COMP 642 — Machine Learning Feb 8, 2022

Lecture 11

Lecturer: Anshumali Shrivastava
Scribe By: Chuyu Duan, Zeyu Yang, Sheng Cheng, Podshara Chanrungmaneekul

Disclaimer: *These lecture notes are intended to develop the thought process and intuition in machine learning. The materials are not thoroughly reviewed and can contain errors.*

Ensembles: Bagging and Boosting, XGBoost (Another Popular Classifier)

1 Bagging

1.1 Introduction

When we try to get a lower variance model, we can apply multiple models and then use their average or the most voted result as the final result. And that's called *bagging*, standing for *bootstrap aggregating*.

Suppose we have M models and our dataset is X, then the prediction can be made as follows:

$$y(X) = \frac{1}{M}\sum_{m=1}^{M} y_m(X)$$

And using $h(X)$ to represent the true result, then each model can be expresses with their error as follows:

$$y_m(X) = h(X) + \epsilon_m(X)$$

So, the average sum-of-square error is:

$$E_m[(y_m(X) - h(X))^2] = E_m[\epsilon_m(X)^2]$$

Then we can get the average error made by all of those models:

$$E_{AVG} = \frac{1}{M}\sum_{m=1}^{M} E_m[\epsilon_m(X)^2]$$

On the other hand, the prediction made by our final model is $\frac{1}{M}\sum_{m=1}^{M} y_m(x)$, so the expected error of final model is:

$$
\begin{aligned}
E_{final} &= E[(\frac{1}{M}\sum_{m=1}^{M} y_m(X) - h(X))^2] && (1)\\
&= E[(\sum_{m=1}^{M} \frac{y_m(X) - h(X)}{M})^2] && (2)\\
&= E[(\sum_{m=1}^{M} \frac{\epsilon_m(X)}{M})^2] && (3)\\
&= \frac{1}{M^2}E[(\sum_{m=1}^{M} \epsilon_m(X))^2] && (4)\\
&= \frac{1}{M^2}(\sum_{m=1}^{M} E[(\epsilon_m(X))^2]) + \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1,j\neq i}^{M} E[\epsilon_i(X)\epsilon_j(X)] && (5)\\
& && (6)
\end{aligned}
$$

Comparing with Equ 1.1, we can get
$E_{final} = \frac{1}{M}E_{AVG} + CorRelation$
If we assume that the error of each model is independent, then this shows that the average error of a model can be reduced by a factor of M through averaging M models. Even the $CorRealtion$ is not zero, we can still get an expected error no more than the averaged error of the initial models.

1.2 Sample with replacement

To enhance the performance, we want our M different models have diverse predictions, which implies they should get different test data.

One typical way is to generate new data set by sampling with replacement, as Figure 1 shows.

However, one disadvantage is that each model will now see part of the data. To be more specific, only about 0.63 can be seen by each model.

A single data point will not be selected from a set of size N in any of N draws is $(1-1/N)^N$. In the limit of large N, this becomes 0.37, which means only 0.63 of the data points will be selected[1]. Yet, one advantage is that those data out of bag can be used as test data, which is a useful alternative to cross validation.

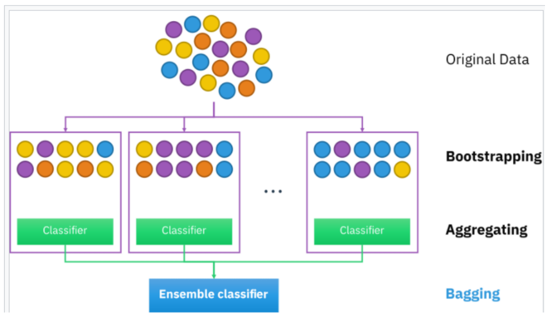

Figure. 1. Sample with replacement[2]

1.3 Advantages

The main advantage of bootstrap is that it prevents the ensemble from relying too much on any individual training example, which enhances robustness and generalization. For example, comparing Figure 2 and Figure 3, we see that omitting a single example from the training set can have a large impact on the decision tree that we learn (even though the tree growing algorithm is otherwise deterministic)[1].

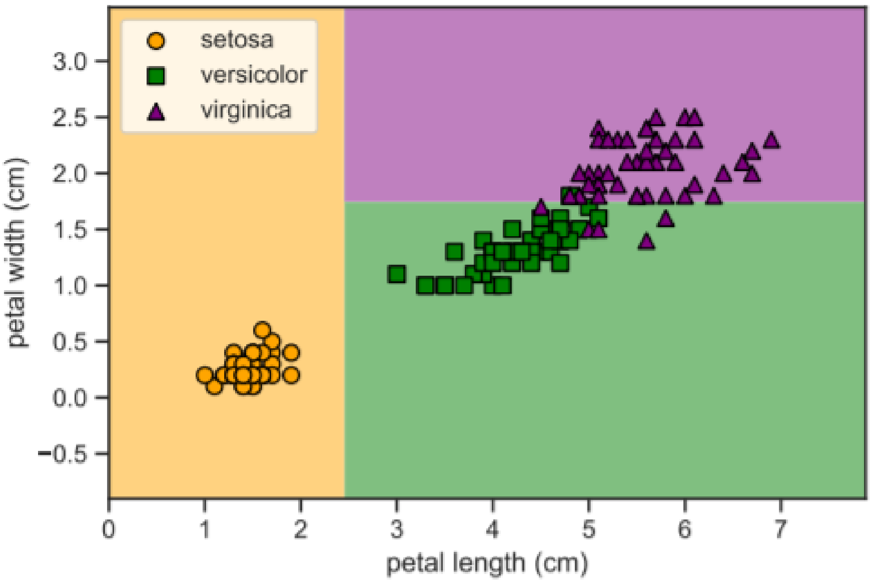

Figure. 2. Decision surface induced a decision tree of depth 2 fit to the iris data, using just the petal length and petal width features[1].

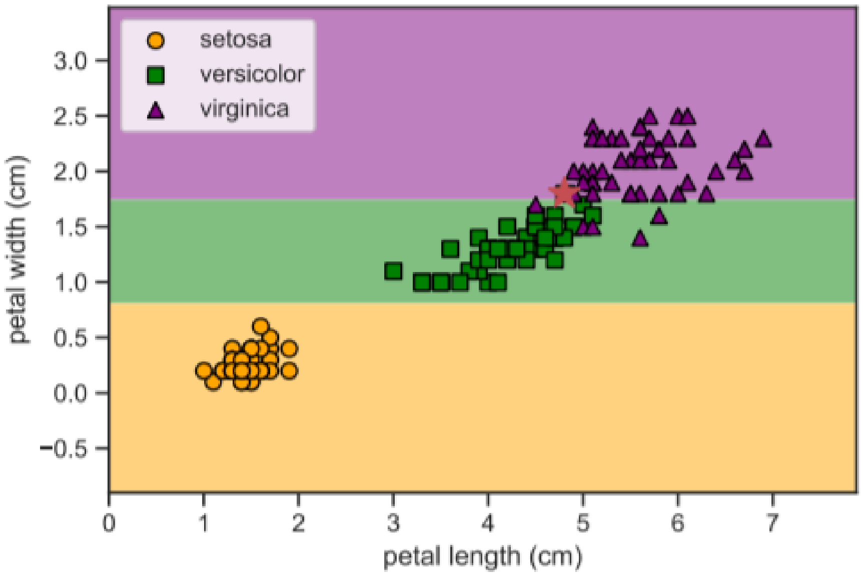

Figure. 3. Fit to data where we omit a single data point (shown by red star)[1].

By averaging the predictions from both of these models, we get the more reasonable prediction model in Figure 4.

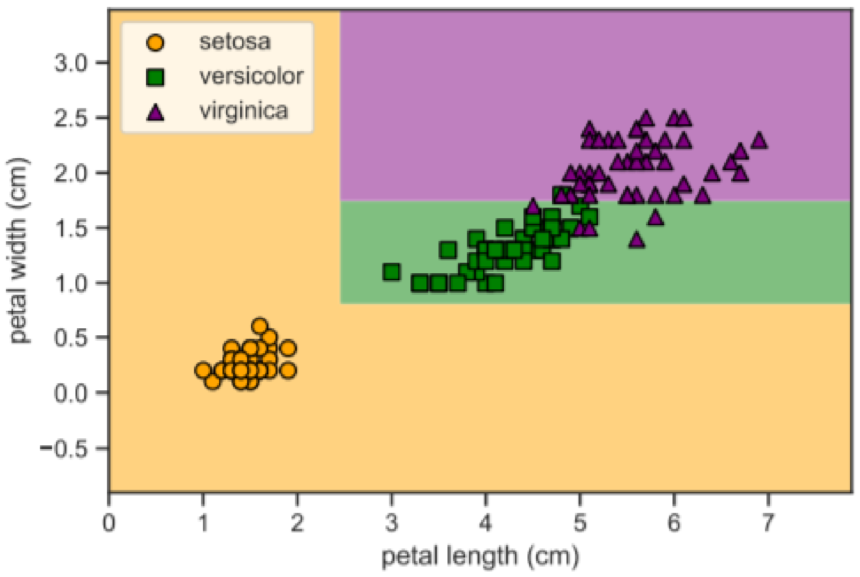

Figure. 4. Ensemble of the two models in figures 1 and 2[1].
This advantage generally increases with the size of the ensemble, as shown in Figure 5. (Of course, larger ensembles take more memory and more time.)

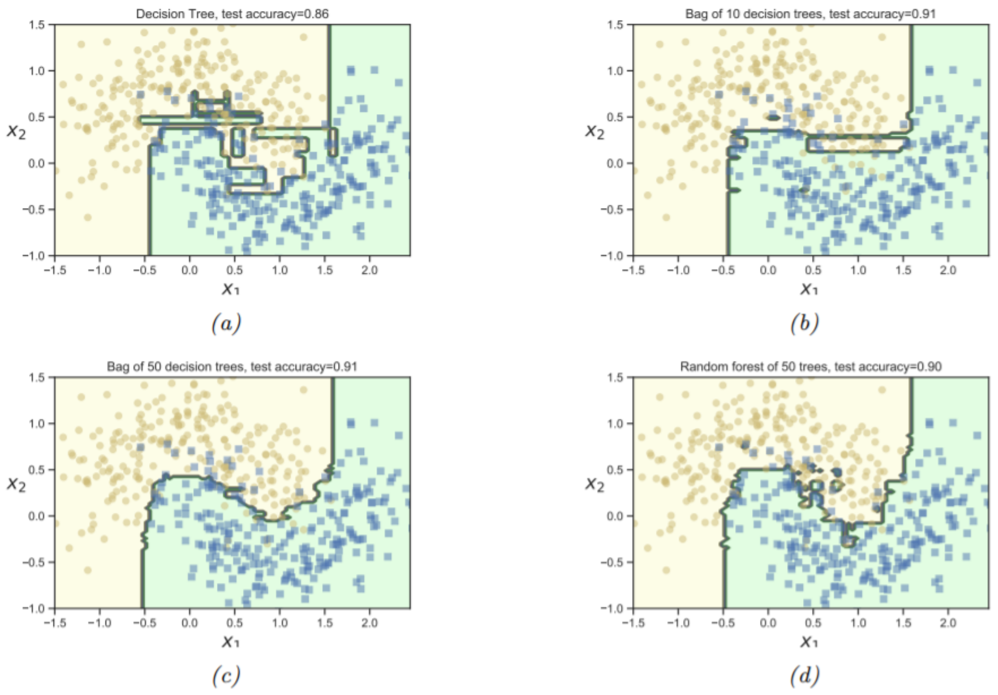

Figure. 5. (a) A single decision tree. (b-c) Bagging ensemble of 10 and 50 trees. (d) Random forest of 50 trees[1].

This is the case for decision trees, but not for other models, such as nearest neighbor classifiers. For neural networks, the story is more mixed. They can be unstable wrt their training set. On the other hand, deep networks will underperform if they only see 63 percent of the data, so bagged DNNs do not usually work well[1].

1.4 Random Forest

Bagging relies on the assumption that re-running the same learning algorithm on different subsets of the data will result in sufficiently diverse base models. The technique known as random forests tries to decorrelate the base learners even further by learning trees based on a randomly chosen subset of input variables (at each node of the tree), as well as a randomly chosen subset of data cases. It does this by modifying Equation 1 so the the feature split dimension j is optimized over a random subset of the features, Si $\subset \{1, ..., D\}$[1].

$$(j_i, t_i) = \arg \min_{j \in \{1,\ldots,D\}} \min_{t \in \mathcal{T}_j} \frac{|\mathcal{D}_i^L(j,t)|}{|\mathcal{D}_i|} c(\mathcal{D}_i^L(j,t)) + \frac{|\mathcal{D}_i^R(j,t)|}{|\mathcal{D}_i|} c(\mathcal{D}_i^R(j,t))$$

Equation. 1[1].

For example, consider the email spam dataset in The book *Elements of Statistical Learning*, 2nd edition. Springer, 2009., on page 301. This dataset contains 4601 email messages, each of which is classified as spam (1) or non-spam (0). The data was open sourced by George Forman from Hewlett-Packard (HP) Labs.

There are 57 quantitative (real-valued) features, as follows:

• 48 features corresponding to the percentage of words in the email that match a given word, such as "remove" or "labs".

• 6 features corresponding to the percentage of characters in the email that match a given character, namely ; . [! $ #.

• 3 features corresponding to the average length, max length, and sum of lengths of uninterrupted sequences of capital letters. (These features are called CAPAVE, CAPMAX and CAPTOT.)

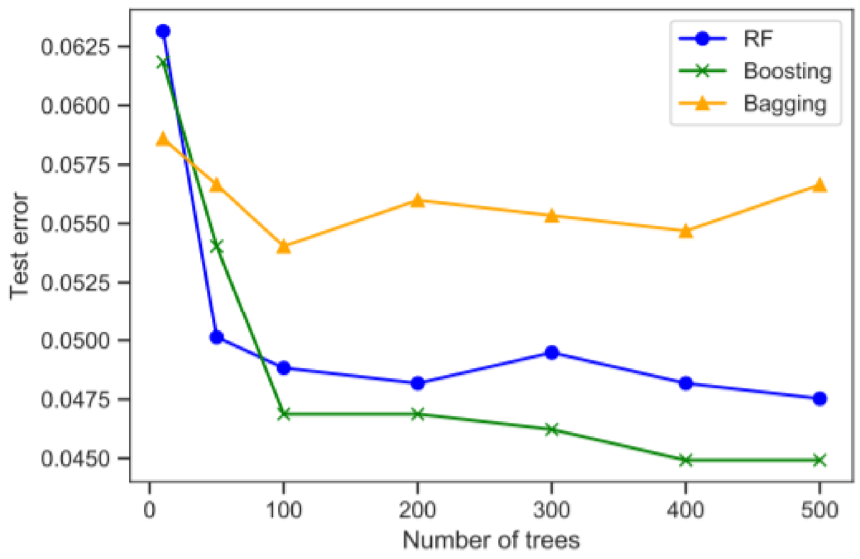

Figure. 6. Preditive accuracy vs size of tree ensemble for bagging, random forests and gradient boosting with log loss[1].

Figure 6 shows that random forests work much better than bagged decision trees, because many input features are irrelevant. (We also see that a method called "boosting", discussed in the next section, works even better; however, this requires sequentially fitting trees, whereas random forests can be fit in parallel[1].)

2 Boosting

Boosting is an ensemble learning method that combines a set of weak learners into a strong learner to minimize training errors. In many cases, boosting can work better than bagging and random forest fits trees that depend on each other to reduce the bias of the strong learner.

2.1 AdaBoost (Adaptive Boosting)

The core principle of AdaBoost is to fit a sequence of weak learners on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction.

Initially, the weights of each boosting are set to be equal. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly.

As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence

Algorithm 1 Adaboost.M1, for binary classification with exponential loss

```
$w_i = 1/N$
for $m = 1 : M$ do
    Fit a classifier $F_m(x)$ to the training set using weight $w$
    Compute $err_m$ on the training set
    Compute $\alpha_m = \log[(1 - err_m)/err_m]$
Return $f(x) = sgn[\sum_{m=1}^{M} \alpha_m F_m(x)]$
```

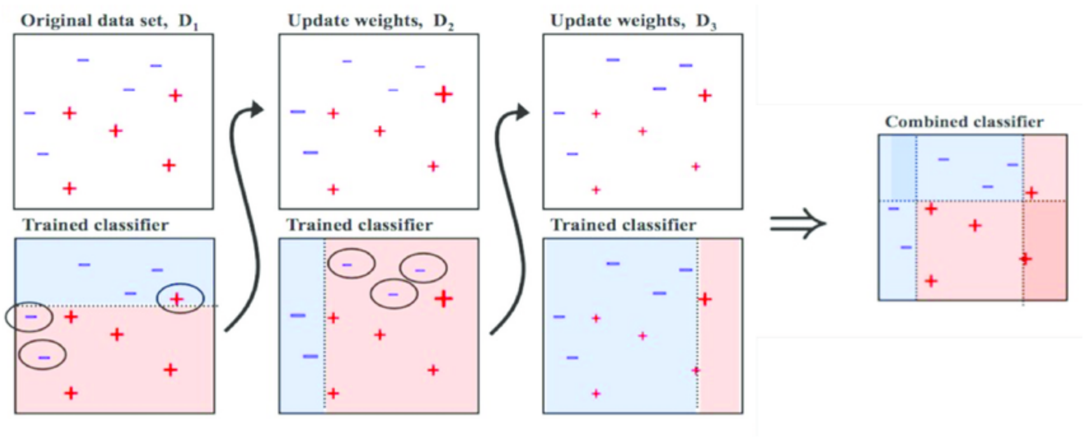

Figure. 7. Method of Ada boosting that aggregate multiple weak learners and combine them to create a more accurate classifier

2.2 Gradient boosting

In gradient boosting, it trains many model sequentially. Each new model gradually minimizes the loss function of the whole system using Gradient Descent method. The learning procedure consecutively fit new models to provide a more accurate estimate of the response variable.

The principle idea behind this algorithm is to construct new base learners which can be maximally correlated with negative gradient of the loss function, associated with the whole ensemble. You can refer article “Learn Gradient Boosting Algorithm” to understand this concept using an example.

Algorithm 2 Gradient boosting

```
Initialize $f_0(x) = argmin_F \sum_{i=1}^{N} L(y_i, F(x_i))$
for $m = 1 : M$ do
    Compute the gradient residual using $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$
    Use the weak leaner to compute $F_m = argmin_F \sum_{i=1}^{N} (r_{im} - F(x_i))^2$
    Update $f_m(x) = f_{m-1}(x) + F_m(x)$
Return $f(x) = f_M(x)$
```

2.3 XGBoost

XGBoost, which stands for "extreme gradient boosting", is a very efficient and widely used implementation of gradient boosted trees. In addition to the previous method, it adds a regularizer on the tree complexity, it uses a second order approximation of the loss instead of just a linear approximation, it samples features at internal nodes (as in random forests), and it uses various computer science methods (such as handling out-of-core computation for large datasets) to ensure scalability.

More specifically, XGBoost optimizes the following regularized objective function.

$$\mathcal{L}(f)=\sum_{i=1}^{N} \ell\left(y_i, f\left(\boldsymbol{x}_i\right)\right)+\Omega(f) \tag{7}$$

where

$$\Omega(f)=\gamma J+\frac{1}{2} \lambda \sum_{j=1}^{J} w_j^2 \tag{8}$$

is the regularizer.

The loss is given by

$$\mathcal{L}_m\left(F_m\right)=\sum_{i=1}^{N} \ell\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right)+F_m\left(\boldsymbol{x}_i\right)\right)+\Omega\left(F_m\right)+\text{ const} \tag{9}$$

XGBoost optimize the objective function in a greedy recursive manner. That is, iteratively split data according to gain for such a split.

$$\text{gain } =\frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{\left(G_L+G_R\right)^2}{\left(H_L+H_R\right)+\lambda}\right]-\gamma \tag{10}$$

where $G_L=\sum_{i \in I_L} g_{im}, G_R=\sum_{i \in I_R} g_{im}, H_L=\sum_{i \in I_L} h_{im}$, and $H_R=\sum_{i \in I_R} h_{im}$. Thus we can split the data as long as gain is large. The full algorithm for XGBoost is as follow.

Algorithm 3 XGBoost

```
Initialize $f_0(x)=argmin_F \sum_{i=1}^{N} L(y_i, F(x_i))$
for $m=1:M$ do
    Compute the gradient using $g_{im}=\left[\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$
    Compute the Hessian using $h_{im}=\left[\frac{\partial 2L(y_i,f(x_i))}{\partial f(x_i)^2}\right]_{f(x_i)=f_{m-1}(x_i)}$
    Use the weak leaner to computer$F_m=argmin_F \sum_{i=1}^{N} \frac{1}{2} h_{im}\left[-\frac{g_{im}}{h_{im}}-\phi(x_i)\right]^2$
    Update the model $f_m(x)=f_{m-1}(x)+sF_m(x)$
Return $f(x)=f_M(x)$
```

References

[1] Murphy, K.P., 2022. *Probabilistic machine learning: an introduction*. MIT press.
https://en.wikipedia.org/wiki/Bootstrap_aggregating