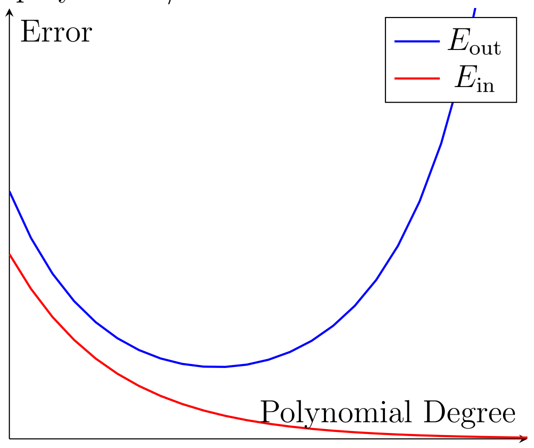
Model Selection

Validation

Last time we learned about overfitting. We did the exercise to show that we shouldn't use sixth-order polynomial to fit something that looks like parabola we will get a really low in sample error but we will get a high out of sample error. Last time we learn how to use regularization to alleviate this problem.

We are going to do something more general this time. But before we do that let us draw a graph of E_{in} and E_{in} as we increase the degree of the polynomial/features:



Let us understand this graph. First, at low polynomial degree you fit will be terrible giving you high E_{in} and you will do equally bad for E_{out}. This is like trying to fit a straight line through a parabola it wouldn't fit well in the training sample but it will also do badly out of sample. This is called underfitting.

On the other end of the plot, as you model gets complicated your E_{in} will be low since you have more than enough parameter to fit anything you want in side the training sample. However, you can't expect this to do too well out of sample since you are using a too complicated model. This end is called overfitting.

In between, there has to be somewhere that is just right. The question for this week is how to we pick the just-right degree of polynomial. Or more generally, how do we pick the right amount of complexity for our model. At one end, the simpler model will give you bad E_{in} and bad E_{out}. The other end will give you good E_{in} but still bad E_{out}.

We want to find this "Just Right" complexity. By now you should realize that what we really after is not E_{in} even though we optimize it to get the hypothesis. What we are really after is E_{out}: the out of sample error. We want our hypothesis to do best out of sample: we want the best E_{out}.

For example, let us consider a linear regression problem. We don't know the true complexity of the model so we are trying to pick the best hypothesis from the following hypothesis: (This is just the simplest example. Make sure you understand it enough to generalize it higher dimension)

$$h_1(x) = \vec{w} \cdot [1, x] \tag{1}$$
$$h_2(x) = \vec{w} \cdot [1, x, x^2] \tag{2}$$
$$h_3(x) = \vec{w} \cdot [1, x, x^2, x^3] \tag{3}$$
$$\vdots$$
$$h_n(x) = \vec{w} \cdot [1, x, x^2, x^3, \ldots, x^n] \tag{4}$$

Picking these hypothesis base on E_{in} would be a terrible idea since we will just ended up picking the over fitted one. Instead, we want to select among the above hypothesis the one that would give us the lowest E_{out}. But, the trouble is that we never know E_{out} until we use it in real application. All we have at hand is just the training sample.

We could fix this problem by splitting the training sample we have into two sets(we will make more split later on)

1. Training set.
2. Validation set.

The *training set* will be use to get the weight for each h_i. The learning algorithm for each model will never know of any sample in "validation set". This should make "validation set" a good proxy to measure the true E_{out}. After we get each h_i we measure the error on the *validation set*. We denote this error on the validation set $E^{\text{V}}_{\text{out}}$ then we will select the one that give us the best $E^{\text{V}}_{\text{out}}$. An example is shown below:

Model	$E^{\text{V}}_{\text{out}}$
h_1	0.2
h_2	0.1
h_3	0.05
h_4	0.07
h_5	0.2
h_6	0.3

From the above table we will then use h_3 as our hypothesis since it give us the lowest value for $E^{\text{V}}_{\text{out}}$.

The logic the above methodology can be summarize as the following

- Since we know that could overfit the data, E_{in} is not a reliable proxy for the true E_{out}. Selecting the best model from E_{in} would be like fooling ourself.
- To get a reliable proxy for E_{in}. We split the data into two sets. Once is use in training and the other is validation set.
- We train the hypothesis for various models using only the training set. The learning algorithm never seen the test data during the training. This should make $E^{\text{V}}_{\text{out}}$ a good proxy for E_{out}.
- We select among them the one that give us the best $E^{\text{V}}_{\text{out}}$.

This is all good except that we have to report how well our model is expected to work in real world application. Even though $E^{\text{V}}_{\text{out}}$ is a good proxy for E_{out}. But we already use it to pick the best model. So reporting $E^{\text{V}}_{\text{out}}$ to our customer how well our model work will bias us toward the lower value. This is not a good idea.

We can fix this using a similar approach. Just make more split in the training sample. We need one to train. We need one to pick the model. We need another one to measure the performance of the model we select. So we need to split the training sample into three set

1. Training set. To train hypothesis. Typically 60%.
2. Validation set. To select the model. Typically 20%.
3. Test set. To measure the performance. Typically 20%.

The percentage given at the end is just a guide line there is really no reason why you shouldn't change that to fit your purpose.

General Idea

The idea we discussed in the previous section go way beyond just picking the just-right degree of polynomial.

The same idea can be used to select the cutoff for Logistic regression. Just try them all and pick the one that give the lowest $E^{\text{V}}_{\text{out}}$

The same idea can be used to select the right amount of regularization parameter. Again, just try them all and select the one with the lowest $E^{\text{V}}_{\text{out}}$.

The same idea can be used to select the right amount of feature. Too much information is really not a good thing.

We will also learn later that for some model like Neural Network the complexity is can be adjusted and to pick the right complexity, we can use validation technique to do it too.