MATERIAL FOR LEARNING THEORY
MACHINE LEARNING E20

Mathias Ravn Tversted
Department of Computer Science
Aarhus University
Åbogade 34, 8200 Aarhus N
`Tversted@post.au.dk`

December 27, 2020

Contents

1 What is learning?

If we are working with data which is given by some unknown input distribution D on input feature vectors from R^d, then we want to find some hypothesis $h(x)$ where $h(x) \approx f(x)$. Or at least $h(x) = f(x)$ with *good probability* over the distribution D.

In order to do this, we will use two concepts:

- In-sample error E_{in} which is the error on training data. We can minimajs this.
- Out-sample error E_{out} which is how far the hypothesis is from the actual function.

We want to hope that $E_{in} \approx E_{out}$, because then minimajsing E_{in} does the same for E_{out}. This may not always be the case, so we want instead that with good $probability$ that $E_{in} \approx E_{out}$, or that we are *probably approximately correct* (PAC).

We will consider a general loss function which describes E_{in}.

$$E_{in}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x_i), f(x_i)) \tag{1}$$

Which makes

$$E_{out}(h) = \mathbb{E}_D[L(h(x), f(x))] \tag{2}$$

2 Hoeffding bounds

We would like to put a bound on the probability that our in-sample and out-sample error are close to each other, so that we can know if we are *probably approximately correct.*

The Hoeffing inequality gives us an upper bound on the probability that $E_{in} - E_{out} > \epsilon$.

$$Pr[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 n} \tag{3}$$

If we have lots of samples, then with high likelihood that $E_{in}(h) \approx E_{out}(h)$, and if our data is drawn *independently and identically distributed* (i.i.d), then it is likely.

If our hypothesis h is chosen *AFTER* our data set (which our optimal $h^*(x)$ is by our learning algorithm), then our hypothesis is not independent of the sample, and so the Hoeffing inequality fails. The Hoeffing inequality is a general bound regardless of the data set. Replacing the h hypothesis based on a dataset would break this generality.

But we can make a new bound that uses the size of the hypothesis-set to still make a meaningful bound. We know that the Hoeffing inequality for a single hypothesis from the hypothesis set must be smaller than the sum of all the Hoeffing inequalities for all hypothesis from the set if we consider the hypotheses independent of the sample.

$$Pr[|E_{in}(h^*) - E_{out}(h^*)| > \epsilon] \leq \sum_{n=1}^{M}(Pr[|E_{in}(h_1) - E_{out}(h_1)| > \epsilon]) \tag{4}$$

Which gives us that

$$Pr[|E_{in}(h^*) - E_{out}(h^*)| > \epsilon] \leq 2Me^{-2\epsilon^2 n} \tag{5}$$

Where M is the size of our hypothesis set. Observe that the larger our hypothesis set, the worse our bound.

We can also note that if $E_(h^*)$ is the smallest among our hypotheses, then any other hypothesis has at most 2ϵ better E_{out}.

This inequality can not handle infinite hypothesis sets.

3 Growth function and Hoeffing Bound

The growth function described as

$$m_H(n) = \max_{x_1,\cdots,x_n \in X} |(h(x_1), \cdots, h(x_n)) : h \in H| \tag{6}$$

Is a function of n points, which finds the maximum over a set of n (a sample) inputs, and finds the maximum of the number of *distinct ways* H can classify the n points.

Few examples:

- Positive rays (1d): $m_H(n) = 1 + n$
- Intervals (1d): $m_H(n) = 1 + n^2/2 + n/2$
- Convex sets (2d): $m_H(n) = 2^n$

Can we replace the M (the size of our hypothesis set) in the Hoeffing inequality with this new way of measuring number of *"distinct hypotheses"*? We can do, but with a few constants.

$$Pr[|E_{in}(h^*) - E_{out}(h^*)| > \epsilon] \leq 8m_H(2n)e^{-\epsilon^2 \frac{n}{8}} \tag{7}$$

4 VC-dimension

Instead of calculating $m_H(n)$, which may be difficult, we can attempt to produce an upper bound instead.

Shattering: A set of k samples $x_1, \cdots, x_k \in X$ is *shatered* by H if all 2^k dichotomies can be generated.

Break points: If *no* data set of k samples $x_1, \cdots, x_k \in X$ can be *shattered* by H, then k is a *break point* for H.

VC-dimension: Let k be the smallest *break point* for H, then $d_H = k - 1$. Or, d is the largest integer such that there is a data set of dize d that can be shattered by H. Intuitively, the VC-dimension measures the complexity of our hypothesis-set.

Examples of VC-dimensions:

- Positive rays: 1
- Intervals: 2
- Convex sets: ∞

The bounds on our growth function is

$$m_H(n) \leq n^d + 1 \tag{8}$$

This gives us a bound of

$$Pr[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 8((2n)^d + 1)e^{-\epsilon^2 \frac{n}{8}} \tag{9}$$

We can see that n^d grows polynomially, and $e^{-\epsilon^2 n/8}$ decreases exponentially, that means that with large n, E_{in} and E_{out} get closer to each other. If n and d increase, the R.H.S gets bigger, but only a larger n allows a smaller ϵ, which means a small d and large n allows for greater precision.

5 Testing

Instead of using the Hoeffding bound, which can be extremely far off in practice, we can set aside some data and measure $E_{\text{test}}(h^*)$, which is an approximation of $E_{out}(h^*)$.

If we choose a random subset of k items of the data before finding h^*, then both the k items and the $n - k$ items are i.i.d samples from D, which means that the larger k is, the more representative of E_{out} it will be.

We get a better bound of

$$Pr[|E_{test}(h^*) - E_{out}(h^*)| > \epsilon] < 2e^{-2\epsilon^2 k} \tag{10}$$

6 Over/Under-fitting

We want the in-sample and out-sample errors to be close. Solution: Choose simple model. Problem: Not very low in-sample error.

We want to minimajs in-sample error. Solution: Complex model. Problem: in-sample and out-sample errors not close. (We fit noise in data.)

What we get from this: The more data we have, the more complex a model we can use to some degree. E.g if we have 50 data points then a 50-degree polynomial to fit data is probably going to create overfitting, but could work if we had e.g. 10.000 data points.

7 Regularization

Regularization is any modification made to an algorithm with the goal of the modication being reducing E_{out} at the cost of increasing E_{in}. It makes the algorithm favour some hypotheses over others, and unpreferably solutions are only chosen if they result in large E_{in} gains.

We can use a complex hypothesis set without too much danger if we penalise complex hypotheses, which in practice is penalising the fitting of noise.

An example of this is *weight decay*. In linear models, one can choose a parameter $\lambda > 0$, such that $E_{in}(h(w)) = E_{in}(h(w)) + \lambda \|W\|^2$. By minimajsing E_{in} you are also penalising large weights, unless they result in large gains.

These are often useful in linear models.

8 Validation

In the same way that we made a test set we can create a validation set. This set will be used to make decisions on different hyperparameters.

There can be many combinations of hyper parameters for a loss-function. This can create a very big space of hyperparameters to test. Common solution is to pick random hyper parameters as long as you have time.

Cross-validation is training your model on e.g. the first 4/5 of the training set and then validate with the rest. Then doing this 5 times shifting which part is used for validation. In the end averaging the validations errors.

9 Bias-variance

We can do other things to set bounds on E_{out}, also with methods that take the algorithm used into account, which contrasts with the VC-bound which is *independent* of the algorithm used.

We want to bound $\mathbb{E}_D[E_{out}(h^D)]$ where h^D is the hypothesis produced by our algorithm on data set D.

9.1 The average hypothesis

We want to examine $\mathbb{E}_D[h^D(x)]$.

If we sample $D_1, \cdots, D_m$ and train h^{D_i}, then

$$\frac{1}{m}\sum_i h^{D_i} \tag{11}$$

Will tend to $\bar{h}(x) = \mathbb{E}_D[h^D(x)]$. Where $\hat{h}$ is the expected hypothesis. This measures the expected "square loss" of the average hypothesis against the actual values under our training data.

9.2 The bias-variance decomposition

Bias: $\mathbb{E}_x[(\hat{h}(x) - f(x))^2] = E_{out}(\bar{h})$. This is the performance of the average hypothesis.

Variance: $\mathbb{E}_x[\mathbb{E}_d[(h^D(x) - \hat{h}(x))^2]]$. This measures how much our hypotheses on training data.

The *bias-variance decomposition* is then

$$\mathbb{E}_D[E_{out}(h^D)] = \text{bias} + \text{variance} \tag{12}$$

In other words: The expected E_{out} on a hypothesis trained by our algorithm on the data set D is the average performance $+$ how much our models vary.

This roughly corresponsids to complex models being able to "achieve better results"' but they can also vary wildly. Opposite true for simple models. More data reduces variance.

10 Bagging

If we want to reduce variance, we can take the average of $i.i.d$ variables. If we can train $h_1, \cdots, h_m$ hypotheses, then we can take the average hypothesis.

10.1 The algorithm

Do m times:

- Sample $n' < n$ data points from D with replacement, call it D_i
- Train hypothesis h_i on D_i
- Output $h = (\frac{1}{m}(h_1 + \cdots + h_m))$ for regression
- Output $h = \text{sign}(h_1 + \cdots h_m)$ for classification

Reduces risk of overfitting noise on point, but reduces variance. Because we have less data for each hypothesis, our bias increases.

11 Random Forest

Random Forest is an algorithm for decision trees that attempts to reduce the correlation that you get when you subsample the data sets. The intuition is that if there is a pairwise correlation $\rho > 0$, and if $Y = (\frac{1}{m})(X_1 + \cdots X_m)$, then $Var(Y) = \rho\sigma^2 + (1-\rho)\sigma^2/m$.

We will train a number of trees where at each split it only considers a subset of features in order to reduce correlation between hypotheses. It is akin to something like Dropout for neural networks where, if a few neurons dominate, then randomly turning them off will enable other neurons to do the work.

Here randomly selecting a subset of features will mean that trees train less on the same things, and therefore it reduces correlation.

11.1 The algorithm

Do m times:

- Sample $n' < n$ data points from D with replacement, call it D_i
- Train hypothesis h_i on D_i
  - Randomly choose split p of d data features.
  - Only compute split on these p features.
- Output $h = (\frac{1}{m}(h_1 + \cdots + h_m))$ for regression
- Output $h = \text{sign}(h_1 + \cdots h_m)$ for classification