Lecture 14: Ensemble Methods

Construct a set of classifiers from the training data - this is analogous to our conglomerate classifier, where we combine information from multiple different classifiers.

Why does this work? Well, imagine we have 25 classifiers that are all independent. If each classifier has an error rate of 35%, then the probability that the ensemble classifier errors, assuming we take the majority rule, is 6%. That is, since we are taking the majority answer, we need at least 13 of the 25 classifiers to be wrong - the chances of that happening are only 6% in this case. This is a geometric distribution.

Clearly there is something to be gained here. How can we create these? There are two methods: bagging and boosting.

Bagging

Here, we sample with replacement, in multiple rounds. In each round, we build a classifier on that sample. The purpose of replacement is to ensure that each round has different data. In some sense, we are changing the distribution of the data, and that ensures that the classifiers will be independent.

Boosting

Here, we try to assign weights to each of the records, based on how well our classifiers can do on that record. The idea is to have an iterative procedure that adaptively changes the distribution of training data by focusing more on previously misclassified records.

In each round, we modify the weights associated with the data points in such a way as to encourage learning on records that are hard to classify. Records that are wrongly classified have their weights increased.

For example: 4 is hard to classify, and so each time we increase the weight of class 4 in the sampling procedure, to increase the chances of seeing a 4 in the classifier. The hope is that at some point we get a classifier that is really good at classifying everything.

Example: Adaboost: given base classifiers $C_1, C_2, \ldots, C_t$, we can define an importance measure of each classifier. Note that it's possible to expose the classifier to new data, and it doesn't improve. In this case, we can revert back and start over.

Professor highly recommends we trying one of these methods for the midterm. sklearn has packages for this.

Linear Regression

Here, we are trying to predict a continuous variable. We are no longer talking about specific classes - that is, the outcome doesn't fall into a specific bucket. Rather, the outcome is a continuous distribution.

Motivation: given a series of data points, we want to understand how y changes with respect to x. The goal is to find a function h that explains well how y varies as a function of x. In other words, a best fit or best trend in the data.

How can we do this?

Imagine we have a function $h(x)$, how can we evaluate whether or not h is good? Intuitively, we want to compare $h(x_i)$ for each data point x, and then compare how well it did to the corresponding y_i. More of less, if it's sufficiently close to each y_i, then we can reasonably conclude that h is a good fit to the data.

We need a distance function, so that we can find the sum of the distances between each $h(x_i)$ and y_i. That is:

$$L(h) = \sum_i d(h(x_i), y_i) \tag{1}$$

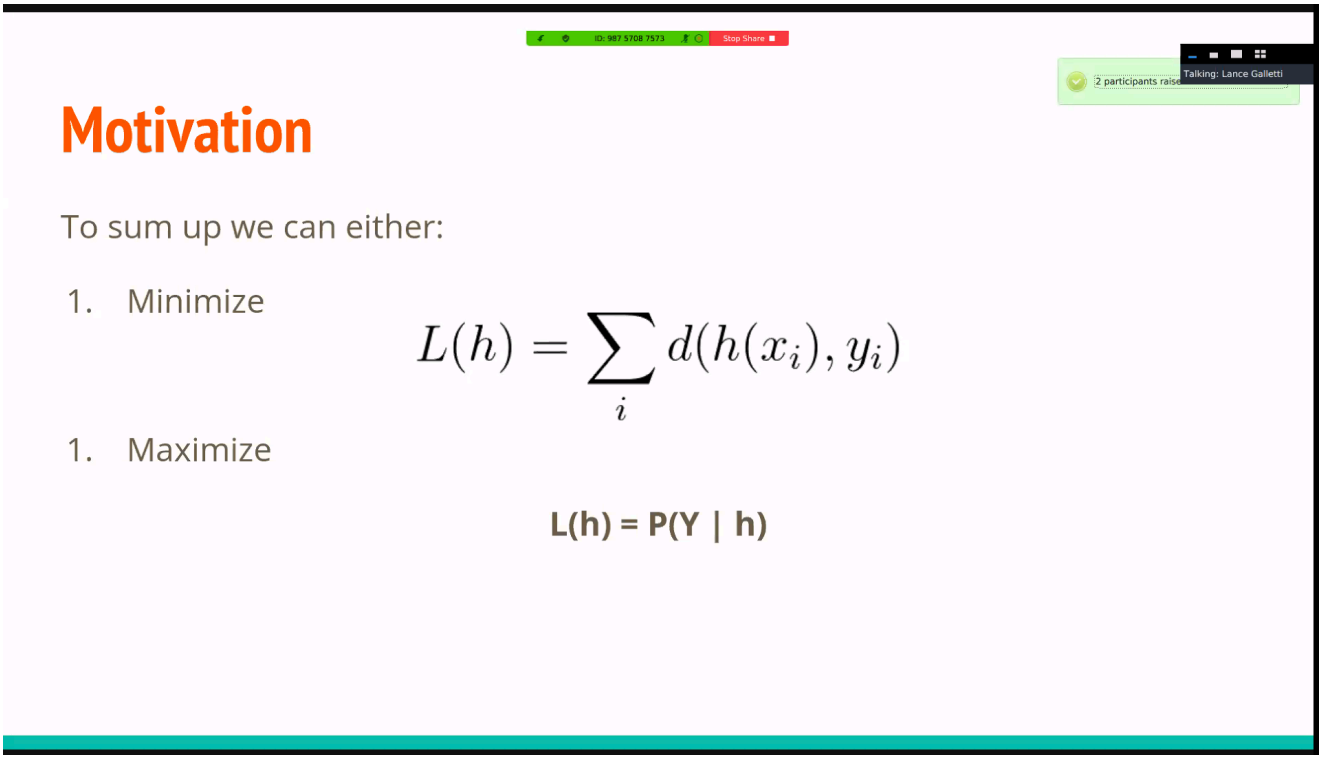
And, here, we want to find such a function where this value is minimized.

Do we want to find h such that it goes through the most samples? Well, no, because we are overfitting to the data, and it may not generalize easy. Further, the likelyhood is that if we find a function h that goes through all the data points, it will likely be extremely complex, which is not ideal. That makes it hard to explain, or possibly be non-continuous, or even just piece-wise continuous. None of these are desirable, and it's better to have a function h that is simpler, more explainable, and generalizes well.

Simple is always good.

How do we obtain these curves?

Let's reframe the problem a little bit. We can talk about this in terms of probability. Define $P(Y|h)$ as the probability of observing Y given that it was sampled from h. Then, the goal is to find h that maximizes the probability of having observed the data that we saw.



Do we have enough to get started here?

Not really - we haven't defined what h is - we need to constrain our problem, so that we have some reasonable amount of possible h function. As is, the problem statement is too vague - there are too many possible h functions.

How do we constrain the problem? We need to make some assumptions:

- We're going to say that h is a linear function in some parameter β
- We're going to assume that our data is generated by some linear function h plus some noise. That is, $\overrightarrow{y} = h_\beta(X) + \overrightarrow{\epsilon}$

Which function is linear in β?

- $h(x) = \beta_1 x$ - is this linear in β? yes
- $h(x) = \beta_0 + \beta_1 x$ - is this linear or not? yes, this one is linear as well.
- $h(x) = \beta_0 + \beta_1 x + \beta_2 x^2$ - is this linear or not? This is linear in β, because each term β is linear, even tho x is not.
- $h(x) = \beta_1 log(x) + \beta_2 x^2$ - this is also a linear function in β
- $h(x) = \beta_0 + \beta_1 x + \beta_2^2 x$ - finally, we have a function that is not linear in β

We now have an idea about the kind of functions we are looking for. To be clear:

- The relation between x (the independent variable) and y (the dependent variable) is linear in β
- The actual data is linear with respect to β, just with some noise added, such that each point doesn't lie precisely on h. That is, each point has $\epsilon_i \sim N(0, \sigma^2)$

Ok, so to actually solve this problem, we ned to find β. Since h is uniquely characterized by β, if we can find β, we have found our function h. That is - it doesn't really matter what we do with each x term, but we do have to define that up front - those are our features. Once we have those, then β uniquely defines h.

So how do we find β?

We can use the Euclidean distance between each point y and $h_\beta(x)$:

$$\arg\min_\beta || \overrightarrow{y} - h_\beta(X)||_2^2 \tag{2}$$

We can solve this by taking the derivative, setting it 0, and solving that function. We end up with:

$$\beta_{LS} = (X^T X)^{-1} X^T y \tag{3}$$

LS here stands for least squares, and the result here is a vector of betas.

We can also show a similar result from the maximization side.

An Unbiased Estimator

β_{LS} is an unbiased estimator of the true β. That is, the expectation of the least squares version is beta. Said simply, on average, we have the true parameter beta. We can show this by simply applying the rules of expecation.

Professor did a quick demo in class.