Variance-Bias Generalization extra notes

October 25, 2020

1 Definition of overfitting

- I have a "world", which I characterize by $\mathcal{Z}$ and $\mathcal{P}$. Data/label pairs (x_i, y_i) live in my world $\mathcal{Z}$, and are drawn i.i.d. from this world according to some distribution $\mathcal{P}$.
  - In estimation problems, $\mathcal{Z}$ is $\mathbb{R}^n$ and $\mathcal{P}$ may be a Gaussian distribution $\mathcal{N}(x^*, \sigma^2)$ where x^* is the true location of something (e.g. a star) and σ^2 is the measurement noise variance.
  - In classification, $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ and $\mathcal{P}$ may be a Bernoulli distribution (e.g. coin toss) with probability $\mathbf{Pr}(y_i = 1) = f(x_i)$, for some deterministic (but unknown) function f.
  - ...etc.
- I have a *loss* associated with a task I wish to complete.
  - In regression, that task may be to find f where $f(x_i) \approx y_i$, so my loss may be squared error
    $$\mathcal{L}(x, y) = (f(x) - y)^2.$$
  - In binary classification, that task is to find $f(x_i)$ where $\mathbf{sign}(f(x_i)) \approx y_i$, and can be achieved using a margin maximizing loss, e.g.
    $$\mathcal{L}(x, y) = f(yx).$$
    For example, in logistic regression, $f(yx) = -\log(\sigma(yx^T\theta))$.
  - In estimation, that task is to find an estimate of some truth. Given an estimate x_i of some ground truth $\bar{x}$, we may again use mean squared error as a loss function:
    $$\mathcal{L}(x) = (x - \bar{x})^2.$$
- The *expected loss* over the "entire world" is just the overall Bayes risk:
  $$\mathcal{E}^* = \mathbb{E}_{x,y\sim\mathcal{P}}[\mathcal{L}(x, y)] \qquad \text{(or simply } \mathcal{E}^* = \mathbb{E}_{x\sim\mathcal{P}}[\mathcal{L}(x)] \text{ for estimation)}$$
  This is what we want. (It is not the *only* thing we could go after. Recall, we used to have notions of minimax risk, and of course we can keep defining new ones. But in this lecture, we're going to just go after the Bayes risk.
- The *empirical loss* is the loss based on a snapshot of data. That is, suppose I observe a snapshot of data $(x_1, y_1), ..., (x_m, y_m)$. Then the empirical loss is simply
  $$\hat{\mathcal{E}} = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(x_i, y_i).$$
  From previous lectures, we know that, using only linearity of expectation, that $\hat{\mathcal{E}}$ is an unbiased estimator of $\mathcal{E}^*$.

- The difference between empirical and expected loss is called *generalization loss*

$$\textbf{generalization loss} = \mathcal{E}^* - \hat{\mathcal{E}}.$$

We can see from simple math that the expected loss can be written as a composition of empirical loss and generalization loss. This observation is useful, because it tells us that, when accomplishing a machine learning task, we must always provide balance between $\hat{\mathcal{E}}$ (which can also be viewed as *training loss*) and $\mathcal{E}^* - \hat{\mathcal{E}}$ (which is sometimes called *loss from overfitting*).

- More generally, we will say that *a model has overfitted* if $\hat{\mathcal{E}} < \mathcal{E}^*$.
- Questions for thought: What does an "overfitted model" look like, in the case of
  - polynomial regression?
  - decision trees?
  - K-nearest neighbors?

2 Variance and bias tradeoff

- Whether we are doing regression, classification, or estimation, we can somewhat sum up our goal as to finding $\hat{f}$ where, over our world $\mathcal{X}$ and probability distribution $\mathcal{P}$, $\hat{f}(x) \approx f(x)$ for some unknown function $f(x)$
- Given some batch of training data $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$, and some training scheme, we will end up with some guessed model $\hat{f}_{\mathcal{D}}(x)$.
- Then using simple math tricks, we can write the squared error between the "learned" model and the "true" model as

$$(f(x) - \hat{f}_D(x))^2$$

which is a function of x, but is also random over the draws $\mathcal{D}$. Taking the expectation over D, we arrive at

$$\begin{aligned}
\mathbb{E}_D[(f(x) - \hat{f}_D(x))^2] &= \mathbb{E}_D[(f(x) - \mathbb{E}_D[\hat{f}_D(x)] + \mathbb{E}_D[\hat{f}_D(x)] - \hat{f}_D(x))^2] \\
&= (f(x) - \mathbb{E}_D[\hat{f}_D(x)])^2 - 2\mathbb{E}_D[(f(x) - \mathbb{E}_D[\hat{f}_D(x)])(\mathbb{E}_D[\hat{f}_D(x)] - \hat{f}_D(x))] \\
&\quad + \mathbb{E}_D[(\mathbb{E}_D[\hat{f}_D(x)] - \hat{f}_D(x))^2] \\
&= (f(x) - \mathbb{E}_D[\hat{f}_D(x)])^2 - 2(f(x) - \mathbb{E}_D[\hat{f}_D(x)]) \underbrace{\mathbb{E}_D[(\mathbb{E}_D[\hat{f}_D(x)] - \hat{f}_D(x))]}_{=0} \\
&\quad + \mathbb{E}_D[(\mathbb{E}_D[\hat{f}_D(x)] - \hat{f}_D(x))^2] \\
&= \underbrace{(f(x) - \mathbb{E}_D[\hat{f}_D(x)])^2}_{\text{bias}^2 \text{ of model } \hat{f}_D} + \textbf{Var}_D(\hat{f}_D(x))
\end{aligned}$$

- Thus, we can see that the mismatch between a "guess" and a "truth", as determined by the random variable D, can be decomposed into a squared bias and variance term.
- Some people call this a tradeoff, because often the bias is

3 Ways we can control the generalization error

Assume we have now overfit our model, e.g. $\hat{\mathcal{E}} < \mathcal{E}^*$. Now we will consider some strategies in which we can rein in the overfitting a bit.

3.1 More data

- Suppose I have m observations. Then my generalization loss is

$$\mathcal{E}^* - \hat{\mathcal{E}}_m \quad = \quad \mathbb{E}[\mathcal{L}(x,y)] - \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(x_i,y_i).$$

  Note that since we assume $\mathcal{E}^* > \hat{\mathcal{E}}_m$, this term is just the square root of the variance of the estimator $\hat{\mathcal{E}}_m$!

- Now suppose that I have $10m$ observations, leading to an empirical loss $\hat{\mathcal{E}}_{10m}$. Then, by linearity of variance, we know that

$$\mathbf{Var}(\hat{\mathcal{E}}_{10m}) = \frac{1}{m}\mathbf{Var}(\hat{\mathcal{E}}_m).$$

  That is to say, the generalization error for $10m$ observations is $1/\sqrt{10}$ times that of m observations!

- This is by far the best way to combat overfitting. However, it may not always be possible in practice.

3.2 Weakening the model

- The most popular example of this strategy is to limit the depth of decision trees, e.g. returning the tree as soon as the depth has hit some limit. An extreme version of this is depth = 2, leading to decision *stumps*.

- Another popular example of this is to limit the degree of a polynomial, or the complexity of a basis function, in a generalized linear model. For example, in a polynomial fit, we only allow the degree of the polynomial to be, say, at most 2 or 3.

- Along similar lines, in deep neural networks, you can limit the complexity by limiting the depth or width of the networks being trained.

- Weakening can also be done in the *training* of the model. For example, in many continuously-trained models, it is not uncommon to use any of these strategies:
  - early stopping (you only run a set, small number of gradient descent steps)
  - "dropout", or other gradient modifications that look like adding noise,
  - avoid using acceleration methods [1]

- Does weakening always help reduce expected loss? It depends. It certainly reduces error from overfitting, by increasing the training loss. But remember that the goal is not entirely to reduce the gap $\mathcal{E}^* - \hat{\mathcal{E}}$, but rather the expected loss $\mathcal{E}^*$ itself! So, weakening the model must be done with care, to ensure that the "best weak model" is chosen. (See cross validation.)

3.3 Regularization

- Often, you will see the training of machine learning models written as

$$\underset{\theta}{\text{minimize}} \quad \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(x_i,y_i) + \rho\mathcal{R}(\theta)$$

  where $\mathcal{R}(\theta)$ is a *regularization term*.

[1] Somewhat controversial, but there is evidence to show this can help.

- Actually, we have already seen this term when we did linear regression. In particular, we looked at the special case of

$$\underset{\theta}{\text{minimize}} \quad \|X\theta - y\|_2^2 + \rho\|\theta\|_2^2$$

and here, $\mathcal{R}(\theta) = \|\theta\|_2^2$ had the interpretation of

  - modeling uncertainty in θ
  - providing "stability" to the θ estimate
  - improving the conditioning of the overall objective function.

- If you tried one of the past challenge problems, you also know that the regularization penalty can also take the form $\mathcal{R}(\theta) = \|\theta\|_1$, to enforce sparsity in θ.

- The other popular form of regularization in machine learning is negative entropy:

$$\mathcal{R}(\theta) = \sum_i \theta_i \log(\theta_i)$$

which can be used to encourage "diversity" in θ.

- Regularization can be seen as a technique for weakening the model, as it restricts the expressibility of the model. We also see that, in many cases, the regularization term adds bias to the resulting estimator. There are several advantages, however, of using this technique:

  - Usually it's easy to improve, and adds very little overhead to the training procedure. In some cases, it may even improve the problem conditioning, and reduce the training runtime.
  - Picking the hyperparameter ρ can be done in a grid search or cross validation way, both of which is parallelizable. So, although it does add computational overhead, the "way to do it" are simple and not that controversial.
  - In linear regression, as we saw in a past homework, adding 2-norm regularization reduces explosions of the solution in non-gradient-reachable directions. (See homework 2, problem 1.d.iii.)

- In the homework, you will explore regularization in estimation, where the solution can be computed and analyzed directly.

3.4 Cross Validation

Hyperparameters

- When we train models using gradient descent, we often distinguish between *parameters* and *hyperparameters*.

  The *parameters* θ are the parts of our model that we update throughout the gradient descent procedure. The *hyperparameters* are anything else that affects the model, but are usually fixed during training.

  Examples of hyperparameters include

  - depth of decision tree
  - width of decision tree
  - stopping criterion for gradient descent
  - degree of polynomial in polyfit
  - regularization parameter ρ
  - size of minibatching in stochastic methods
  - use of acceleration methods
  - ...

Model selection

- As an example, let's consider the problem of regularized linear regression:

$$\underset{\theta}{\text{minimize}} \quad \frac{1}{2m}\sum_{i=1}^{m}(x_i^T\theta - y_i)^2 + \frac{\rho}{2}\|\theta\|_2^2$$

- We pick out some optimal value θ^*, and return our new model

$$y = x^T\theta^*.$$

- But, $\theta^* = \theta^*(\rho)$ depends on ρ! So, each choice of ρ actually leads to a different model.
- The problem of *model selection* is to determine which value of ρ leads to the right model.
- Note the same setup can be used to determine any of the suggested hyperparameters

Grid search

- Suppose I need to pick p hyperparameters, $h_1, ..., h_p$, to help determine my final model.
- One straightforward accepted practice is to sample the model's behavior over a grid, usually on a logarithmic scale. Pseudocode for this may look like

```
for h1 in [1e-5,1e-3,1e-1,1e1,1e3]:
    for h2 in [1e-5,1e-3,1e-1,1e1,1e3]:
        for h3 in [1e-5,1e-3,1e-1,1e1,1e3]:
            .
            .
            evaluate model with hyperparams h1,h2,h3,...

return best performing model.
```

- What is the computational complexity for sampling through p hyperparameters, if we sample each hyperparameter's range with q samples?
- Usually, we
  - have access to a massive computational resource and just run all of these guys in parallel
  - pick some hyperparameters based on "sensible choices" and just hope the model isn't that fragile
  - pick very coarse discretizations to reduce the search space.

Validation set

- The *validation set* is the portion of the training set set aside for validating the hyperparameters.
- It is *very important* to never use the test set for hyperparameter searching, as it leads to overfitting of hyperparameters.
- Often, the validation set is chosen to be around 15% of the training set, but if there is not much training data, that number may be adjusted.
- K-fold leave one out cross validation (LOOCV)
  - Partition your training set into K equal portions
  - For $k = 1, ..., K$, set aside the kth portion as the validation set, and train on the remaining parts.
  - Pick the model with the best average validation score.

  If your validation is very small, this is much more stable.