Week 3 - Assessing the Performance of the Model

Assessing performance of the model is done by measuring loss – the difference between a predicted value and the true value. The **general loss function** is given by;

$$L\left(y, f_{\hat{w}}(x)\right)$$

y is the actual value
$f_{\hat{w}}(x)$ is the predicted value, sometimes given as $\hat{f}(x)$ or $\hat{y}$

In the real world, it is possible for the cost of a loss to be asymmetrical – under-estimating the value of your house could result in a lower sales price, so loss of money. Over-estimating the value of your house may result in no-offers, so a waste of time. Depending on your particular situation, one of these may be less attractive than the other.

Simple measures of loss are symmetrical;

Absolute Error: $L\left(y, f_{\hat{w}}(x)\right) = |y - f_{\hat{w}}(x)|$
Squared Error: $L\left(y, f_{\hat{w}}(x)\right) = \left(y - f_{\hat{w}}(x)\right)^2$

Training Data

Training data is some subset of the dataset. It has known output values and data input values used as features. We use this training data to create our predicted set of coefficients $\hat{w}$ for our model (in the case of regression, we choose some function to fit, then use RSS to minimize the residual sum of squares of for the predicted coefficients).

Training Error

For a given loss function, the **training loss is the average loss over all the training data**;

Training Error: $\frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f_{\hat{w}}(x_i)\right)$

Note that the 1/N term is not used universally, but we will use it in this class.

Then using **squared error** as the loss function in **training error**;

$$\frac{1}{N}\sum_{i=1}^{N}\left(y_i, f_{\hat{w}}(x_i)\right)^2$$

When using **squared** error, the **training error simplifies to** $\frac{1}{N}$ $\boldsymbol{RSS}(\hat{\boldsymbol{w}}(\boldsymbol{x_i}))$, since RSS is the sum of the squared errors. So

Root Mean Squared Error is the square root of the training error. So this value depends upon the loss function. **For squared training error**, the RMSE is;

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_i, f_{\hat{w}}(x_i)\right)^2}$$

RMSE has the nice attribute that it **is stated in the same units as the predictions** (as opposed to the squared error, which is in units-squared of the predictions).

Training Error vs Model Complexity

For polynomial regression the simplest model is the constant model, then a simple line, then a quadratic, etc. as the complexity increases.

As model complexity increases, training error will decrease. However, if the model becomes too complex, then it may *overfit* the training data. **As the model gets more complex, it becomes very specific to the training data** and so it is less general – **an over-fitted model is less predictive of other data**. Another way to say this is that training error does not tell us how well the model matches the general data, it only tells us how well it matches the training data.

So training error is NOT an adequate measure of the performance of the model. The model must be measured against data that is not in the training set.

What we really want is 'generalization' or 'true' error. We want the error over all data we are likely to see. It is likely that each feature has values in some distribution over all the data. This is often bell shaped; feature values at the extremes are rare and feature values in the middle are more common. So in our house example, we can look at the distribution of square feet over all the houses we might see. It is unlikely to see very, very small houses or very, very large houses.

We can also fix the feature and look at the distribution of the output value for the fixed feature value. In our house example, we could look at the distribution of house prices for a given fixed square footage. For instance, what is the distribution of house prices for houses with 2000 square feet of space. Again, it is unlikely there

will be lots of very low prices or lots of very high prices for a house of that size; the distribution is likely to be some bell shape.

So **the generalization error weights the error by the probability that we will see that data point in the general population**.

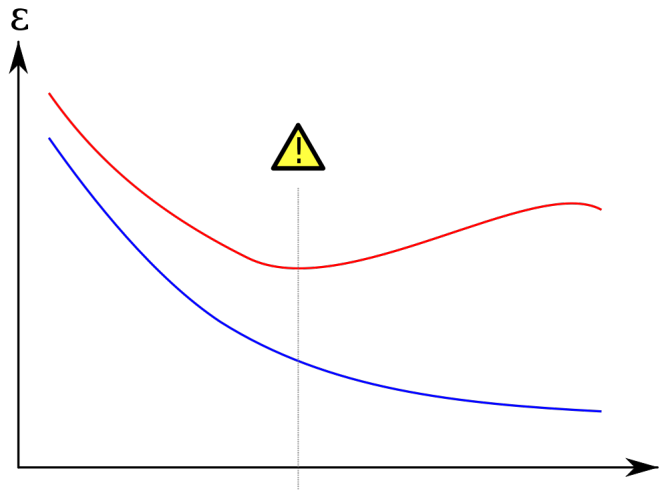
Generalization Error: $E_{\vec{x},y}\left[L\left(y, f_{\hat{w}}(\vec{x})\right)\right]$

$L\left(y, f_{\hat{w}}(\vec{x})\right)$ is the loss function

$E_{\vec{x},y}[]$ We average the loss over all possible $(\vec{x}, y)$ pairs weighted by how likely each is.

If we were to create models of increasing complexity against the same data and measure their generalization error vs complexity, we would get a convex curve – the very simplest models have high but decreasing generalization error. We would get to a minima where we have a complex enough model. The generalized error would then increase with model complexity because the model would be increasingly over-fitted to the training data and so would not model the general data as well.

Here is an illustration from this Wikipedia of how training error (the bottom, blue line) can decrease, but generalization error (the top, red line) can increase;



Error versus Model Complexity

The fact is, we cannot actually make this curve because **we can't actually calculate the generalization error**. We can't know how likely each (x,y) pair is, because we do not have all the data the world. If we had all the data in the world we would not need to make predictions!

We can't actually calculate generalization error, and we know that training error does not tell us how well we model the general data, it only tells us how well we model that training data, which can lead to over-fitting. So we need another error

measure that approximates generalization error. **We approximate generalization error with test error**. This is the error calculated with some dataset (the test data) that is held out of the training set, specifically so we can measure error on data that is NOT part of the training set. Hopefully the test data is representative of the overall population in the real world.

Specifically, **test error is the average loss over all data in the test set**.

Test Error: $\frac{1}{N_{test}} \sum_{i_{test}}^{N_{test}} L\left(y_i, f_{\widehat{w}}(\vec{x}_i)\right)$

N	is the **number of data points** in the *test* set.
i	is the **index of the i-th data point** with the *test* set.
y_i	**output value for the i-th point** in the *test* set.
$f_{\widehat{w}}(\vec{x}_i)$	is the **predicted output for the i-th data point** (given the features of that data point) using the predicted weights $\widehat{w}$ (created in a regression model by minimizing the RSS on the *training* data.)

Remember that $\widehat{w}$ **is the predicted weights (the coefficients) created using the *training* data** and **we are assessing the predictions of this model using the *test* data**.

Overfitting

A model is overfit if there exists another model that is less well fit to the training data (has a higher training error) AND that has less generalization error. Another way to say this is that a model is overfit if the model is better fitted to the training data than another model but less well fit to the general data than that other model. This would be seen on the graph of model error vs complexity as a model where the slope of the training error is negative but the slope of the test error is positive. In other words, we have gotten to a place on the graph where generalization error is increasing. For instance, look at the illustration above; Error versus Model Complexity, and look at the fourth tick on the horizontal axis. At that place the training error is going down, but the generalization error is going up.
More formally,

A model w'' is over-fitted if there exists another model w' where

- Training error of w'' < training error of w'
- True error of w'' > true error of w'

Splitting the Training and Test data

How do we 'best' split our data into a training set and a separate test set? There are to competing notions here.

- We want as much data in the training set as possible so our resulting model better reflects general data.

- We want as much data in the test set as possible so our test error better approximates true error.

Clearly, we can't give all the data to both the training and test sets, since these sets need to be mutually exclusive. There is no 'best' choice here. In general, we want to only use as much data for the test set as is necessary to get a good approximation of the general population and we want to use all the other data for the training set. A common split is 80% training, 20% test data.

Sources of Error

- Noise
- Bias
- Variance

Expected prediction error = σ^2 + bias^2 + variance^2

σ^2	the variance of the noise (the average of all predicted output points' squared deviation from the true curve)
bias^2	The difference between the 'average' or expected fit and the chosen fit.
Variance^2	The variation in the fit across all possible training sets of size N.

Noise is the error inherent in the data. It exists because the real world is far more complex that our dataset, and so our model, can indicate. Noise is the error term in our model assuming we can fit all possible data;

$$y_i = f_{w(true)}(x_i) + \varepsilon_i$$

Noise is irreducible error because we cannot eliminate it no matter how good our fit is because it is inherent in the data. The noise has zero mean and it has some spread; the statistical variance of the noise from that mean.

So noise cannot be eliminated; we do not have control over it. However, we do have control over the other two sources of error; bias and variance.

Bias is the difference between the 'average' fit and the true fit. The 'average' fit is the average of the fits over all possible training sets of a given size N times the likelihood of seeing that training set. The **average fit is also called the expected fit** for a training set of size N. So we are looking at the error in the particular model complexity we have chosen – how well can this model match the true model.

Variance is a measure of the spread (the differences) between all the possible fits for a given data set and the average fit for that dataset. Variance is telling us

how different the curves can be given different training sets. So overfit models will look very different with different training sets.

In **low complexity models, variance is generally low**. So provided we have enough data, the choice of training set does not change the outcome much from the expected or average fit. Think about a constant model – the fit will only vary based on the range of output values, however in a more complex model, the curve can swing wildly outside of those limits. Even though the variance is low in low complexity models, the average fit of a low complexity model might not be good, so **low complexity models can have high bias**.

In high complexity models, the data is fit very specifically to the training data, so the variance across different training sets is large. By choosing different points for our training set, we get very different curves because the curves are highly specific to the data; they are over-fit. **High complexity models have high variance**. Averaging all of these curves will still produce a reasonable average curve, so **high complexity models have low bias**.

Bias-Variance Trade-off

If we plot bias vs complexity we will get a curve that starts out high and slopes down (negative slope), because low complexity models have high bias and high complexity models have low bias.

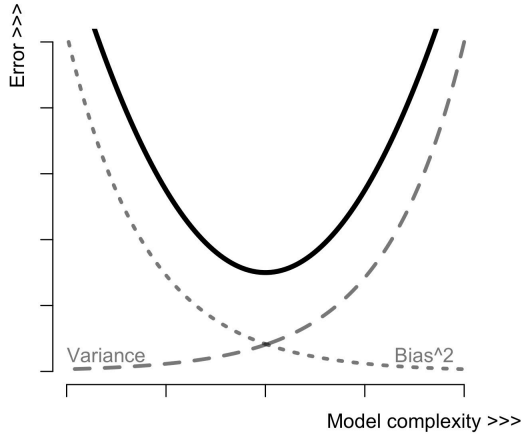
If we plot variance versus complexity, we will get a curve that starts out low and slopes up (positive slope) because low complexity models have low variance and high complexity models have high variance.

We can combine these into a single curve know as Mean Square Error (MSE).

$$\text{MSE} = \text{bias}^2 + \text{variance}$$

This creates a convex curve. The minimum is the place we want that trades off bias and variance the best.

This can be seen in this illustration from this blog;



Figuur 5

Error vs Bias, Variance and MSE

The problem is that **we can't directly compute either true bias or true variance** because **they are defined in terms of true fit and true error** and we can't measure these (remember that if we could, we would need to make predictions).

Error vs Amount of Training Data

We can look at error for a fixed model complexity as the amount of data used to train the model increases.

If we were to plot **true error** for a fixed model complexity **versus the amount of training data**, we would start out with high error and slope down to an asymptote where we cannot decrease the error any more. We start out high because, at the extreme, one data point is a poor sample of the whole world, so the error is high. As we include more data, the set better represents the real world, so error decreases. **At some point, even if we include all the data points in the world, we still cannot decrease the error**. This is because of a) the inherent noise b) the bias of the model. We know that the noise represents irreducible error, so that will always be there. We are also choosing a fixed model that will not be a perfect model of the real world, so there will be differences between the model and the real world, which is our bias. **The remaining true error is the noise + bias**.

Now if we plot **training error versus the amount of training data**, we get the opposite. Training error starts out relatively low because we can fit a small number of points very well. **As data points are added training error increases** (positive slope). As we approach all the points in the world, **the training error becomes asymptotic at the limit of true error: noise + bias**. This makes sense because if

we train on all the data in the world, then training error is actually true error and we know we cannot remove irreducible noise or the bias our chosen model has.

Here is a plot of error vs the amount of training data (validation error is used as an estimate of generalization error) taken from this blog. As we add more data, the lines converge, but not to zero. Where they converge represents the irreducible noise and bias errors.

These notions of noise, bias and variance as the sources of error in our model and their relationship to model complexity and true error are not specific to regression models. These notions can be applied to all kinds of machine learning models. In a regression model, we are choosing the number of coefficients for a polynomial. In other models, we are choosing different kinds of parameters. In general, we can call the parameters that control the complexity of the model tuning parameters and talk about noise, bias and variance as it relates to model complexity in terms of our choice of tuning parameters.

The general machine learning workflow attempts to choose a set of tuning parameters that minimizes generalization error – it attempts to choose a model complexity that is predictive of the general data. In the case of linear regression, when determining complexity of the model we are deciding on how many linearly independent terms are in our the model (which is a separate notion from choosing the best set of coefficients w for that model).

General Machine Learning Workflow

1. Choose a Model
   In the general ML workflow, we choose tuning parameters, λ for controlling the model. In the case of regression, we want to choose how complex our regression model is; how many terms are in the prediction function.

2. Assess the Model
   Assess the generalization error of the model at the given complexity.

To choose λ we need another set of data independent from our training and test sets. We use our training set to train the model and derive a set of coefficients, w. In our prior work, we've tested this fit using a test set of data that is held out of the training set (it is not used for training). Now we also need to decide the best tuning parameter λ and we need to test this choice so we find the best λ that minimizes generalization error. We can't use the test set for both test error and generalization error because if we do we will end up being overly optimistic about our choice of λ. This would be like testing the $w^{(t)}$ we choose against the training set – we would get an overly optimistic result that would lead us to create an over-fitted model, in this case our choose of λ would be overfitted to the test data.

We cannot use test error to both choose λ AND to assess it's approximate generalization error.

So **we need a third set of data, the validation set, to test our choice of λ**. So we split our data into training, validation and test sets.

1. **Split the data** into 3 sets, training, validation and test sets. Typical splits are 80:10:10 and 50:25:25
2. **Find the model tuning parameters with the lowest validation error**.
   For each considered model complexity λ:
   - **Estimate** the model **parameters $\widehat{w}_\lambda$ using the training** data
   - **Assess the performance** of the estimated parameters using **validation** data.
   - Choose the model complexity λ where $\widehat{w}_\lambda$ has the lowest validation error.
3. **Approximate the generalization error via test error**; assess the chosen $\widehat{w}_\lambda$ against the test data.