Underfitting and Overfitting in Machine Learning

Jeff Bilmes

September 26, 2020

1 Notation and Set Up

Let the sample distribution be $p(x, y)$. We have a **training data set** of n_{tr} training samples $\mathcal{D}_{\text{tr}} = \{(x_1, y_1), (x_2, y_2), \dots, (x_{n_{\text{tr}}}, y_{n_{\text{tr}}})\}$ drawn from this distribution, meaning $(x_i, y_i) \sim p(x, y)$ for all $1 \le i \le n_{tr}$.

We train a classifier $h \in \mathcal{H}$ on this training set using some form of empirical risk minimization (ERM).

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n_{\text{tr}}} \mathcal{L}(y_i, h(x_i)) + \lambda \Omega(h) \tag{1}$$

where $\mathcal{L}()$ is a loss, or error, function and $\Omega()$ is a regularizer (that prefers simpler models in some way).

For the purposes of this discussion, lets change notation a little bit. Firstly, we use set-theoretic notation so that we are explicit about what data set we are averaging over as it will simplify things below. I.e., we say

$$\sum_{i=1}^{n_{\text{tr}}} \mathcal{L}(y_i, h(x_i)) = \sum_{(x,y) \in \mathcal{D}_{\text{tr}}} \mathcal{L}(y, h(x)) \tag{2}$$

meaning that we are summing over training samples pairs (x, y) in the training set $\mathcal{D}_{\text{tr}}$.

Secondly, we define an accuracy function $\mathcal{A}$ that is antitonely related to $\mathcal{L}$ (e.g., $\mathcal{A} = \kappa - \mathcal{L}$ for a constant κ). Hence, instead of Equation (1), we can just as well do the following optimization:

$$\hat{h} \in \operatorname*{argmax}_{h \in \mathcal{H}} \sum_{(x,y) \in \mathcal{D}_{\text{tr}}} \mathcal{A}(y_i, h(x_i)) - \lambda \Omega(h) \tag{3}$$

which maximizes accuracy on the training set while preferring, to the extent that λ is large, a simple model. Learning therefore involves doing either Equation (1) or Equation (3) which are equivalent.

Once we have $\hat{h}$, we wish to evaluate it. In an ideal world, we will evaluate $\hat{h}$ over the entire distribution $p(x, y)$ and (should we wish to compute accuracy) this would mean computing

$$\text{accuracy}(\hat{h}) = E_{p(x,y)}[A(Y, \hat{h}(X))] = \int_{x,y} p(x, y) \mathcal{A}(y, \hat{h}(x)) \tag{4}$$

Unfortunately, we do not have access to $p(x, y)$, and even if we did, the computation in Equation (4) is prohibitive at best and hopelessly impossible in general. Thus, we instead create or obtain a separate **validation data set** (or sometimes called a **development set** or a **holdout set**) of n_{va} samples $\mathcal{D}_{\text{va}} = \{(x_1, y_1), (x_2, y_2), \dots, (x_{n_{\text{va}}}, y_{n_{\text{va}}})\}$ drawn from the same distribution as the training set, meaning that $(x_j, y_j) \sim p(x, y)$ for all $1 \le j \le n_{va}$.

With this set, we can then approximate the accuracy as follows:

$$\text{accuracy}_{\mathcal{D}_{\text{va}}}(\hat{h}) = \frac{1}{|\mathcal{D}_{\text{va}}|} \sum_{(x,y) \in \mathcal{D}_{\text{va}}} \mathcal{A}(y_j, \hat{h}(x_j)) \approx \text{accuracy}(\hat{h}) \tag{5}$$

where $n_{\text{va}} = |\mathcal{D}_{\text{va}}|$. As $n_{\text{va}} \to \infty$, $\text{accuracy}_{\mathcal{D}_{\text{va}}}(\hat{h}) \to \text{accuracy}(\hat{h})$ by the law of large numbers.

With this notation, we can denote the computation of the accuracy $\text{accuracy}_{\mathcal{D}}(\hat{h})$ on any data set $\mathcal{D}$, as follows:

$$\text{accuracy}_{\mathcal{D}}(\hat{h}) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \mathcal{A}(y_j, \hat{h}(x_j)) \tag{6}$$

Thus, we can also denote the computation of the accuracy of $\hat{h}$ on the training data set as $\text{accuracy}_{\mathcal{D}_{\text{tr}}}(\hat{h})$ and/or on the validation set as $\text{accuracy}_{\mathcal{D}_{\text{va}}}(\hat{h})$.

2 Overfitting and Underfitting

We say that a learnt hypothesis $\hat{h}$ **overfits** the training samples $\mathcal{D}_{\text{tr}}$ if there exists some other hypothesis, lets call it h', that fits the training examples $\mathcal{D}_{\text{tr}}$ worse than $\hat{h}$, but performs better over the entire distribution. In other words, a precise definition of overfitting is the following:

DEFINITION 1 (overfitting): *We say that $\hat{h} \in \mathcal{H}$ **overfits** the training data $\mathcal{D}_{tr}$ if there exists $h' \in \mathcal{H}$ such that*

$$\text{accuracy}_{\mathcal{D}_{\text{tr}}}(\hat{h}) > \text{accuracy}_{\mathcal{D}_{\text{tr}}}(h') \quad \text{and} \quad \text{accuracy}(\hat{h}) < \text{accuracy}(h'). \tag{7}$$

Since we can't compute accuracy$(\hat{h})$ or accuracy(h') as mentioned above, a practical definition of overfitting changes this to:

$$\text{accuracy}_{\mathcal{D}_{\text{tr}}}(\hat{h}) > \text{accuracy}_{\mathcal{D}_{\text{tr}}}(h') \quad \text{and} \quad \text{accuracy}_{\mathcal{D}_{\text{va}}}(\hat{h}) < \text{accuracy}_{\mathcal{D}_{\text{va}}}(h'). \tag{8}$$

This definition is consistent with Tom Mitchell's definition of overfitting, which states the same thing but it is less explicit, and uses error rather than accuracy.

DEFINITION 2 (overfitting (Mitchell, pg.67, [1])): *Given a hypothesis space $\mathcal{H}$, a hypothesis $h \in \mathcal{H}$ is said to overfit the training data if there exists some alternative hypothesis $h' \in \mathcal{H}$, such that h has smaller error than h' over the training examples, but h' has a smaller overall error than h over the entire distribution (or data set) of instances.*

How can we tell if a given hypothesis h overfits on the training data $\mathcal{D}_{\text{tr}}$? It really depends on two things: (1) the complexity of any given hypothesis h, and (2) the number of training samples n_{tr}.

2.1 Over/Under-fitting and Model Complexity

Let complexity(h) be the complexity of hypothesis h. How do we measure the complexity of h? There are many possible ways. Simple ways include counting the number of free parameters in h (useful and practical if h is a neural network) or the number of nodes (if h is a decision tree). Alternatively, we could measure it in the amount of training computational that takes place (e.g., the number of epochs through the data) since, after all, any model that can be achieved with a certain amount of computation can certainly also be achieved with more computation, but the opposite is not necessarily true – i.e., the more computation allowed during training, the more potentially complex and expressive the resulting model and, hence, the more opportunity there is to better fit the training data (this is the principle behind **early stopping**, where we stop training once the validation set accuracy starts decreasing). Alternatively, there are more mathematical strategies including a norm of the parameters, the VC dimension, Radamacher complexity, or minimum description length (MDL). Different ways of measuring complexity are useful for different purposes. In fact, $\Omega(h)$ above used to help select h can measure complexity in the same way, and as can be seen, the goal is to find an accurate classifier that is also, depending on the magnitude of λ, simple.

Regardless of how complexity is measured, in general, as the complexity of a given h increases, it's **expressiveness**, **capability** , **ability**, or **capacity** (all terms that are used and that are roughly synonymous with each other), also increases. For example, if $\Omega(h)$ measures the complexity of h, then given two models

h, h' with $\Omega(h) < \Omega(h')$, then h' has more expressiveness or ability. More precisely, given two thresholds t, t' with $t < t'$, then the family $\{h : \Omega(h) < t'\}$ has more expressiveness than does the family $\{h : \Omega(h) < t\}$.

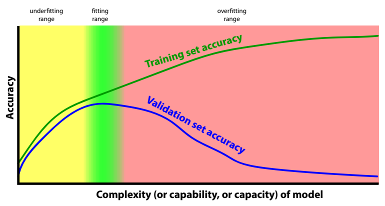
Therefore, as the complexity of h increases, h's tendency to well fit the training data also increases. If we are not careful, h's increasing complexity relative to the training set size also increases h's propensity to overfit on the training data, since it has the ability to (and like will) learn the detailed idiosyncrasies of the specific sample $\mathcal{D}_{\text{tr}}$ that are not common in general over the entire distribution $p(x, y)$, and are also not shared in other sampled data sets such as the validation set $\mathcal{D}_{\text{va}}$.

As complexity(h) gets smaller, the chance of overfitting decreases – on the other hand, the possibility of **underfitting** increases. Underfitting is the opposite of overfitting. A learnt hypothesis $\hat{h}$ underfits the training samples $\mathcal{D}_{\text{tr}}$ if there exists some other hypothesis, lets call it h'', that fits the training data better than $\hat{h}$ and also performs better over the entire distribution. In other words:

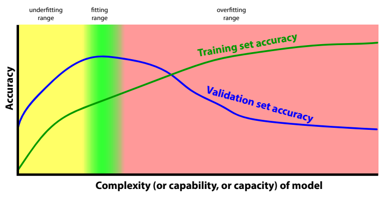
Definition 3 (underfitting): *We say that $\hat{h} \in \mathcal{H}$ **underfits** the training data $\mathcal{D}_{tr}$ if there exists $h'' \in \mathcal{H}$ such that*

$$\text{accuracy}_{\mathcal{D}_{\text{tr}}}(\hat{h}) < \text{accuracy}_{\mathcal{D}_{\text{tr}}}(h'') \quad \text{and} \quad \text{accuracy}(\hat{h}) < \text{accuracy}(h''). \tag{9}$$

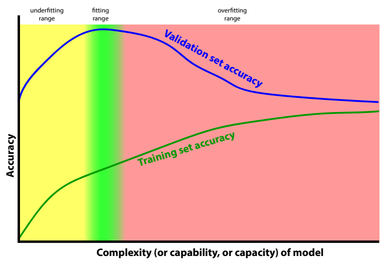
We can show overfitting and underfitting by showing how a trained model's accuracy changes as a function of the trained model's complexity, assuming n_{tr} is fixed, as is done in Figure 1 which shows what are sometimes called **validation curves**. Any h in the red region (on the right) is overfitting since there is some h' (in the green) that has lower training set accuracy but higher validation set accuracy. Any h in the yellow (on the left) is underfitting since there is some h' (in the green) that has higher training set accuracy and also higher validation set accuracy.



(a) The highly typical (optimistically biased) case is when the training set accuracy is everywhere higher than the validation set accuracy.



(b) It is possible, but somewhat unlikely, that the validation set accuracy crosses the validation set accuracy. The overfitting/underfitting ranges are still the same.



(c) It is possible, but very unlikely, that the validation set accuracy is higher than the training set accuracy, as shown in this plot. But the overfitting/underfitting ranges are still the same.

Figure 1: Validation curves. Overfitting and underfitting shown as a function of model complexity for a fixed training set size n_{tr}. Any h in the red region overfits the training set. Any h in the yellow region underfits the training set. Any h in the green (middle) region properly fits the training set — the best model choice would be one that has a complexity in the middle of the green region. The regions are based all on the accuracy $\text{accuracy}_{\mathcal{D}_{\text{va}}}(h)$ computed on an validation data set $\mathcal{D}_{\text{va}}$, but the same principle would be true if it were possible to measure accuracy it on the entire distribution accuracy(h). From a bias/variance perspective, models with high bias are to the left of the yellow region, while models with high variance are those on the right of the red region.

There are three figures in Figure 1.

The left figure (Figure 1a) shows the case where the training set accuracy is everywhere higher than the validation set accuracy. This is by far the most common case since the model has been trained directly on the training set in Equation (1) (or Equation (3)) but not directly on the validation set, so it is not surprising that the accuracy would be higher on the training set. This typical case is called the "optimistically biased" case since the training set performance, while biased, is biased optimistically (the accuracy is higher on the training set than the real accuracy on the distribution, or in other words the error on the training set will be an underestimate of the true error, something that is approximated by the validation error). In this plot, we can see that the model that underfits (i.e., the **high-bias case**, as we will eventually see) shows that the model does about the same on the training and on the validation data sets. For the model that overfits (i.e., the **high-variance** case, as we will see) shows that the model does much much better on the training set than on the validation set, as it has overfit to the idiosyncrasies of training set by quite a bit without capturing the general properties of the overall distribution.

On the other hand, it is in theory possible to have a model whose accuracy is better on the validation set than on the training set as shown in Figures 1b and 1c. On the right, Figure 1c, is the extremely unlikely case where we see that the validation set accuracy is everywhere higher than the training set accuracy. There are several reasons why this might happen:

1. This phenomenon can happen if the validation set is very small and/or is quite redundant, from an information theoretic perspective, with the training set. One possible way this could occur is if the validation set is a subset of the training set consisting only of very easy samples — this would be a very poor validation set, though, since the goal of the validation set is to act as a surrogate to $p(x, y)$.

   Another possible way this could happen is if the overall available data was split into two sets, training and validation, based on some order having to do which chronology, file ID, or spatial position. For example, it might be that the training set comes from samples taken weekdays (Monday through Friday) while the validation set comes from samples taken on weekends (Saturday, Sunday). It could be that the weekend samples are for some reason easier than the weekday samples. In general, we want to avoid such bias and, uniformly at random, partition any given data into a training and validation set.

2. Another way where this could happen is if the training and validation sets are not uniformly at random independent samples from the distribution $p(x, y)$. I.e., if the validation set was not sampled statistically independently from the training set, and if the validation set was instead sampled from a conditional distribution of some sort $p(x, y|\mathcal{D}_{\text{tr}})$, then this could cause the problem. For example, $p(x, y|\mathcal{D}_{\text{tr}})$ could have higher probability on regions in x, y-space that are well-represented in the training set. As a very contrived, but also specific example of this, suppose $p(x, y|\mathcal{D}_{\text{tr}}) \propto \sum_{(x,y)\in\mathcal{D}_{\text{tr}}} \text{Kernel}(x, y)$ where $\text{Kernel}(x, y)$ is a kernel density around point x, y. This conditional distribution would put concentrated mass around each training sample, and so any sample from $p(x, y|\mathcal{D}_{\text{tr}})$ would have a high probability of being highly similar to the training set.

3. Another case where this might occur would be if the training set consists of either outliers or label errors that do not occur in the validation set. The outliers or errors might live in heavily populated regions of the training data so that they do not effect the resulting trained model too much, but the overall training accuracy could be reduced — how much the accuracy reduced would correspond to the degree to which there are outliers or errors. If those types of errors do not occur in the validation data, then the validation accuracy could be higher.

4. Yet another situation that can arise and that might cause this phenomenon is if certain randomized regularization procedures are performed during training that are not performed during validation. This is particularly true when training deep neural networks (DNNs) and other non-linear models. For example, if noise being added to the samples x during training in order to keep the model from overfitting, but the noise is not being added at validation time, this can make training accuracy worse.

   Data augmentation (when random slightly modified or distorted samples are augmented to the training data set, but not the validation set) could have a similar effect. For example, if a training data set is augmented with a large number of additional and difficult samples, the accuracy on the training data might be worse than on the validation set, and this is particularly likely to happen early in training. Hence, if you find that augmented training set training accuracy is worse than test accuracy after you are finished training, it might be the case that more training epochs are warranted.

   Alternatively, if **dropout** (a random perturbation of a neural network units) is being applied at training time, this means random changes in the model are happening at training time and are working to prevent overfitting during training. If this is not being applied at validation time, then validation accuracy could be higher than training accuracy. Another factor to consider is **batch-normalization** a technique that might be different during training and validation. All of these methods can also lead to such training/validation performance mismatches.

In all of the above, this happens when it is somehow not the case that the training and validation sets are chosen independently and uniformly at random from $p(x, y)$, and so the properties of the sets are different.

Getting back to over- and under-fitting, it is important to realize that, irregardless of these three cases shown in Figure 1, the overfitting, underfitting, and fitting ranges are the same (although the extent to which overfitting really occurs would depend on how good the validation sample set $\mathcal{D}_{\text{va}}$ is at representing $p(x, y)$). Notably, Definition 1 and Definition 3 do not say anything about the relative accuracy of a particular model on the training vs. the validation set (i.e., the definitions do not require any relationship, for given model h, between the two quantities $\text{accuracy}_{\mathcal{D}_{\text{va}}}(h)$ and $\text{accuracy}_{\mathcal{D}_{\text{tr}}}(h)$).

It is also important to realize that because of the figure, looking only at two values of training and validation accuracy (i.e., looking at only one thin vertical slice located at one horizontal coordinate on one of the figures in Figure 1) is not enough to determine if you have overfit, underfit, or fit. That is, if you're given just two values, training accuracy and validation accuracy and nothing more, this is insufficient to tell if the model has overfit or underfit. To get a strong sense of if the model has overfit, underfit, or fit, one must look at the entire validation curves, and also the learning curves described in the next section.

2.2 Over/Under-fitting, Number Of Training samples, and Learning Curves

In the above, we stated that as the complexity $\text{complexity}(h)$ of h increases, h's tendency to overfit the training data also increases. As $\text{complexity}(h)$ gets smaller, the chance of overfitting decreases and the possibility of underfitting increases.

There is another critical parameter that helps to determine of overfitting or underfitting occurs, and that is the the training data size n_{tr}. In general, as the training data size n_{tr} decreases, h's tendency overfit increases; while as n_{tr} gets larger, the chance of overfitting decreases (and the possibility of underfitting increases).

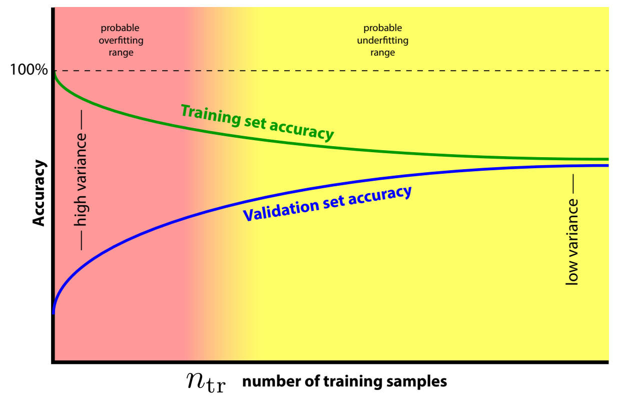
A given model with a fixed complexity (i.e., $\Omega(h) = \text{const.}$) will tend to overfit a small training data set, and this means that the training accuracy (on the small training set) will be high while the validation accuracy (on the validation data set) will be low. On the other hand, a given model with the same fixed complexity (i.e., again $\Omega(h) = \text{const.}$ for the same constant) will tend to underfit a large training data set. This means that the both the training accuracy and validation accuracy will be low (in general), although the validation accuracy will be lower (in the optimistically biased case) than the training accuracy.

A good way to empirically study how the number of training samples affects the tendency of a machine learning model to overfit or underfit the data is via **learning curves**. A learning curve is a plot that shows both training set accuracy and validation data set accuracy of a given model with fixed complexity, as a function of n_{tr}, the number of samples used to train the model.

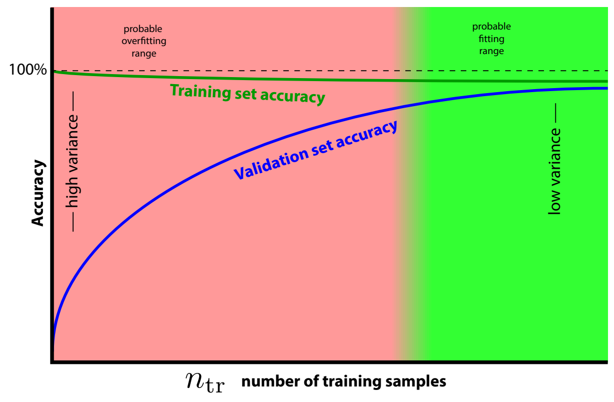
It is important to realize that in a learning curve, the training set accuracy is the accuracy on the given training set. I.e., at the very left of the plot, when n_{tr} is very small (or even $n_{\text{tr}} = 1$), almost any model will be able to learn to memorize this small training set. This means that the training accuracy (on the small training set) will be high. For example, even a linear model can perfectly fit either one or two training samples perfectly well, but as the number of points increases, it becomes extremely unlikely that the linear model will continue to well fit the data and so the training set accuracy decreases as more samples are used to train on. On the other hand, with such a small training set, very little about the actual distribution $p(x, y)$ can possibly be learned, and thus the validation accuracy (on the validation data set) will be low.

What happens when n_{tr} gets large depends on the complexity of the model $\Omega(h)$, and we show two cases in Figure 2.

Figure 2a shows a learning curve with a low complexity (i.e., $\Omega(h)$ small) model. While the performance of the model starts out high on the training set with a small number of samples, the training performance quickly gets worse with more training data owing to the models inflexibility and inexpressivity — there is just not much that the model can do, and hence as we get more training data, it is unable to even learn the patterns in the training data. The validation set accuracy starts out very low, since with little training data, it is unlikely to be a good representation of the validation set. Thus, we have high training accuracy and low validation accuracy, so this is likely a form of overfitting (i.e., there is probably a model that would do worse



(a) Learning curve with a low complexity (i.e., $\Omega(h)$ small) model.



(b) Learning curve with a high complexity (i.e., $\Omega(h)$ large) model.

Figure 2: Learning curves

on the training data, training just on that much training data, but would do better on the validation set as per Definition 1). As the training data set increases, then the training data set accuracy decreases, while the validation date set increases, until they meet at a point of mediocrity. This is the optimistically biased case, so the training set accuracy is always higher than the validation accuracy, but the model is never able to learn well and get high accuracy (i.e., close to 100%) on the validation data even with a lot of training data. Therefore, rest of the region is underfitting (yellow) since the model is never able to fit the data well (relating to Definition 1, we can see that this is underfitting since there is another model, shown in Figure 2b that does better on the both that much training data and also on the validation data). Thus, in this case the model moves from (probable) overfitting to underfitting without ever being able to properly represent (and fit) the information in the data since the model is to simple (i.e., biased).

Figure 2b shows a learning curve with a high complexity (i.e., $\Omega(h)$ large) model. Here also the training set accuracy starts out high with a small amount of training data (since a very small amount of training data is always easy to learn). This is probably overfitting since there is likely a simpler and less capable model that, with that much training data, would do worse on the training data but better on the validation data. As the amount of training data increases, the training accuracy stays high since the model is flexible and is able to learn the patterns in any size training data set well. At some point, the training data set size becomes large enough that it is a good representation of $p(x, y)$ and since the model is expressive it represents the true patterns in $p(x, y)$ and so the validation set accuracy also increases. Eventually, with a large enough training data set, the validation set accuracy gets very high and approaches the training set accuracy. In theory, with an expressive enough model, and an infinite amount of training data, the validation and training set accuracies would converge to each other at the best accuracy possible (which would correspond to one minus Bayes error). On the other hand, if eventually with enough training data, and if the task needing to be learnt is itself extremely complex, then even the model's high complexity and inherent ability will eventually run out, and we will reach again a region of underfitting (which is not shown in the figure).

Note that learning curves and model complexity curves are really just two slices of the same three-dimensional plot. Note that the above also refers to **bias** and **variance** and the **bias/variance tradeoff**, and this is something we will discuss very precisely in the very near future.

References

[1] Tom Mitchell, *Machine Learning*, McGraw-Hill Science/Engineering/Math; 1997