

- Averaging these B trees reduces the variance. (Why?)
- B is not a critical parameter with bagging; a very large value of B will not lead to overfitting.
- In practice, use B sufficiently large so that the error has settled down.
- For a given test observation in `classification`, we can record the class predicted by each of the B trees, and take a vote: the overall prediction is the most commonly occurring class among the B predictions.

Out-of-bag Error Estimation

- For a bagged model, we can estimae the test error without doing CV or a validation set approach
- Each bagged tree makes use of a part of the original observations
- The remaining observations not used to fit a given bagged tree are referred to as the *out-of-bag* (OOB) observations
- We can predict the response for the ith observation using each of the trees in which that observation was OOB
- This will yield a little more than $B/3$ predictions for the ith observation (what is $(1-1/n)^n$ as n goes to infinity?)
- To obtain a single prediction for the ith observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal). This leads to a single OOB prediction for the ith observation
- An OOB prediction can be obtained in this way for each of the n observations, from which the overall OOB MSE (for a regression problem) or classification error (for a classification problem) can be computed

Variable Importance Measures

- Although the collection of bagged trees is much more difficult to interpret than a single tree, one can obtain an overall summary of the importance of each predictor using
  - the `RSS` (for bagging regression trees)
  - the `Gini index` (for bagging classification trees)
- In the case of bagging regression trees, we can record the total amount that the `RSS` is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor
- For bagging classification trees, we can add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.

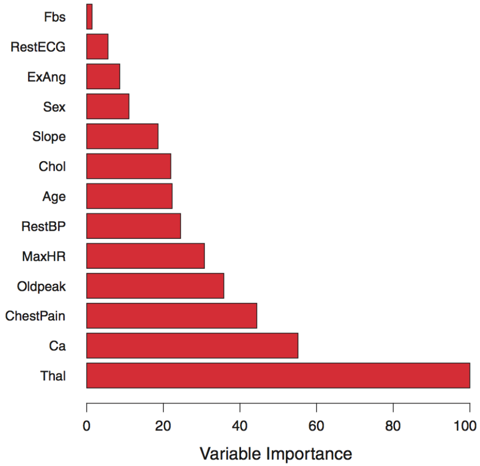
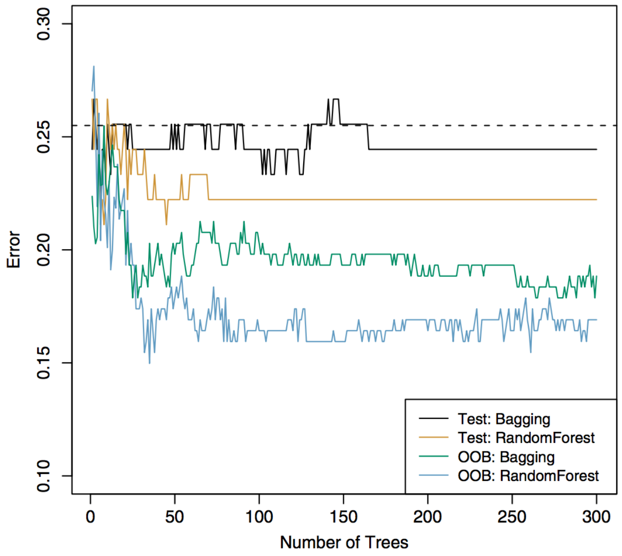


FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

Figure 1

Random Forest(s)

- Bagging constructs trees that are too "similar" (why?), so it probably does not reduce the variance as much as we wish to.
- Random forests provide an improvement over bagged trees by a small tweak that *decorrelates* the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, *a random sample of m predictors* is chosen as split candidates from the full set of p predictors. The split is allowed to use *only one* of those m predictors.
- So bagging is a special case of random forest when $m = p$
- If one does not want to spend extra efforts on m, one might use $m = \sqrt{p}$ as a canonical choice for classification and $m = p/3$ as a canonical choice for regression.
- As with bagging, random forests will not overfit if we increase B, so in practice people use B sufficiently large for the error rate to have settled down.
- Random forest is a really good `off-the-shelf` algorithm



Error
0.10
0.15
0.20
0.25
0.30
0
50
100
150
200
250
300
Number of Trees
Test: Bagging
Test: RandomForest
OOB: Bagging
OOB: RandomForest

Figure 2

Python implementation

- `RandomForestClassifier` and `RandomForestRegressor` in sklearn implement random forests in Python for classification and regression problems, respectively
- Our tutorial covers `RandomForestClassifier`
- Parameters: `n_estimators` (default 100) is the number of trees in the forest; `max_features` (default `sqrt(n_features)`) is the number of features to consider when looking for the best split.
- You can learn pick up `RandomForestRegressor` from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
- `RandomForestRegressor` in sklearn has a default setting of `max_features=n_features`. This is at odds with the recommendation by `ISLR`

A definition and some questions

- If you need to communicate a one sentence ad-hoc definition of random forests:
  – *Random forests are bagged decision tree models that split on a random subset of features on each split*.
- Q: In random forest algorithms, we restrict our attention to randomly selected m out of p features in each split. Now we change this procedure to restriction to the first m features (i.e., $X_1, \cdots, X_m$) in every split. Do you expect the new procedure to work well? And why?
- Q: If a decision tree partitions the feature space into regions $R_1 \cdots, R_J$, can any of these regions be a ball?
- Q: Is random forest always a better algorithm compared to decision trees?
- Q: What are the sources of randomness that a random forest model has? Hint: 3.
- Q: (open question) Can we extend the random forest idea to other base algorithms rather than trees?

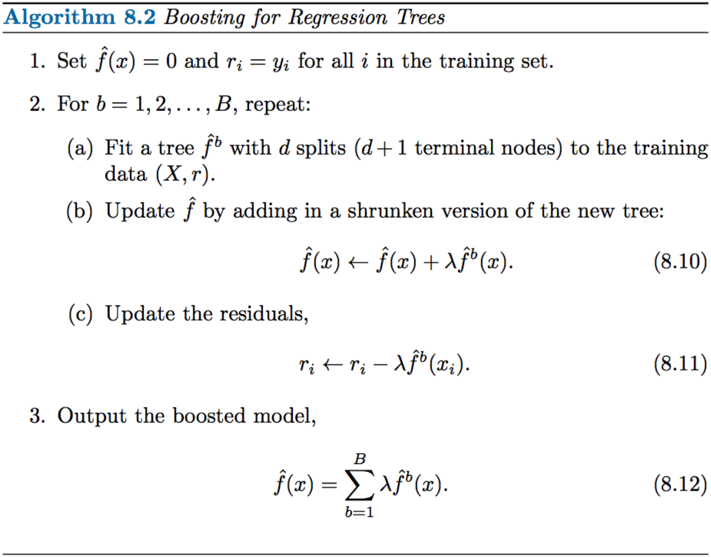
A definition and some questions

- If you need to communicate a one sentence ad-hoc definition of random forests:
  – *Random forests are bagged decision tree models that split on a random subset of features on each split*.
- Q: In random forest algorithms, we restrict our attention to randomly selected m out of p features in each split. Now we change this procedure to restriction to the first m features (i.e., $X_1, \cdots, X_m$) in every split. Do you expect the new procedure to work well? And why?
- Q: If a decision tree partitions the feature space into regions $R_1 \cdots, R_J$, can any of these regions be a ball?
- Q: Is random forest always a better algorithm compared to decision trees?
- Q: What are the sources of randomness that a random forest model has? Hint: 3.
- Q: (open question) Can we extend the random forest idea to other base algorithms rather than trees?

Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification.
- Boosting is an ensemble technique where new models are added to correct the errors made by existing models.
- A differentiating characteristic `Random forest: parallel vs. boosting: sequential`

An example from the ISLR book



Algorithm 8.2 *Boosting for Regression Trees*

```
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
    (a) Fit a tree $\hat{f}^b$ with $d$ splits ($d + 1$ terminal nodes) to the training data $(X, r)$.
    (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
            $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$        (8.10)
    (c) Update the residuals,
            $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$        (8.11)
3. Output the boosted model,
            $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$        (8.12)
```

Figure 3: Boosting Tree

In the boosting tree algorithm

- We use cross-validation to select B.
- The shrinkage parameter λ, a small positive number.This controls the rate at which boosting learns. Typical values are 0.01 or 0.001.
- The number d of splits in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a *stump*, consisting of a single split. More generally d is the *interaction depth*.
- Boosting is a *slow learner*.

In practice

- XGBoost is the to-go implementation of boosting algorithms for its execution speed and model performance.
- It is more complicated than what we described on the previous slide. For example, subsampling and thrinkage ideas are adopted.
- `xgboost` available in Python.