Introduction to Machine Learning (67577)

Ensemble Methods: Bagging and Boosting

Special Covid-19 edition, 4/2020

Contents

Suggested Reading:

- Bootstrap: ESL 7.11

- Bagging: ESL 8.7
- Random forests: ESL 15
- Boosting: UML 10
- Adaboost: UML 10.2, ESL 10.1

More advanced reading (beyond the material covered):

- Statistical learning view of boosting: ESL 10.2, 10.4, 10.5
- Gradient boosting: ESL 10.10
- Interpreting Boosted Trees and Random Forests: ESL 10.13

1 Introduction

After two lectures on theory of machine learning, we are back to new machine learning algorithms - and specifically, for supervised batch learning.

In the classification lecture we saw several learning algorithms for classification. We saw that they are very different from each other - each of them implements a different **principle** for choosing the learned rule $h_S \in \mathcal{H}$ based on the training sample S, and each uses a different algorithm to implement the chosen principle computationally - first as a formally stated algorithm, and then actual implementation in software.

The classifiers we saw (eg. SVM, trees) are not "cutting edge" - none of them would be considered the best choice for classification today. But with the methods of this lecture we finally get to create state-of-the-art classifiers. Some of the best-known classification methods, which often that win competitions, are based on the methods we'll see this in this lecture[1].

This lecture will be different from our classification lecture. We won't be seeing any new learning algorithms. Instead, we will see **"meta-algorithms"** - general methods that can be applied to any **existing** learning algorithm and improve its performance. The improvement in performance can be quite radical.

We will address classification problems for simplicity, but everything you see here generalizes (sometimes trivially) to regression problems as well.

We will learn about the three B's: **Bootstrap**, **Bagging** and **Boosting**. You should package these ideas together (call them B^3) and take them with you wherever you go.

- **Bootstrap** is one of the most useful, important and influential ideas in statistics since computers stared being used to analyze data. It is a truly magical idea that has many, many uses and applications.

[1]For example, a list someone compiled of competitions won using gradient boosting: `https://github.com/Microsoft/LightGBM/blob/master/examples/README.md#machine-learning-challenge-winning-solutions`

- One of the uses of Bootstrap is for improving prediction of a supervised learning algorithm. (We'll see another use in future lectures.) **Bagging** is a nickname for Bootstrap as it is used to improve prediction of a supervised learning algorithm.
- **Boosting** is another truly magical idea, which is completely different. And is one of the most useful, important and influential ideas in machine learning.

So, we're learning powerful and broadly applicable ideas today. You'll be able to use these ideas when you encounter difficult problems, even outside the context of the meta-algorithms we will see today.

1.1 Bias/Variance

The meta-algorithms we will see in this lecture succeed because they change the bias and the variance of the learning algorithms on top of which they are applied. So, before we actually start talking about them, let us recall the **bias-variance tradeoff** (also known as the bias-complexity tradeoff).

Several times in the course so far, we stated - informally - that the "larger" or "more complicated" our chosen hypothesis class, typically our learner will have lower **bias** and higher **variance.**

We said informally that **bias** is part of the generalization error that is incurred by the "best" hypothesis in $\mathcal{H}$. If we think of an unknown labeling function f chosen by nature, then bias measures how well the unknown labeling function f can be decried by the "closest" hypothesis in $\mathcal{H}$. Obviously, the larger $\mathcal{H}$, the more expressive power it has to describe more complicated functions f - hence a lower bias.

We said informally that **variance** is the part of the generalization error that is incurred by the fact that the training sample is random, hence our chosen rule h_S is also random. The larger $\mathcal{H}$ will be, the more freedom our learning algorithm has to "chase" random fluctuations in the training sample, which do not represent the underlying labeling we are trying to learn. (Variance can be further broken down into two parts - one part comes from randomness in the choice of training samples, and another part comes from the measurement noise or noise in the labels. Let's keep in simple and not go into that now.)

We mentioned the bias-variance **tradeoff** - the more complicated the model, the smaller the bias and the larger the variance. Informally, the generalization error is the sum (or somehow the combination) of these two. So when we can tune the model complexity (another name for the size / complexity of our hypothesis class) we'll look for the "sweet spot" of a model that not has just the right amount of complexity, not too much and not too little.

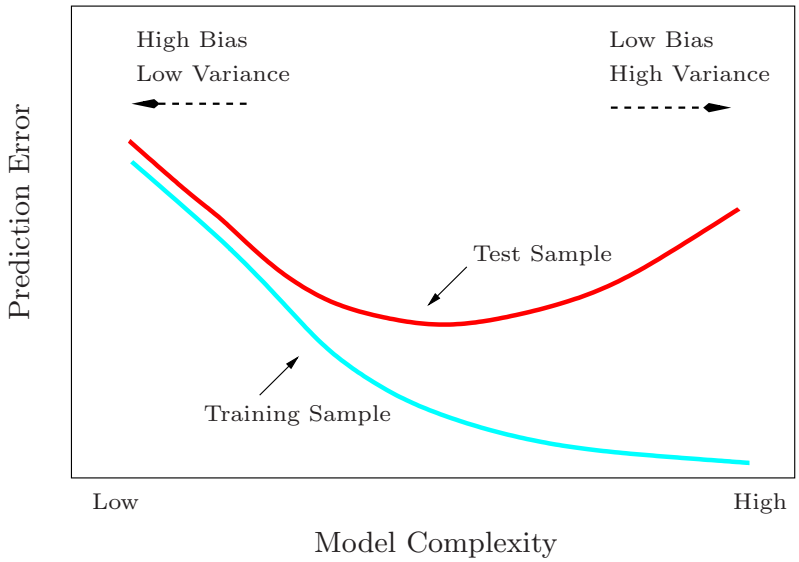

Figure 1: the bias-variance tradeoff

We will revisit the bias-variance more formally in one of the next lectures.

The magic in the methods we'll see in this lecture is that they allow us to escape the tradeoff in a certain sense. One meta-algorithm we'll see will reduce the bias of the learning algorithm it's applied to - without substantially increasing its variance. The other meta-algorithm we'll see will reduce variance without substantially increasing the bias. So, magic.

1.2 Ensemble / Committee methods

> "A collective wisdom of many is likely more accurate than any one." — Aristotle, in Politics, circa 300BC

It's been known for a long time that committees typically make better decisions than individuals. (The original Greek democracy basically comes down to this idea.)

Consider a committee of T members, which has to make a yes/no decision. In hindsight we'll know whether the decision has been right or wrong. Each member casts her vote. Each one has probability p of being correct and probability $1-p$ of being wrong. Let's assume for simplicity that all members are "equally wise", so that p is the same for all members. After all members vote, the committee's decision is simply the majority vote.

Exercise. Let $X_1, \ldots, X_T$ be i.i.d Bernoulli (p) random variables taking values in $\{\pm 1\}$. Show that the above committee's random decision is given by (when each member has equal probability p to be right) is simply $\overline{X} := sign(\sum_{t=1}^{T} X_t)$. Conclude that the probability that the above committee will make the right decision is $\mathbb{P}\left\{\overline{X} > 0\right\}$.

1.3 The uncorrelated case.

Accuracy of the committee's decision. It turns out that if each member is typically right ($p > 0.5$), then the probability that the committee is right is much higher than any individual

member, and growing with the number of committee members T: **Exercise.** Assume that all members vote **independently** of each other. What is the probability that the committee's decision is right (namely, what is $\mathbb{P}\left\{\overline{X} = +1\right\}$ as function of p and T? What is the limit of this probability as $T \to \infty$? Plot the probability that the committee's decision is right over T, for $p = 0.4, 0.5, 0.7, 0.9$. (Note: A committee of fools is a terrible decision maker: Observe that if $p < 0.5$, so that each member is usually wrong, then the majority vote will be worse than a single vote.)

Variance of the committee's decision. Now we wonder if the committee makes decisions **consistently**, namely, if it votes several times - each time the entire voting process is independent of all other times - how likely is the committee to make the same decision? So let's consider the **variability** in the committee's decision

Exercise. Given i.i.d real-valued random variables $X_1, \ldots, X_T$, each with variance σ^2, show that the variance of $\overline{X} := T^{-1} \sum_{t=1}^{T} X_t$ is $(\sigma^2)/T$.

Exercise. What is the variance of a single Bernoulli random variable $X \sim Ber(p)$? What is the variance of the committee's vote $\overline{X}$ as function of T and p? plot this variance over T for $p = 0.4, 0.5, 0.7, 0.9$.

1.4 The correlated case.

In practice, however, committee members rarely vote independently. So let's assume that each two members are correlated with equal correlation $0 \leqslant \rho \leqslant 1$. So that each member is right with (equal) probability p and each pair of members have (equal) correlation ρ.

Accuracy of the committee's decision.
Exercise. Assume every two committee members have equal correlation ρ as above, and that their individual decision-making process is the same. **Use a simulation** to plot the probability that the committee's decision is right (namely, what is $\mathbb{P}\left\{\overline{X} = +1\right\}$ as function of T, for $p = 0.7$ and $\rho = 0.2, 0.5, 0.9$. What do you think is the limit of this probability as $T \to \infty$?

Variance of the committee's decision.
Exercise. Given identically-distributed real-valued random variables $X_1, \ldots, X_T$, each with variance σ^2, and such that $corr(X_i, X_j) = \rho$ for all $1 \leqslant i \neq j \leqslant T$, show that the variance of $\overline{X} := T^{-1} \sum_{t=1}^{T} X_t$ is

$$\rho \cdot (\sigma^2) + (1 - \rho) \cdot \frac{\sigma^2}{T}$$

Exercise. **Use a simulation** to plot the variance of the committee's vote $\overline{X}$ over T T, for $p = 0.7$ and $\rho = 0.2, 0.5, 0.9$. What do you think is the limit of this variance as $T \to \infty$?

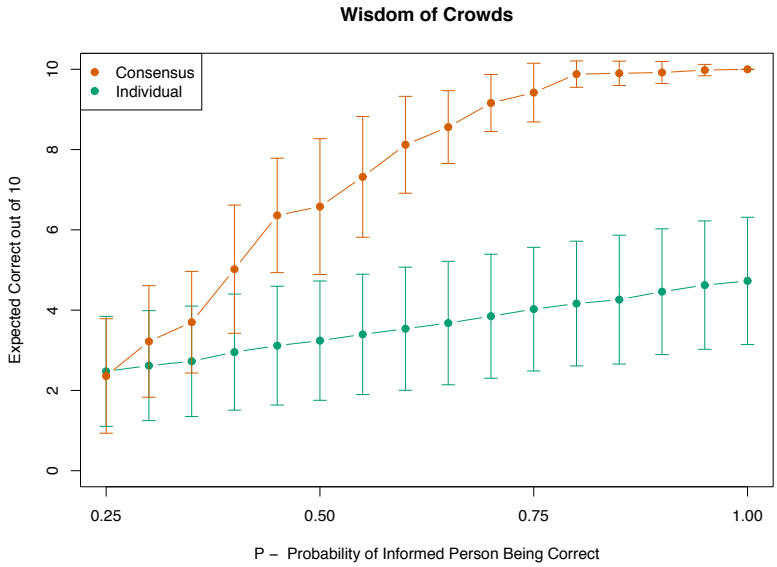

Figure 2: Simulation: wisdom of the crowd. The probability of the committee being right (and its standard deviation in errorbars) overlay with the probability (and standard deviation) of each member individually being right (ESL figure 8.11)

1.5 Summary

We have seen quantitatively the following general statement: A committee of members decides by majority vote. Decisions are correlated with correlation ρ and each member is right with probability p. If $p > 0.5$, the committee's decision will improve with the number of members T in two ways: it will have a higher probability of being right, and its decision will be more consistent (less variable). If $\rho > 0$, increasing T will increase the probability (that the committee's decision is right) up to a certain point, so ρ gives a bound on the improvement possible by moving from a single member ($T = 1$) to a committee ($T > 1$).

2 Committee methods in machine learning

Back to machine learning. Suppose we had T training samples of size m chosen independently from $\mathcal{X}$ according to some distribution $\mathcal{D}$. Denote them by $S_1, \ldots, S_T$. Suppose have a learning algorithm $\mathcal{A}$ and train it on each of the training samples, to obtain $h_{S_1}, \ldots, h_{S_T}$. The prediction $h_{S_t}(x)$ on some sample $x \in \mathcal{X}$ is independent of all other predictions of the other trained rules, and has the same distribution. So if we used $h_{S_1}, \ldots, h_{S_T}$ in a committee - using majority vote - we have the situation from above. The generalization loss will improve with T, tending to 0 as $T \to \infty$ (if any rule h_{S_t} separately has generalization loss < 0.5). As we saw above, the variance of the prediction will decrease as $1/T$.

However, in batch learning, we don't have T training samples. We just have one. So why not train the same algorithm $\mathcal{A}$ again and again on the same training sample S? well, that won't do much good - the predictions will be identical - perfectly correlated.

So the first magic we would like to perform is to create T training samples from our (single) original training sample S, in a way that will simulate fresh independent draws of new training samples of size m according to $\mathcal{D}$.

Committee methods - Definition. Committee methods are **"meta-algorithms"**. In a committee method, we take an existing learner $\mathcal{A}$ (which we call the "base" learner, or sometimes the "weak" learner, for reasons we will see below) and apply it to a sequence of T "artificial" training samples.

In this lecture we work with classification. The label set is $\mathcal{Y} = \pm 1$ and the committee decides by $h(x) = \text{sign}\left(\sum_{t=1}^{T} h_t(x)\right)$. Everything here applies to regression as well: in a regression problem, the committee decides by $h(x) = T^{-1}\left(\sum_{t=1}^{T} h_t(x)\right)$.

We are going to see two very different ideas for building the committee member rules. But After the committee is built, it is averaged the same way in all cases. Sometimes the committee's decision will be a weighted average, to allow some members tohave more weight than others: $h(x) = \text{sign}\left(\sum_{t=1}^{T} w_t h_t(x)\right)$, for some $w_t \geqslant 0$, $\sum w_t = 1$.

3 The Bootstrap

Here comes the first magic. Creating new "artificial" training samples from the one training sample S we have seems impossible. But yet it actually is, and the fact that it is is one of the most groundbreaking ideas of statistics in the 20th century.

Given a training sample $S = \{(x_i, y_i)\}_{i=1}^m$ we are going to construct a new training sample S^{*1} as follows. We are going to sample m times **with replacements** from the set S. The first sample we draw from S will be denoted (x_1^{*1}, y_1^{*1}). The second sample we draw will be denoted (x_2^{*1}, y_2^{*1}), and so on. So we now have a sample

$$S^{*1} = \{(x_i^{*1}, y_i^{*1})\}_{i=1}^m$$

Of course, since we sampled from S with replacements, there might be repeated samples in S^{*1}, even if S itself had no repeated samples. Now we can repeat this process B times, obtaining B training samples, each of length m: $S^{*1}, \ldots, S^{*B}$. The samples in the b-th training sample will be denoted

$$S^{*b} = \{(x_i^{*b}, y_i^{*b})\}_{i=1}^m .$$

This method of new training samples is called The Bootstrap, and the sample S^{*b} is called a bootstrap sample created from S.

3.1 Why does the Bootstrap work?

Assume for a moment that samples in our learning problem are i.i.d samples from an unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. We are hoping that each Bootstrap from S somehow behaves like a fresh i.i.d sample from $\mathcal{D}$ itself.

It may seem at a first glance crazy that bootstrap samples can serve us instead of fresh, i.i.d samples from $\mathcal{D}$. But in fact it is often the case. Why?

Given a training sample S (assume for simplicity that all the points of S are distinct) let's define the **empirical distribution** $\widehat{\mathcal{D}}_S$ induced by S on $\mathcal{X} \times \mathcal{Y}$ as the following probability distribution on $\mathcal{X} \times \mathcal{Y}$: for a subset $C \subset \mathcal{X} \times \mathcal{Y}$, define

$$\widehat{\mathcal{D}}_S((X,Y) = (x,y)) = \begin{cases} \frac{1}{m} & (x,y) \in S \\ 0 & (x,y) \notin S \end{cases}$$

or equivalently, for any $C \subset \mathcal{X} \times \mathcal{Y}$,

$$\widehat{\mathcal{D}}_S(C) := \frac{|C \cap S|}{m}\,.$$

Observe that this is equivalent to putting a probability mass of $1/m$ on each of the points of S, and zero mass on all other points in $\mathcal{X} \times \mathcal{Y}$.

Now observe that a bootstrap sample S^{*b} is just an i.i.d draw of m points from the empirical distribution $\widehat{\mathcal{D}}_S$ induced by the one training sample we have, S.

As m grows, namely as S becomes larger, the empirical distribution $\widehat{\mathcal{D}}_S$ converges in distribution to $\mathcal{D}$. The idea behind the bootstrap is that, if $\widehat{\mathcal{D}}_S$ is not so different from $\mathcal{D}$, then m i.i.d draws from $\widehat{\mathcal{D}}_S$ is a good approximation to m i.i.d draws from $\mathcal{D}$.

One way to see the convergence of the empirical distribution to the underlying distribution is on the real line:

Exercise. Let $X_1, \ldots, X_m$ be i.i.d random variables with some distribution F on the real line $\mathbf{R}$. Let $x_i \in \mathbb{R}$ be a sampled value of X_i. Show that the CDF (cumulative distribution function) of this sample is a step function, increasing from 0 to 1 (as all CDFs should), with jump of size $1/m$ at each value x_i (see Figure 3). In simulation, take F to be, say, $\mathcal{N}(0,1)$. For each value $m = 10, 100, 1000$, draw a sample and plot its empirical CDF - and overlay the CDF of $\mathcal{N}(0,1)$. Recall that on the real line, convergence in distribution is equivalent to convergence of the CDFs to a limiting CDF at the continuity points of the limiting CDF.

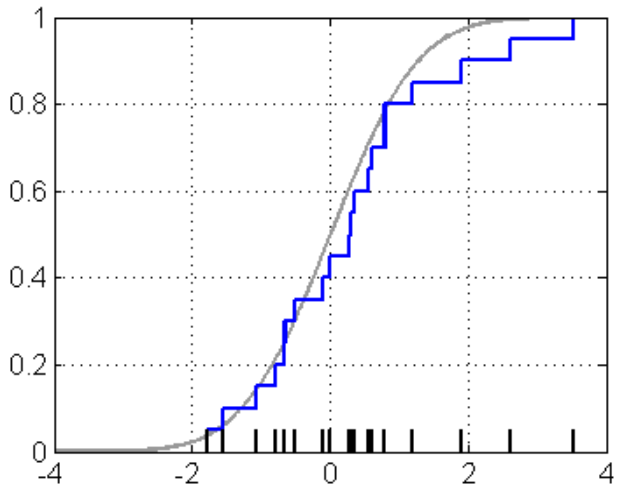

Figure 3: CDF of a probability distribution on the real line, and empirical CDF of an i.i.d sample from that distribution. Black lines on the horizontal axis show the random sample.

Why the name "The Bootstrap"? We're seemingly creating new datasets out of nothing, as if we were pulling ourselves up by the straps of our own boots. As you may know, there was only one man strong enough to do that - the Baron Munchausen.
Here's a question you might ask: How many point of S are left **out** of each bootstrap sample, typically? Answer: About a third.

Exercise. Show that, for m large, about 37% of data points are left out of a bootstrap sample created from S.

4 Bagging

The idea of Bootstrap samples can be used whenever we would like to create new artificial samples from our only training sample S. It has many uses throughout machine learning, statistics and data science. **Bagging** is a nickname for a straightforward use of the Boostrap in machine learning, to improve accuracy of an existing supervised machine learning algorithm.

We start with a "base" learning algorithm $\mathcal{A}$ and a training sample S. We choose T (later we'll discuss how to choose it) and form T bootstrap training samples, $S^{*1}, \ldots, S^{*T}$, each of size m. We then train our learner **separately** on each of the T bootstrap training samples. We form the committee $h_{S^{*1}}, \ldots, h_{S^{*T}}$. We store all T trained models. When we need to classify a new test sample $x \in \mathcal{X}$, we run x through all the rules and classify using the majority vote of the committee,

$$h_{bag}(x) := sign\left(\sum_{t=1}^{T} h_{S^{*t}}(x)\right)$$

For example, if we run Bagging on top of the Decision Tree classifier, we'll obtain a committee of decision trees:

Figure 4: The Great Baron Munchausen pulling himself up

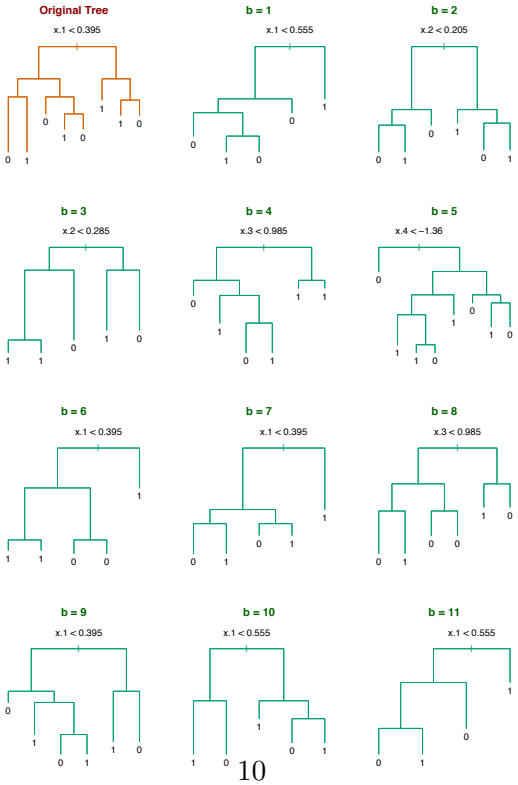

Figure 5: Collection of Bagged Decision Trees. (Source: ESL)

Handling repeated samples. Note that our learner $\mathcal{A}$ must know how to handle repeated samples. We may have them anyway in S, but running on a bootstrap sample we are sure to have them. Some learning algorithms don't like repeated samples - as they cause numerical problems (for example, linear and logistic regression.) Some really don't care (for example, decision trees and k-NN).

4.1 This is shockingly effective

Does Bagging a learning algorithm reduce its generalization error? This is what the original paper observed:

Data Set	$\bar{e}_S$	$\bar{e}_B$	**Decrease**
waveform	29.1	19.3	34%
heart	4.9	2.8	43%
breast cancer	5.9	3.7	37%
ionosphere	11.2	7.9	29%
diabetes	25.3	23.9	6%
glass	30.4	23.6	22%
soybean	8.6	6.8	21%

Figure 6: Improvement of Bagging Decision Trees over a single tree. From the original paper Shang and Breiman, Distribution Based Trees Are More Accurate

So a simple and straightforward trick can hugely reduce generalization risk.

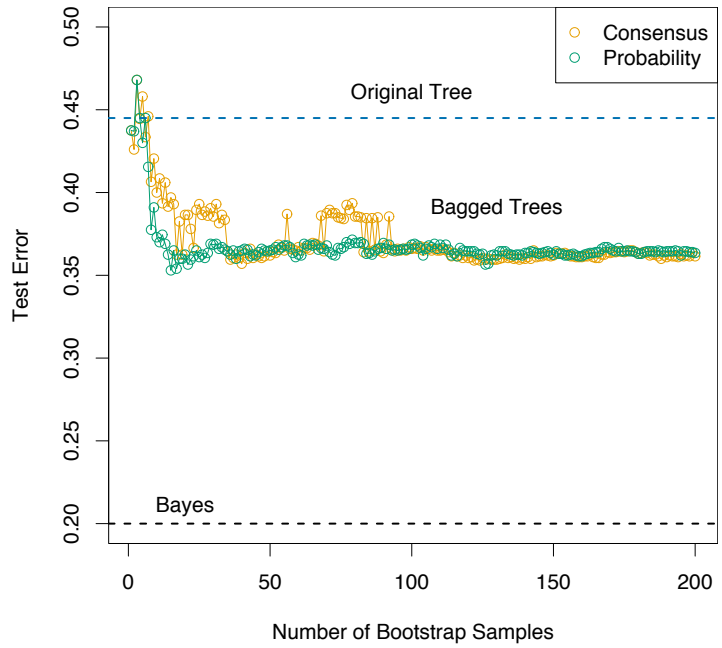

Figure 7: Test error of simple Bagging of decision trees, over T the number of bagged trees (Source: ESL)

4.2 Bagging reduces variance

We saw that a committee majority vote reduces variance - but only to a certain degree, which is determined by the correlation between committee members. So, we can expect bagging to reduce variance as T increases - and therefore to reduce the generalization error - but only to a certain degree, determined by the correlation between the bagged prediction rules. So, Bagging can be improved by somehow de-correlating the bagged prediction rules.

4.3 Decorrelation

How do we de-correlate the committee members - namely, cause their predictions somehow to be less correlated? One way to do this is by handicapping (restricting) each learner a little, in a random way, and hope that the performance gain (in bagging them) due to de-correlation is more than the performance loss to each learner by handicapping. The most well know example of this principle is **Random Forests.**

4.4 Random Forest: Bagging of Decision Trees + De-correlation

Recall the Decision Tree classification algorithm over $\mathcal{X} = \mathbb{R}^d$. We have a training sample S with m points. The Random Forest classifier is obtained by using Bagging on top of the Decision Tree algorithm, **with an important twist** for de-correlation: the algorithm has a tuning parameter $k \leqslant d$. When growing each decision tree, in each split, we choose k out of the d coordinate uniformly at random, and only choose the split among these k coordinates. Formally:

- The tuning parameters are:

- $R \in \mathbb{N}$, the maximum depth of each tree
- m_{min}, the minimal number of training samples in any leaf of any of the trees
- T, the number of Bagging samples (number of trees in the forest)
- k, the number of coordinates allows in choosing each split.
- (There is an additional tuning parameter for **pruning** a decision tree which we'll discuss in a future lecture.)

- For each $t = 1 \dots T$:
  - Draw a Bootstrap sample S^{*t} from S
  - Train a decision tree $h_{S^{*t}}$ on the sample S^{*t}. While growing the tree, in each split do the following:
    * Select k coordinates from $\{1, \dots d\}$ uniformly at random
    * Pick the best (coordinate,split-point) combination using only the k coordinates chosen
    * Split on the best combination
  - Do not split a box if the maximal depth R or the minimal number of training samples m_{min} are reached.
- Output the grown trees $h_{S^{*1}}, \dots, h_{S^{*T}}$.

This de-correlation trick works: pretty much on every classification problem you'll work on, you'll observe something like the next plot: Bagging trees is much better than a single tree, and Random Forest (Bagging with the de-correlation trick) is better than just Bagging trees. (And, sometimes, Boosting trees is better than both - but we're coming up to that.)

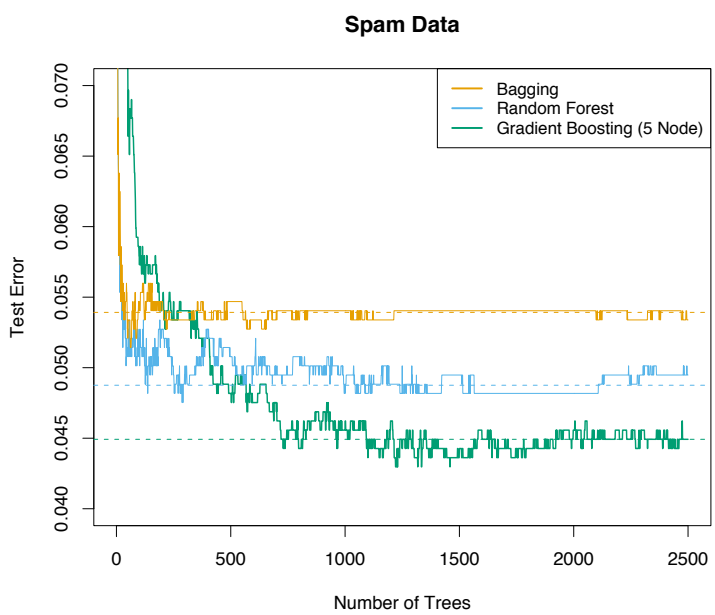

Figure 8: Test error of simple Bagging of decision trees (no de-correlation), Random Forests, and Gradient Boosting of Trees. (Source: ESL)

4.5 Some discussion points about Bagging

Can Bagging hurt us?

Always remember that a committee of fools (a committee where each member has probability $p < 0.5$ to make the right decision) makes worse decisions than a single member. So, when our base learner is so poor that its generalization loss is less than 0.5 we shouldn't use Bagging.

What are the disadvantages of Bagging?

- We need to train T models, not just one
- For prediction on new samples, we need to store T models, not just one
- Loss of interpretability: it's harder to understand why the committee made a decision - we need to understand the decision of each of the T members

Bagging and predicted class probabilities

Question: Can we use the **proportion** of the committee members who voted +1 as a predicted class probability?

Answer: It's not a good idea. Estimated class probabilities are estimates of $\mathbb{P}\{Y = +1, | X = x\}$. The proportion of members who voted +1 estimates $\mathbb{P}\{h_S(x) = +1\}$, which is a different quantity.

Parallel implementation of Bagging and Random Forests

From the computational perspective, it is important to note that Bagging in general (and Random Forests in particular) is **embarrassingly parallelizable**. When training a Bagging model with T committee members, we can use T machines in parallel, each using its own random seed to select Bootstrap samples (and random splits, in Random Forest). The machines do not need to interact; when each machine is done, it returns the committee member h_t to the master node.

Decision boundary in bagging

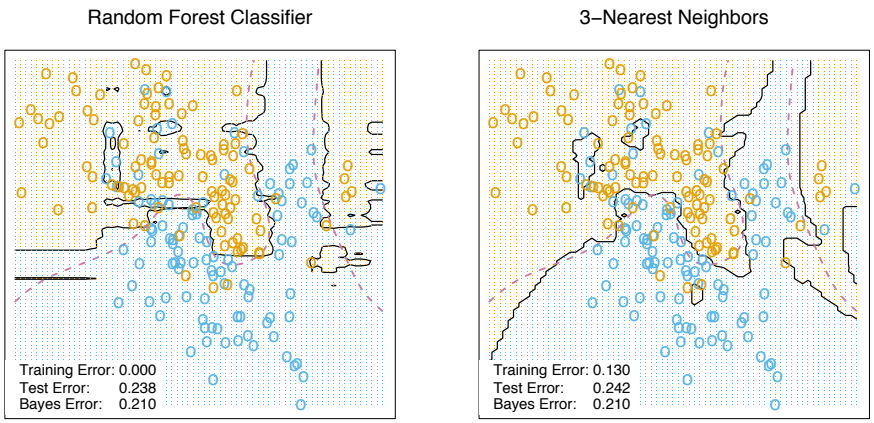

Figure 9: Left: Decision boundary of a Random Forest classifier. Right: Decision boundary of a $3-NN$ classifier. Observe that the Random Forest tends to have axis-parallel boundaries. (Source: ESL)

4.6 Random Forest classifier summary

Random Forest is a very popular classifier. As it doesn't overfit for T (number of trees) too large, we just try to avoid making T too large for efficiency reasons. For choosing k (the number of random coordinates allowed for each split), the rule of thumb is $k := \sqrt{d}$ where d the ambient dimension, $\mathcal{X} = \mathbb{R}^d$.

Exercise: Complete the following summary of the Random Forest Classifier (note that we still didn't learn how to **prune** each decision tree in the forest.)

- Hypothesis class
- Learning principle for training model (choosing $f \in \mathcal{H}$):
- Computational implementation of learning principle:
- How to make predictions on new samples:
- Interpretable
- Estimates class probabilities
- Family of models
- Time complexity for training, and for predicting on a new sample
- How to store trained model

5 Boosting

Bootstrap was magic of the following kind: we take a single training sample S and turn it into many training samples. Bagging uses this magic by training a model over these "new" training samples, and averaging the result to reduce the variance and hence the generalization error.

Boosting is magic of a different kind. In Boosting we take a "weak" learning algorithm - an algorithm with better-than-random but possibly not so good accuracy (accuracy = generalization error) and **boost** it - using a clever committee method - to obtain a learning algorithm with good accuracy.

The core magical idea of Boosting is a completely different idea for creating a committee of prediction rule from a base learning algorithm $\mathcal{A}$ and a single training sample S. In Bagging, we "pretended" to have fresh training samples $S_1, \ldots, S_T$, and each committee member trained on a different sample. In Boosting, we go even further and "pretend" to have **different underlying distributions $\mathcal{D}$ from which the training sample is drawn**.

More specifically, in Boosting each committee member h_t is the result of running $\mathcal{A}$ against a training sample S_t that mimics an i.i.d sample of size m from a **different distribution D^t**. Whereas in Bagging each committee member is trained independently of all other members, in Boosting the committee members are trained sequentially - one after the other - and each is an improvement, in some sense, on the previous one.

The clever idea behind Boosting is that after we finish training h_t, based on the distribution D^t, we update the distribution in a way that **increases the distribution at training samples where h_t was wrong**. This way, h_{t+1} will try very hard not to be wrong on those particular samples, and so on.

Here is a cartoon of how Boosting iterations progress:

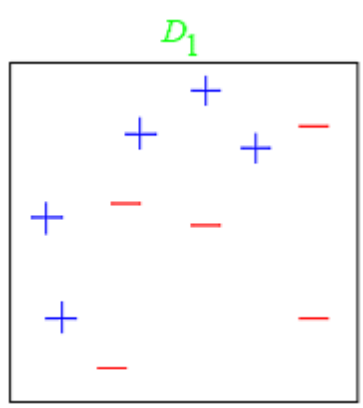

Figure 10: Original problem. Uniform distribution D^1.

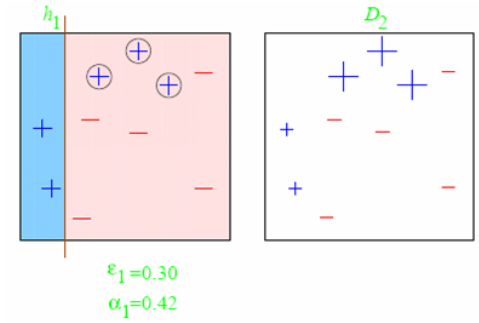

Figure 11: Iteration 1. Left: h_1 with D^1. Right: D^2

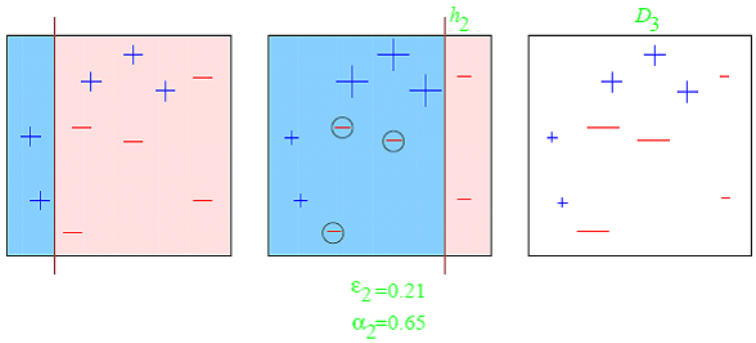

Figure 12: Iteration 2. Left: h_1 with D^1. Center: h_2 with D^2. Right: D^3

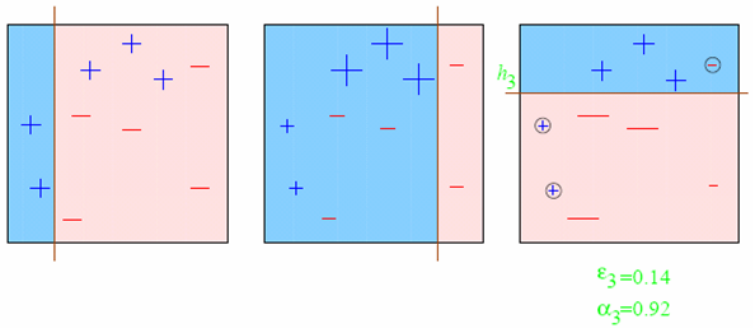

Figure 13: Iterations 3. Left: h_1 with D^1. Center: h_2 with D^2. Right: h^3 with D^3

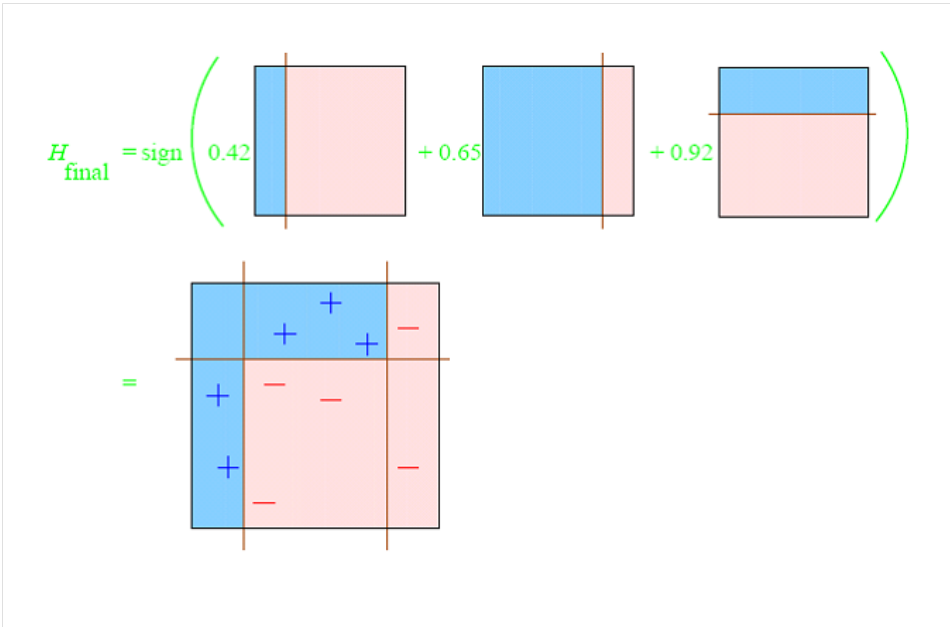

Figure 14: The Boosting committee

5.1 Classification problem with a weighted sample

But first, we have to understand what is meant, exactly, by "running $\mathcal{A}$ against the training sample S with distribution D^t".

One way to interpret this is to take a **weighted Bootstrap** sample from S, where the probability of selecting $(x, y) \in S$ is proportional to $D^t(x, y)$.

A simpler way to interpret this is as follows. If $\mathcal{A}$ uses the ERM principle, say for standard misclassification ($0-1$ loss), namely, looking to minimize the empirical risk,

$$L_S(h) = \sum_{i=1}^{m} \mathbf{1}_{[y_i \neq h(x_i)]}$$

then we can use S itself (not any bootstrap sample) and have the base learner minimize the **weighted** empirical risk

$$L_{S,D^t}(h) = \sum_{i=1}^{m} D_i^t \mathbf{1}_{[y_i \neq h(x_i)]}$$

where for each $(x_i, y_i) \in S$ we write $D_i^t := D^t(x_i, y_i)$, so that $\sum_{i=1}^{m} D_i^t = 1$.

Observe that these two interpretations are equivalent in expectation. Indeed, the expected number of times for a sample (x_i, y_i) to appear in the weighted Bootstrap sample is D_i^t, and so it would (in expectation) appear D_i^t) times in the empirical risk sum.

Note that we usually prefer second option (using weighted empirical risk) to the first option (using weighted bootstrap). It's more computationally efficient, and does not require worrying about repeated samples. However, the first option (using weighted bootstrap) is always

available. The second option (using weighted empirical risk) is not always possible, and is implemented ad-hoc for the particular base learner we are boosting.

Exercise: To help you understand this point, describe how you would implement a Decision Tree with each of the two methods:

- Using weighted empirical risk: How would you change the Decision Tree algorithm we've seen (CART) to work with a given weight vector D^t over the training sample S? (Hint: what is the best splitting now that we have weights?)
- Using weighted Bootstrap: How would you How would you change the Decision Tree algorithm we've seen (CART) to work with a given weight vector without changing the splitting algorithm, namely, by giving the algorithm a different training sample selected by weighted Bootstrap? Will the algorithm work with repeated samples?

It is much less trivial to adapt learning algorithms that do not use the ERM principle. For example, Soft SVM can be adapted to work with weights, by penalizing the slack variables - but this is not trivial[2].

5.2 Adaboost

There are many Boosting meta-algorithms. The one we will learn here was the original one, known as **Adaboost** (for "**Ada**ptive **Boost**ing")

Adaboost, as a form of boosting, is characterized as by the following statements:

- Set the initial distribution to be uniform, $D_i^1 = 1/m$, $i = 1, \ldots, m$
- Use exponential updates for the distribution

$$D_i^{t+1} \leftarrow \frac{D_i^t \cdot e^{-w_t y_i h_t(x_i)}}{\sum_{j=1}^m D_j^t \cdot e^{-w_t y_j h_t(x_j)}}$$

  for some **exponent** $w_t > 0$. (Notice how if point i is classified correctly then $y_i h_t(x_i) = 1$, so its weight goes down in the next iteration. Conversely, if point i is classified incorrectly then $y_i h_t(x_i) = -1$, so its weight goes up.)

- Choose the exponent w_t exactly such that

$$\sum_{i=1}^m D_i^{t+1} \mathbf{1}_{[y_i \neq h_t(x_i)]} = \frac{1}{2}\,.$$

  (We'll soon discuss why.)

- Each committee member h_t votes with weight w_t, so that the label predicted by the committee is

$$h_{boost}(x) := sign\left(\sum_{t=1}^T w_t h_t(x)\right)$$

[2]Look for "boosting support vector machines."

The idea is simple: from iteration t to iteration $t+1$, we want to **increase** the weights of samples misclassified by h_t (where $y_i h_t(x_i) = -1$) and **decrease** the weights of samples correctly classified by h_t. We want to make the classification problem "maximally hard" in the sense that weighted empirical risk of h_t, with respect to the updated weights D^{t+1}, is the worse possible, namely 1/2. Finally, the prediction rules vote in the committee with weights w_t.

Claim: The exponent we are looking for is

$$w_t := \frac{1}{2} \log \left(\frac{1}{\epsilon_t} - 1 \right)$$

where $\epsilon_t = \sum_{i=1}^m D_i^t \mathbf{1}_{[y_i \neq h(x_i)]}$ is the weighted empirical risk of h_t.

Proof:

$$\sum_{i=1}^m D_i^{t+1} \mathbf{1}_{[y_i \neq h_t(\mathbf{x}_i)]} = \frac{\sum_{i=1}^m D_i^t e^{-\mathbf{w}_t y_i h_t(\mathbf{x}_i)} \mathbf{1}_{[y_i \neq h_t(\mathbf{x}_i)]}}{\sum_{j=1}^m D_j^t e^{-\mathbf{w}_t y_j h_t(\mathbf{x}_j)}}$$

$$= \frac{e^{\mathbf{w}_t} \epsilon_t}{e^{\mathbf{w}_t} \epsilon_t + e^{-\mathbf{w}_t}(1-\epsilon_t)} = \frac{\epsilon_t}{\epsilon_t + e^{-2\mathbf{w}_t}(1-\epsilon_t)}$$
$$= \frac{\epsilon_t}{\epsilon_t + \frac{\epsilon_t}{1-\epsilon_t}(1-\epsilon_t)} = \frac{1}{2} .$$

You may be wondering why the right weights for the committee votes are given by the same exponents w_t we used to update the distribution. This will become clear next.

To recap, here is the Adaboost meta-algorithm:

- **input:** training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$; base ("weak") learner $\mathcal{A}$ that takes both a training sample of size m and a distribution on m points; number of rounds T.
- **initialize** $D^1 = (\frac{1}{m}, \ldots, \frac{1}{m})$
- **for** $t = 1, \ldots, T$
  - invoke base learner $h_t = \mathcal{A}(D^t, S)$
  - compute $\epsilon_t = \sum_{i=1}^m D_i^t \mathbf{1}_{[y_i \neq h_t(\mathbf{x}_i)]}$
  - let $w_t := \frac{1}{2} \log \left(\frac{1}{\epsilon_t} - 1 \right)$
  - update $D_i^{t+1} = \frac{D_i^t \exp(-w_t y_i h_t(\mathbf{x}_i))}{\sum_{j=1}^m D_j^t \exp(-w_t y_j h_t(\mathbf{x}_j))}$ for all $i = 1, \ldots, m$
- **output** the hypothesis $h_{boost}(\mathbf{x}) = sign \left(\sum_{t=1}^T w_t h_t(\mathbf{x}) \right)$.

5.3 PAC view of boosting

Historically, Boosting appeared as an answer to a fascinating question. Let us formulate this question in PAC framework.

Definition 1 (γ-weak-learner) *A learning algorithm $\mathcal{A}$ is a γ-weak-learner for an hypothesis class $\mathcal{H}$ if there exists a function $m_{\mathcal{H}} : (0,1) \to \mathbb{N}$ such that for every $0 < \delta < 1$, for every distribution $\mathcal{D}$ over the sample space $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{\pm\}$, if the realizabiity assumption holds with respect to $\mathcal{H}, \mathcal{D}, f$, then when running $\mathcal{A}$ on a training sample of $m \geqslant m_{\mathcal{H}}(\delta)$ i.i.d samples drawn according to $\mathcal{D}$ and labeled by f, the algorithm returns an hypothesis $h_S = \mathcal{A}(S)$ such that with probability at least $1 - \delta$ (with respect to choice of the training sample S), we have $L_{\mathcal{D},f}(h_S) \leqslant 1/2 - \gamma$.*

An hypothesis class $\mathcal{H}$ is γ-weak-learnable if there exists a γ-weak-learner for $\mathcal{H}$.

How is this different than PAC-learnability? If an hypothesis class $\mathcal{H}$ is PAC-learnable, then for **every** (ϵ, δ) there exists a learner $\mathcal{A}$. This means that we can learn and generalize a labeling function from $\mathcal{H}$ to any accuracy ϵ we want. But if $\mathcal{H}$ is γ-weak-learnable, for any δ **and just for** $\epsilon = 1/2 - \gamma$ there is a learner $\mathcal{A}$. We may not be able to find a learner that has better accuracy (lower ϵ).

The question that motivated Boosting was the following:

- Suppose that $\mathcal{H}$ is PAC-learnable. Then we know that the rule $ERM_{\mathcal{H}}$ will learn (namely, will be probably approximately correct etc) with a near-minimal number of samples.
- But what if $ERM_{\mathcal{H}}$ is computationally hard? (we've seen examples)
- Assume we can find a **simple** hypothesis class (a "base hypothesis class") $\mathcal{H}_{base}$, such that $ERM_{\mathcal{H}_{base}}$ (choosing the hypothesis in $\mathcal{H}_{base}$ with lowest empirical risk) is computationally efficient, and is γ-weak-learner for $\mathcal{H}$ for some γ.
- This means that we have a computationally efficient way to learn with accuracy $1/2 - \gamma$, for some γ. Maybe we can't find an efficient learner with better γ.
- Is there a way to **boost** $ERM_{\mathcal{H}_{base}}$ in a computationally efficient way, and create a computationally efficient learner $\mathcal{A}$ which is close to minimizing ERM over $\mathcal{H}$?

For example, think about Decision trees. We saw that the ERM learner is not computationally feasible on this hypothesis class. But a small tree may be able to achieve accuracy (over a sample labeled by a larger tree) which is not great, but better than random.

Well, as the following theorem shows, Adaboost does just that. (We won't prove this theorem - you can see the proof in UML 10.2).

Theorem. Let S be a training set. Assume that at each iteration of Adaboost, the base learner returns a prediction rule (hypothesis h_t) for which the weighted empirical risk satisfies

$$\sum_{i=1}^{m} D_i^t \mathbf{1}_{[y_i \neq h(x_i)]} \leqslant \frac{1}{2} - \gamma \,.$$

Then the (standard, non-weighted) empirical risk of the output prediction rule of Adaboost, h_{boost}, (the weighted committee vote) satisfies

$$L_S(h_{boost}) \equiv \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}_{[y_i \neq h_{boost}(x_i)]} \leqslant e^{-2\gamma^2 T} \,.$$

5.4 Bias and variance in boosting

The hope is, of course, that we're not overfitting, so that low empirical risk will imply low generalization loss.

Suppose we run T iterations of Adaboost over a learner $\mathcal{A}_{base}$ that returns hypothesis from $\mathcal{H}_{base}$. What is the effective hypothesis class we have now, and how large is it?

Well, Adaboost with T iterations will return a function from the hypothesis class

$$\mathcal{H}_T = \left\{ x \mapsto \sum_{t=1}^{T} w_t h_t(x) \,\middle|\, w_1 \dots w_T \in [0, \infty),\ \sum_t w_t = 1,\ h_1 \dots, h_T \in \mathcal{H}_{base} \right\}$$

namely convex combinations of hypotheses from $\mathcal{H}_{base}$. So $\mathcal{H}_t$ becomes larger (contains more functions) as T grows. But it doesn't grow too fast with T.

For example, we have a canonical way to measure the "size" of $\mathcal{H}_T$. While we won't go into the details, under certain conditions, $VCdim(\mathcal{H}_T)$ is roughly $T \cdot VCdim(\mathcal{H}_{base})$. So we can expect Boosting to increase the variance (compared with the base learner) as T increases, but "not too fast".

On the other hand, it's clear that Boosting decreases bias - that is obvious from the fact that the empirical risk decreases as T grows - indeed $\mathcal{H}_T$ is able to come closer and closer to the labeling function on the training set. And the fact that empirical risk decreases **exponentially** with T tells us that bias decreases quite quickly.

Overall, Boosting typically decreases bias much faster than it increases variance, which is why it typically improves generalization loss quite dramatically.

Question for you: If we use T too large, will boosting overfit?

5.5 It's often better to Boost very simple learners

Very often we see that boosting ERM over a very simple base hypothesis class is better than boosting ERM over a more complicated class. For example, here is the test error (number of misclassification errors on a test set the algorithm has never seen) of boosting **Decision stumps** - Decision trees with a single split - over number of iterations T, compared with the test error of a single stump, and with a single large Decision tree:

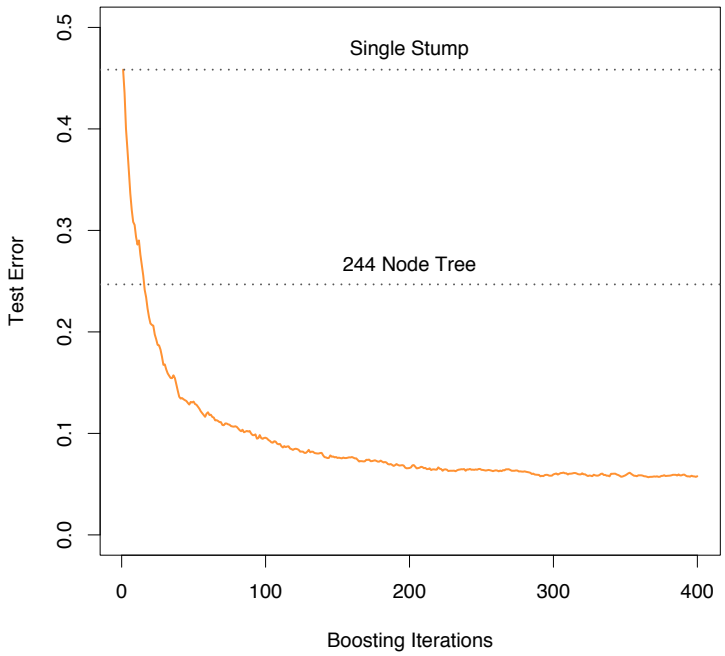

Figure 15: Test error of boosting decision stumps (single level decision trees), with Adaboost over the number of boosting iterations T.

6 Bagging vs Boosting - Comparison

Bagging and Boosting are both committee methods, but they are very different. Let's compare them:

	Bagging	**Boosting**
Learns committee members:	in parallel	sequentially
Dataset for each committee member:	bootstrap training samples	weighted bootstrap **or** original S with weighted ERM
De-correlation:	recommended	not necessary
When T is too large	Does not overfit	may overfit
Cause of improvement in generalization error:	reduces variance	reduces bias
With decision trees, use:	deep trees	shallow trees
Parallel compute implementation:	yes - easy	no
Committee vote:	unweighted	weighted

Table 1: Comparison of Bagging and Boosting

7 Summary

We saw that a committee of learners using majority vote will have better accuracy than a single member if each member is better than a random guess; the improvement in accuracy due to the majority vote improves as the size of the committee grows, but is bounded from below by the correlation between the members.

We saw three general methods:

- **Bootstrap** is a method for generating "new" training sample from the one training sample we have.
- **Bagging** is a committee method where we run the learner against bootstrap samples. Learners are unrelated to each other. All have the same voting weight.
- **Boosting** is a committee method where we run the learner sequentially on weighted bootstrap samples. The weights are larger for samples where we made a mistake in the last iteration. Learners vote with weights related to their empirical loss.

These methods implement three general principles:

- We can create "artificial" training sets from our one training set S by sampling from S with replacements. This method is known as **The Bootstrap.** In a typical Bootstrap sample, about a third of the points a left out and others appear more than once.
- We can create a learner with **improved accuracy** and **reduced variance** by averaging base learners. The base learners **must** be better than a random guess. When each base learner gets a Bootstrap training sample, this is called **Bagging.** Bagging can be done in parallel as each prediction rule is created independently of the others. The prediction accuracy of the Bagging learner improves if the different prediction rules used are as de-correlated as possible. (Example: Random Forest is a classifier than achieves de-correlation by restricting each split in each tree to a random subset of coordinates.)
- We can create a learner with with improved accuracy by **boosting** a base learner. The key idea behind Boosting is working with a probability distribution over S. Boosting means creating a **weighted** committee of prediction rules. Rules are created sequentially (not in parallel). Each rule is created the previous rule by modifying the distribution in such a way that misclassified training samples get an increased weight. (Example: Adaboost is a Boosting method that uses exponential updates to the probability distribution on S, such that the weighted empirical risk of the previous rule according to the updated distribution is exactly 1/2 - the worse it can be.)