Week7: Bagging

Chenyue Cai, Oscar Xu, Yintang Yang
University of California, Berkeley
Berkeley, CA

December 7, 2020

1 Introduction

1.1 Ensemble Methods

So far, you have been dealing with one single model and you have learned to decide on the model to use for a particular problem. You may ponder once in a while, is that the best we could do? Are there any other ways that could improve the accuracy of our machine learning model? In this case, you will need some more advanced machine learning techniques that are collectively referred to as ensembles.

If you searched on google, the definition of the ensemble is as following: 1. A group of musicians, actors, or dancers who perform together. 2. A group of items viewed as a whole rather than individuality. How is that related to machine learning models? Literally speaking, if we treat each machine learning model as a musician that plays a certain musical instrument, an ensemble would be a group of musicians that performs this particular machine learning task together and their collective work would be evaluated at last. In the machine learning field, there are quite a lot of ensemble methods here we are going to focus on one particular ensemble method: bagging. (If you are interested in other ensemble methods, please feel free to check them up. https://www.wikiwand.com/en/Ensemble_learning)

1.2 A Brief History on Bagging

Bagging is first introduced for variance reduction in 1994 by Leo Breiman, a statistician at UC Berkeley (Go bears). Here is the link to his original paper for bagging: Bagging Predictors, which we strongly encourage you to skim over. For your convenience, we provide the abstract of the paper here: Abstract: Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class. The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets. The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy.

After gaining the formal definition of bagging, you may still have quite a few questions. This note will explain in detail all the nitty-gritty of bagging. By the end of the note, you should be able to answer the following questions: Why bagging? How to do bagging? What is bootstrapping? What is the variance reduction? When does bagging work? Hold on to them throughout your reading of the note.

2 Bias and Variance

When dealing with a single machine learning model, how do we evaluate its performance? So far, machine learning scientists have developed interpretable error metrics: bias and variance decomposition. On a higher level, bias measures how well the average model approximates the true underlying ground truth $f(x)$. While variance means, how much a model will change given different training sets.

2.1 Terminology

- Random variable: a random variable is a numerical function that maps domain onto a real number line
- Expectation: the average of the possible values of a random variable weighted by their probabilities.
- Variance: variance is the measurement of deviation from the expected value of a random variable.

2.2 Bias and Variance Decomposition

Suppose we have ground truth function f(x) and the observation data of the ground truth function Y. Since real-life data is noisy we consider Y = f(x) + Z where Z is a number of the standard Gaussian distribution. We choose Gaussian distribution since it is the most common distribution of data and it has the property that we want E(Z) = 0, that the average of Zs are 0 since noises are just random stuff that bounces around the true function. We also have an h(x), the model that we learned from the data. We are interested in knowing the error of h(x), which we will formulate as the mean square error $\sum_x (h(x) - Y)^2$. If we write it in a probabilistic manner using the terms we just define, we will get $E[(h(x; D) - Y)^2]$. By linearity of expectation, we will derive

$$\begin{aligned} E[(h(x; D) - Y)^2] &= E[(h(x, D)^2 - 2 * Y * h(x, D) + Y^2] \\ &= E[h(x; D)^2] + E[Y^2] - 2E[h(x; D) \cdot Y] \end{aligned}$$

Here we used that the fact that

$$E(X^2) - E(X)^2 = Var(X)$$

Then we could further derive that

$$
\begin{aligned}
&Var(h(x;D)) + E[h(x;D)]^2 + Var(Y) + E[Y]^2 - 2E[h(x;D)] \cdot E[Y] \\
&= E[h(x;D)]^2 - 2E[h(x;D)] \cdot E[Y] + E[Y]^2 + Var(h(x;D)) + Var(Y) \\
&= E[h(x;D)] - E[Y]^2 + Var(h(x;D)) + Var(Z) \\
&= \underbrace{E[h(x;D)] - f(x)^2|z}_{\text{(squared bias of method)}} + \underbrace{Var(h(x;D))|z}_{\text{(variance of method)}} + \underbrace{Var(Z)|z}_{\text{(irreducible error)}}
\end{aligned}
$$

2.3 Recap on EE16A

The math could be hard to absorb at first, let's first light up the mood by looking through some easy examples that are based on what you have learned from EE16A.

In EE16A, we have learned about the basics of machine learning: identifying the problem of classification, estimation, prediction, and clustering, mastering some linear algebra techniques to solve machine learning problems, eg. least square, optimizing a loss function. (For a quick 16A ML recap please go to 16A ML)

Let's start with a set of problems that we are all familiar with in EE16A, "the line of best fit" problems. We learned the linear algebra derivation of least square problems and that is always a good way to solve simple regression problems. However, given the nature of linearity, least-square failed to perform well on high dimensional data. Remember the planet orbiting problem in EE16A when we uplift the feature to cater to Kepler's Law?

Here we continue with polynomial examples. We formulate the example just as our mathematical formation, a ground truth function with some gaussian noises. We use a set of different degrees of the polynomial to approximate the function. As you can see, when increasing the polynomial degree, we are getting increasing variance with lowered variance. This is because an expressiveness of a higher degree of polynomial manages to capture all the nitty-gritty of data, however, the variance also increases since such expressiveness leads to varied approximations.

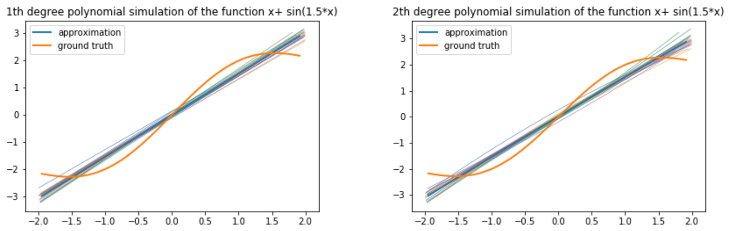
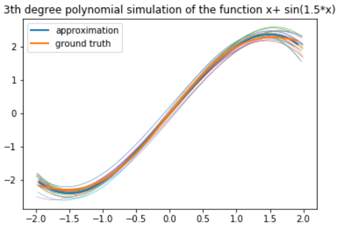
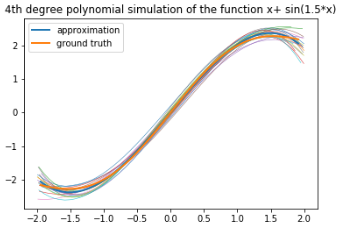
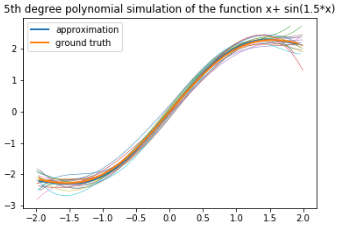
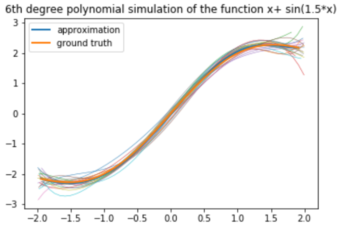


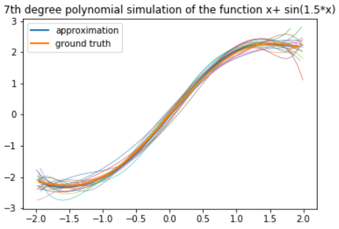
Figure 1:

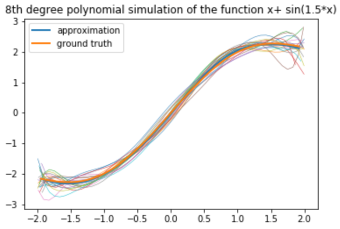

3th degree polynomial simulation of the function x+ sin(1.5*x)


4th degree polynomial simulation of the function x+ sin(1.5*x)


5th degree polynomial simulation of the function x+ sin(1.5*x)


6th degree polynomial simulation of the function x+ sin(1.5*x)


7th degree polynomial simulation of the function x+ sin(1.5*x)


8th degree polynomial simulation of the function x+ sin(1.5*x)

Indeed, different learners and model classes have different bias-variance trade offs. In the polynomial example, we can empirically determine that a high degree of polynomial regression is an example of high variance low bias, contrary to a low degree, an example of low variance high bias. Generally, large bias/small variance means few features, highly regularized, such as highly pruned decision trees, large-k kNN, etc; While small bias/high variance means many features, less regularization, small-k k-NN, etc.

3 Bagging

When we do bagging, we first bootstrap M different samples without replacement from the population. Since we are sampling without replacement, then these samples are independent of each other. Thus we name them as $Y_1, Y_2, ..., Y_m$ independent random variables each with mean μ and variance σ^2. As we average them, we are basically taking the expectation of the following terms which you will see is still μ

$$\frac{1}{M}\sum_{i=1}^{M} Y_i = \frac{1}{M} \times M\mu = \mu$$

Therefore after all these mumble jumbles we still get that good low bias. As for variance, let's do our calculation:

$$var(\frac{1}{M}\sum_{i=1}^{M} Y_i) = \frac{1}{M^2} \times var(\sum_{i=1}^{M} Y_i) = \frac{1}{M^2} \times \sigma^2 \times M = \frac{\sigma^2}{M}$$

As we can see the variance goes down linearly as M increases. This is the exact reason why we are able to get a better result in the end.

3.1 Bootstrap

3.1.1 Train-Test Split

Based on our observations to make predictions is a common topic we will study in Machine Learning. We always separate the data set into a training set and a test set. The training set is used to fit the model we picked, and the test set is to check the generalization ability, which is the accuracy of the model in predicting unknown data. Train-Test Split is the method to separate a data set into a training set and a test set.

You will try to do Train-Test Split in the Jupyter Notebook exercise. Note that Python has a Machine Learning library called `‘‘sklearn.model_selection’’`, we can directly import `train_test_split` function from this library. Commonly, we will select around `70% ~ 80%` of the entire data set for the training set, and in the built-in `train_test_split` function, you can input the values of `test_size` and `train_size` as you want. If you do not initialize them, the `test_size` will equal 0.25 and the `train_size` will equal 0.75 as default. Also, two subsets must be selected from the whole data set uniformly, in order to reduce the bias between subsets and the complete data set.

One of the most important things that you must keep in mind is that the test set can only be used once. Now, you may have a question: Using the test set only once is fine if we have already know which model we are using, but what if we need to select the best model from multiple models? In the next section, we will introduce another set, called the Validation set.

3.1.2 Bootstrap

Bootstrap is a robust method to estimate the population value with some samples. The idea for Bootstrap is pretty simple. If we randomly select some observations from the data set and use the sample to estimate some unknown value we want only once, the value cannot represent the real one. Instead, we can select a sample multiple times, and average the value we found each time. It will make our prediction closer to the real value. Bootstrap treats the data set as the whole population, and repeatedly selects sample data from the whole data set, randomly and with replacement. Each sample group is different from the other, and each sample group influences the overall descriptive indicators such as mean, median, and standard deviation. Perhaps one sample group has a bigger mean than other sample groups, and another sample group has a smaller mean than other sample groups. The effect will be canceled out when we are finding the average of the mean of each sample group. Therefore, bootstrap is robustness and we can use it to test data stability.

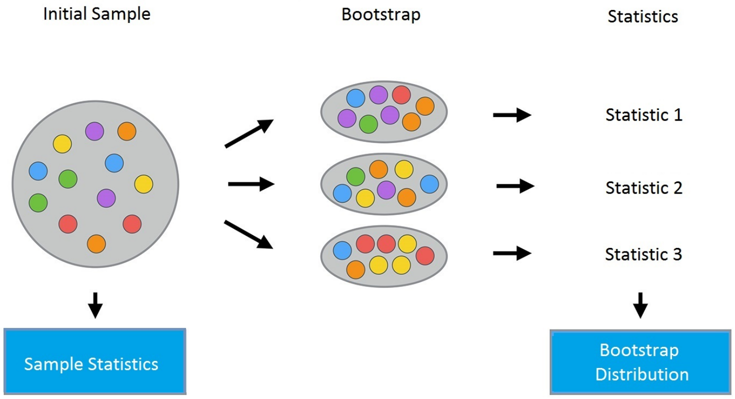


Figure 2: Bootstrap [1]

3.2 K-Fold Cross-Validation

Cross-validation is a technique used to evaluate the performance of the classifiers. As we discussed in the last section, when we have multiple classifiers, we need to pick the classifier that performance best before we use the test set to check the generalization ability, since the test set can only be used once. To do that, we need another set, called the Validation set. The validation set can be seen as an indicator to teat each classifier's performance. After we use the training set to train each classifier, we use the validation set to test each trained model and pick the one that has a minimum error rate. Then we can use a test set to test this model's generalization ability.

K-Fold cross-validation is one of the most common methods used in cross-validation. When we have an insufficient data set, we can use K-Fold Cross-Validation, cause it to maximize the use of the data set. We decide the number of parts - K we want to divide for the training set. After we use Train-Test split to split the whole data set into a training set and test set, we divide the training set into K parts again. Each part usually has the same size. As you can see in the following image, there are total K iterations. At iteration i, the validation set is the ith subset, and the remaining K-1 subsets are used as a training set. We then calculate the error rate for each iteration. Note that in this way, the data used to train the model is independent of the data used to test model performance. Finally, we calculate the mean of all error rates and use it as the estimation of the model's accuracy.

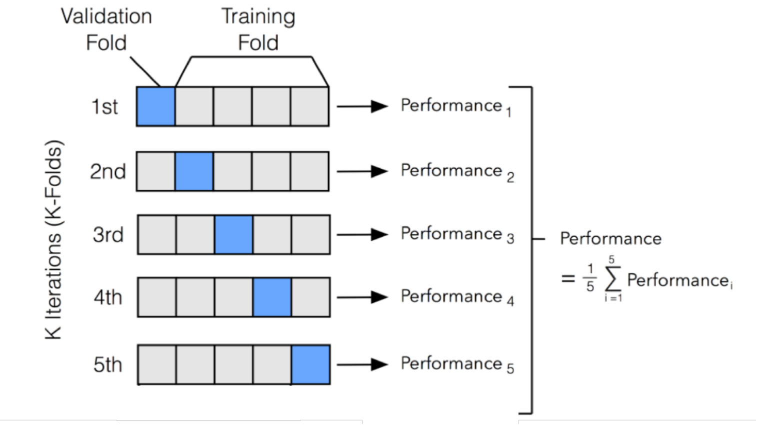


Figure 3: K-Fold Cross-Validation [2]

4 Random Forest

4.1 Decision Tree

Decision trees are a type of model used for both classification and regression. Trees answer sequential questions that send us down a certain route of the tree given the answer. The model behaves with "if this than that" conditions ultimately yielding a specific result.

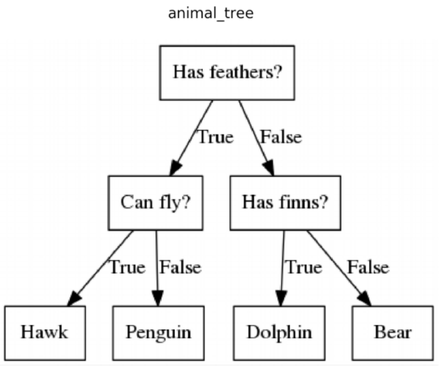


Figure 4: Linear regression [3]

4.2 Bagging and Random Forest

Decision trees tend to have overfitting issues since they can have grown infinitely to fit in all data points. Random forest is an ensemble of many decision trees. Random forests are built using a method called bagging in which each decision tree is used as parallel estimators. If used for a classification problem, the result is based on the majority vote of the results received from each decision tree. For regression, the prediction of a leaf node is the mean value of the target values in that leaf. Random forest regression takes the mean value of the results from decision trees. We will see the effect on the coding assignment.

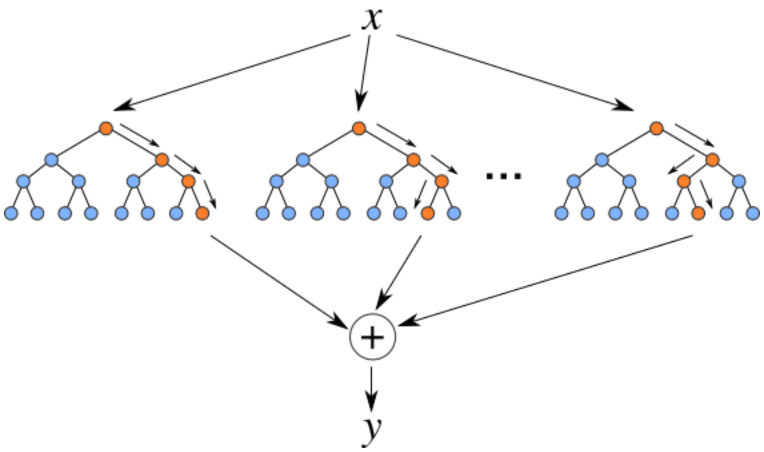


Figure 5: Random forests are multiple decision trees [4]

References

[1] Figure 2: https://www.kaggle.com/kashnitsky/topic-5-ensembles-part-1-bagging
[2] Figure 3: http://ethen8181.github.io/machine-learning/model_selection/model_selection.html
[3] https://towardsdatascience.com/decision-tree-and-random-forest-explained-8d20ddabc9dd
[4] https://levelup.gitconnected.com/random-forest-regression-209c0f354c84?gi=7fa747e0efa3