Outline

In the interest of keeping this as broad as possible, we hope to start with by discussing the general multiplicative weights update framework, with multiple experts. We will introduce the Adaboost Algorithm in the context of Decision trees as a case study of application of the Multiplicative weights method. After going over it we will discuss the advantages/disadvantages of boosting vs bagging, and have students experiment with simple code examples in a jupyter notebook using sklearn. And while we may not emphasize it too much, in the note we shall also include the math behind how we are able to turn a weak base learning into a strong one.

Real life application - Netflix challenge

Prereq stuff - Polynomial interpolation, K nearest neighbors

In general, multiplicative weights.

Experts Frameworok

Boosting - AdaBoost. (Have a footer introducing Gradient boosting)

Bagging

Application:

Decision Trees to Random forests?

In September 2009, The BellKor's Pragmatic Chaos team won a grand total of 1 million USD in the infamous Netflix Challenge. In this challenge, competitors were striving to come up with a better recommendation algorithm than the one currently used by Netflix's own algorithm. Bellkor and his team were able to come up with an algorithm that would beat Netflix's algorithm by a whole 10 percent. How, one would ask? It turned out that a single model was not enough. They used a method called gradient boosted decision trees, which instead aggregated the results of over 500 models. This is one prime example of a technique we will discuss today, known as **ensemble methods**. In this note, we first discuss intuition behind the method. Then, we introduce the **multiplicative weights update** framework in which we can analyze ensemble methods. Next, we will examine two ensemble methods in machine learning known as **Boosting** and **Bagging**, which will make up the bulk of

the note. Finally, we conclude with a widely-used application of this technique, **random forests.**

Introduction

Suppose you are tackling a problem with machine learning and are considering a few possible approaches. An intuitive question one may ask is, why must we constrain ourselves to a single model? Why not just try all the approaches and somehow combine their powers? This is truly the heart of an ensemble method. In an **ensemble method**, we form a stronger hypothesis by intelligently combining the results of multiple hypotheses. The idea is rather simple and transcendent. At Berkeley, you have seen this concept of combining multiple sources in previous courses such as EECS16B through **Lagrange interpolation** or orthogonal matching pursuit (OMP).

In Lagrange interpolation, the heart of the algorithm is to aggregate the information from multiple (n+1) points to allow us to classify our desired polynomial from the space of degree <= n polynomials. Here, without going into too much detail, we create “basis polynomials” in which we then weigh by their corresponding y coordinates and combine linearly into our resulting polynomial. Clearly there is some merit in ensembles. But, before we dive straight into the ensemble methods in machine learning, first we shall introduce a more general **multiplicative weights update** framework in which we can examine these methods for a fuller understanding of its application in machine learning.

Multiplicative Weights Algorithm (MWU)

Let us define some terminology first. Historically, computer scientists have referred to the sources that we aggregate as “experts.” The first and most resounding question is, given our experts, how do we combine their results? Perhaps we can take an average, or a weighted average if we believe that some experts are better than others (e.g the existence of a “best” expert). Perhaps we can take the mode, or most common result. Given the results of their predictions, are there ways to update our strategy? The answer to these questions can be explained by the multiplicative weights algorithm.

There are two possibilities for the experts. Either they are all the same, or there is some difference between them, implying the existence of one or more best expert(s). If they are all the same, then there is no decision to be made. The more interesting case is when the experts are of different skill levels (some experts are correct more often than others), then we must somehow iteratively make choices, deciding which experts are best at any given moment and accordingly weighing their advice more.

This suggests that in the multiplicative weights algorithm, we should assign a weight to each one of the experts and update that based on incoming data. Intuitively, we should update our weights such that if an expert is correct, we would want to weigh their advice more, and if they are incorrect, we would want to weigh their advice less. Thus over time, this ensures that we make decisions more heavily based on the advice of the stronger experts and thus maximizing our overall performance. How do we make our choice at each step? In this model, we assume that there is some adversary working against us, and thus it is in our best interest to make decisions probabilistically. Specifically detailing the algorithm below,

The Algorithm (1):

The algorithm proceeds as follows. It works with a parameter $0 \leq \epsilon \leq 1/2$ that is hardcoded in the algorithm, and it maintains, at each time step t a set of *weights* $w^{(t)} = (w_1^{(t)}, \ldots, w_n^{(t)})$ associated to the strategies, defined as

$$w_i^{(0)} = 1$$

$$w_i^{(t+1)} = w_i^{(t)} \cdot (1-\epsilon)^{\ell_i^{(t)}}$$

The effect of this definition is that strategies that produced high losses in the past have small weight, and strategies that produces low losses have higher weight. At every step, the algorithm chooses allocations proportionally to the weights

$$x_i^{(t)} = \frac{w_i^{(t)}}{\sum_j w_j^{(t)}}$$

The optimal parameter epsilon may be calculated analytically, $\epsilon = \sqrt{\frac{\ln n}{T}}$, the derivation of which we will not show. Furthermore, note in this case, we are not assuming any prior on the experts' skill levels, and thus they are initialized to the same

The performance of our algorithm is measured by "regret", defined to be the difference between our loss, and the best possible loss incurred by choosing the best expert at each iteration. Amazingly, with some amount of mathematical analysis, we can prove that the performance of this algorithm yields a result that is almost as good as if we were to have picked the best expert since the very beginning!

Bagging

Now having seen the multiplicative weights update framework, let us explore what applications of this type are seen in machine learning. Indeed, it will turn out the

ensemble methods we introduce share some striking similarities with the algorithm above. First, we shall explore the method of bootstrap aggregation, or **bagging**.

Bias variance trade off shows that when applying a model to a data set, getting good bias (low bias) often leads to high variance, and vice versa. There are also techniques that perform very inconsistently depending on the data set or the features, like decision trees.
Bagging addresses this problem by applying the model to different sets of data set or features and then combining them by simply taking the average. This results in lower variance if different iterations are relatively uncorrelated.

Boosting

Boosting combines models (especially weak learners) together for better performance
Focus of the boosting:

- Find learners that can correctly classify data point that current model is wrong about
- Weighing models by current performance each time a model is added
- “Harder” points classified with more models

Sources:

https://cs170.org/assets/notes/notes-mw.pdf (1)
https://www.eecs189.org/static/notes/n26.pdf

I think you could try to if you already finish your shit -

1. Try creating a simple K means visualization
2. Try creating a simple polynomial visualiation