Machine Learning Basics : L - 01

GAJENDRA BABU (15MI431)

Regression Metrics

In a regression task, the model learns to predict numeric scores. An example is predicting the price of a stock on future days given past price history and other information about the company and the market.

Root-Mean-Squared-Error (RMSE)

The most commonly used metric for regression tasks is RMSE (Root Mean Square Error). This is defined as the square root of the average squared distance between the actual score and the predicted score:

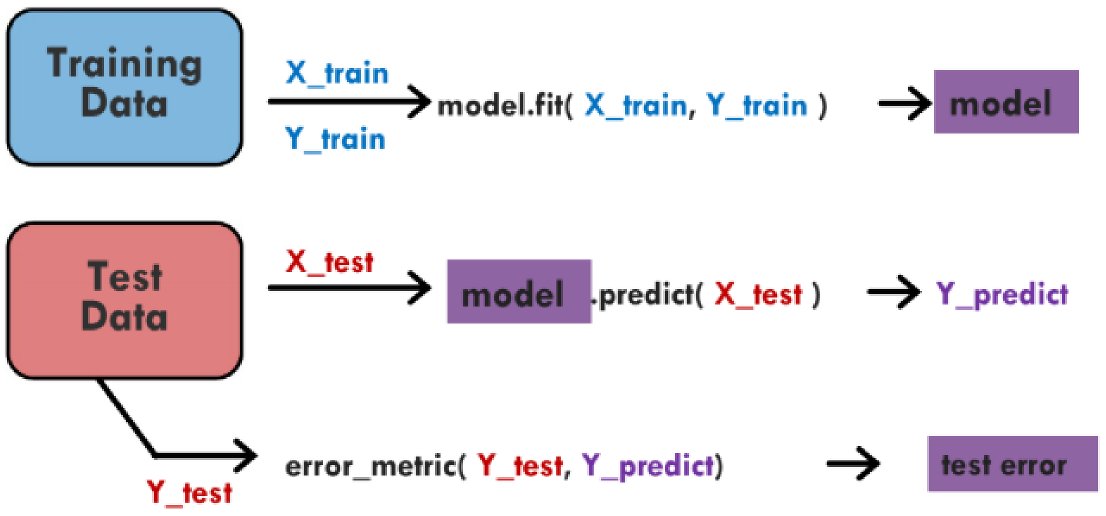
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Here, yi denotes the true score for the i-th data point, and y ̂i$^ denotes the predicted value. One intuitive way to understand this formula is that it is the Euclidean distance between the vector of the true scores and the vector of the predicted scores, averaged by n√, where n is the number of data points.

Mean Absolute Deviation

$$MAD = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Fitting Training and Test Data



Training Data

The observations in the training set form the experience that the algorithm uses to learn. In supervised learning problems, each observation consists of an observed output variable and one or more observed input variables.

Test Data

The test set is a set of observations used to evaluate the performance of the model using some performance metric. It is important that no observations from the training set are included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it.

A program that generalizes well will be able to effectively perform a task with new data. In contrast, a program that memorizes the training data by learning an overly complex model could predict the values of the response variable for the training set accurately, but will fail to predict the value of the response variable for new examples. Memorizing the training set is called **over-fitting**. A program that memorizes its observations may not perform its task well, as it could memorize relations and structures that are noise or coincidence. Balancing memorization and generalization, or over-fitting and under-fitting, is a problem common to many machine learning algorithms. **Regularization** may be applied to many models to reduce over-fitting.

Training and Test Sets: Splitting Data

The previous module introduced the idea of dividing your data set into two subsets:

- **training set**—a subset to train a model.
- **test set**—a subset to test the trained model.

You could imagine slicing the single data set as follows:

Figure 1. Slicing a single data set into a training set and test set.

Make sure that your test set meets the following two conditions:

- Is large enough to yield statistically meaningful results.
- Is representative of the data set as a whole. In other words, don't pick a test set with different characteristics than the training set.

Assuming that your test set meets the preceding two conditions, your goal is to create a model that generalizes well to new data. Our test set serves as a proxy for new data. For example, consider the following figure. Notice that the model learned for the training data is very simple. This model doesn't do a perfect job—a few predictions are wrong. However, this model does about as well on the test data as it does on the training data. In other words, this simple model does not overfit the training data.

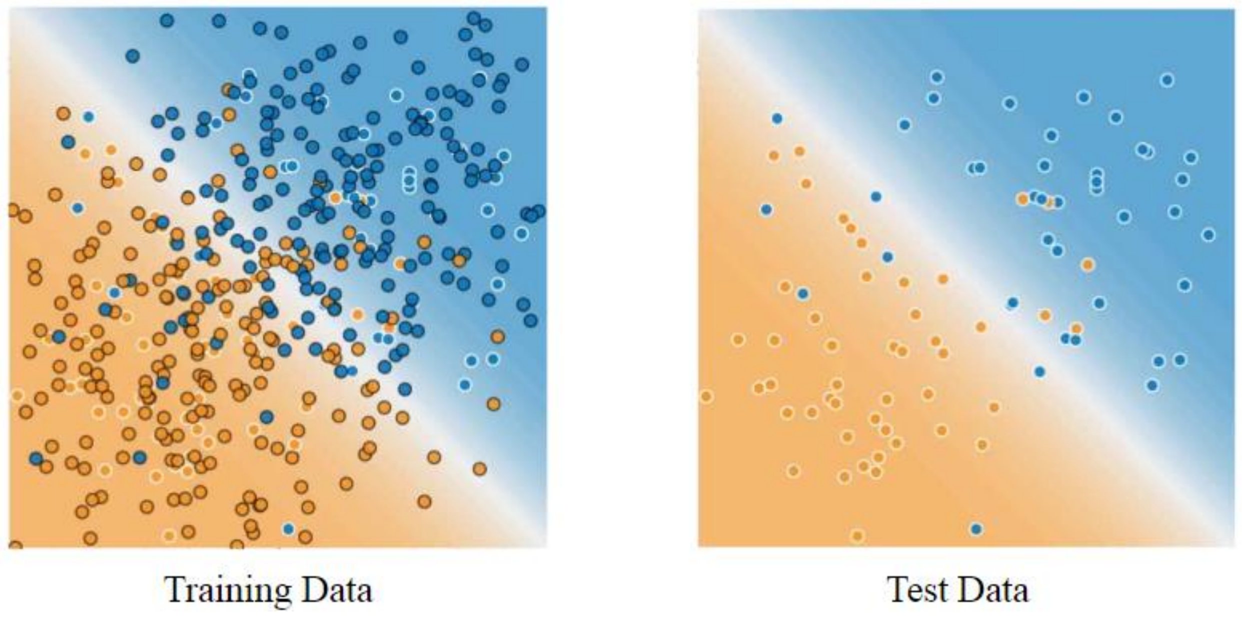
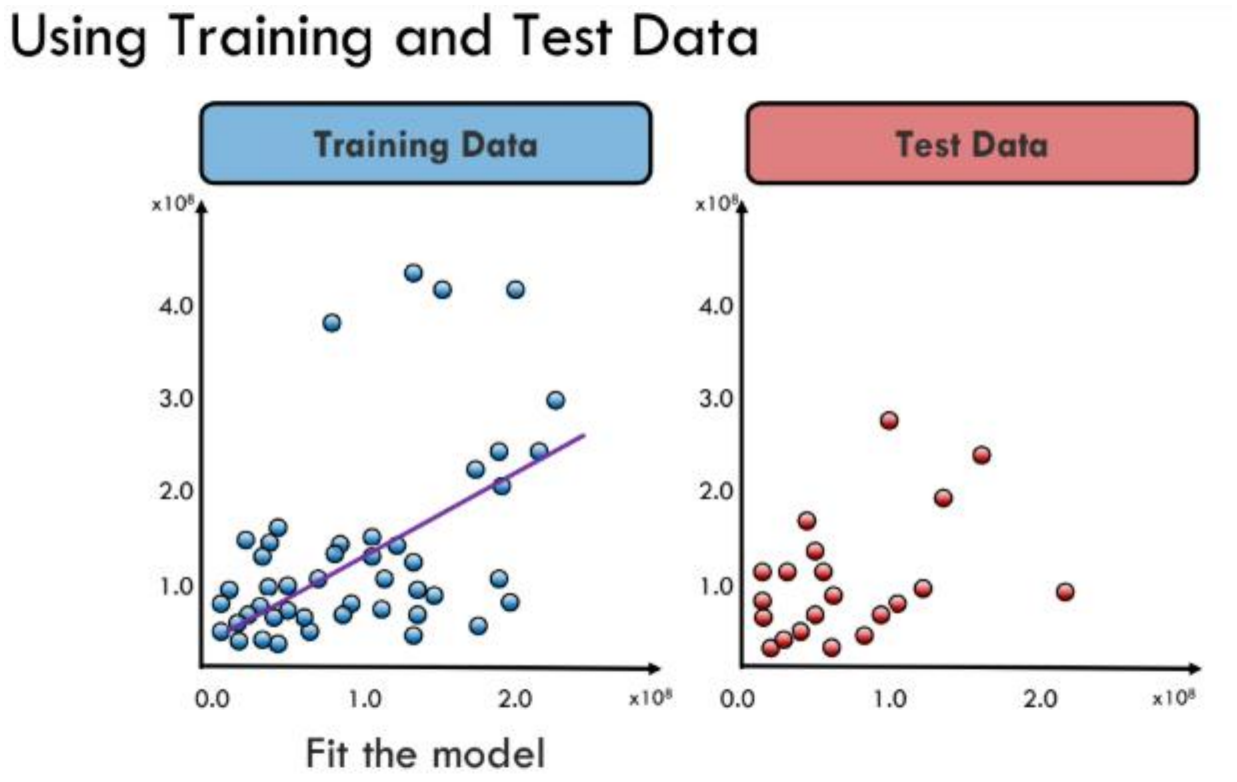


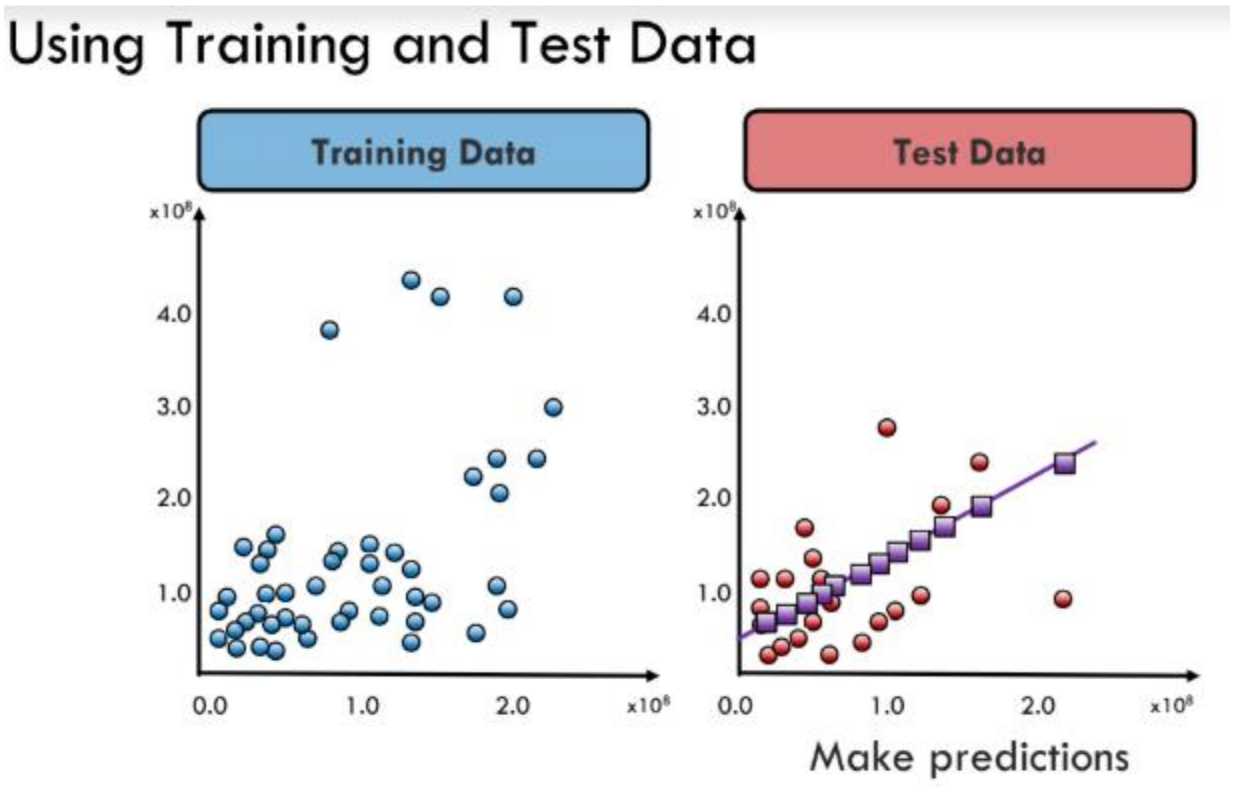
Figure 2. Validating the trained model against test data.

Never train on test data. If you are seeing surprisingly good results on your evaluation metrics, it might be a sign that you are accidentally training on the test set. For example, high accuracy might indicate that test data has leaked into the training set.

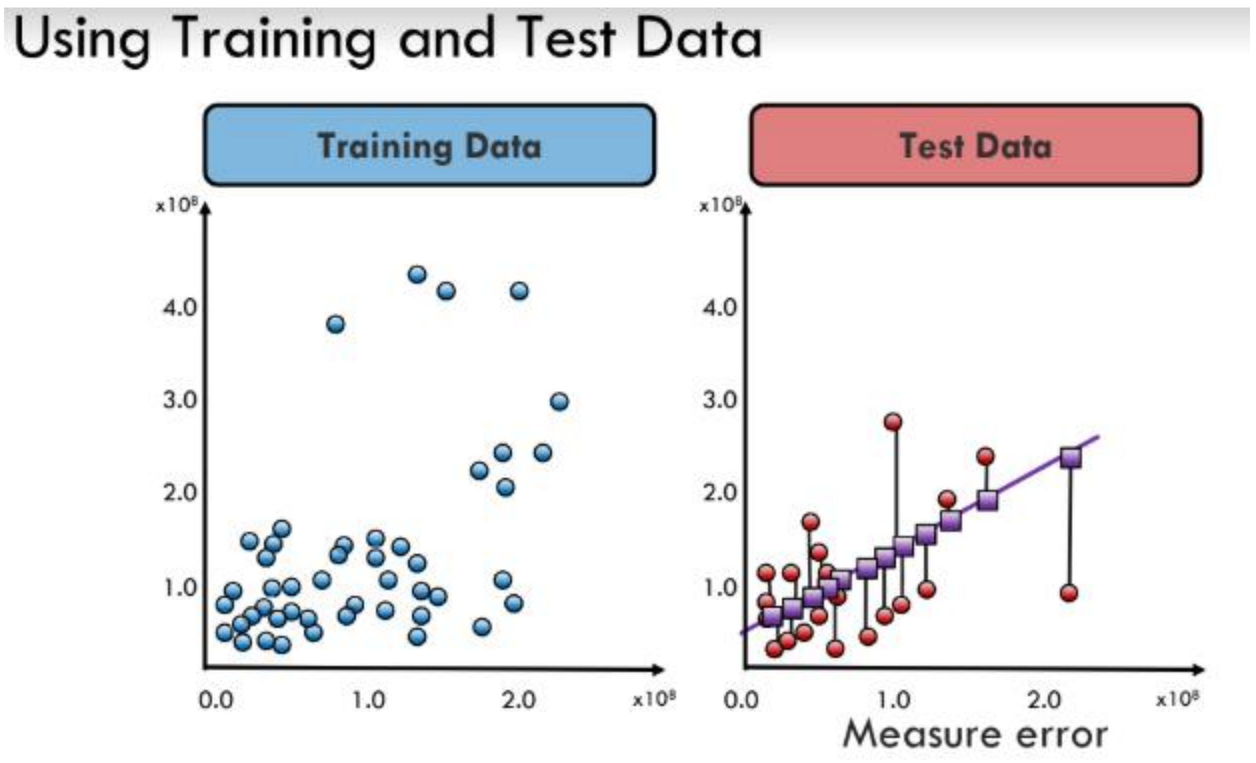
For example, consider a model that predicts whether an email is spam, using the subject line, email body, and sender's email address as features. We apportion the data into training and test sets, with an 80-20 split. After training, the model achieves 99% precision on both the training set and the test set. We'd expect a lower precision on the test set, so we take another look at the data and discover that many of the examples in the test set are duplicates of examples in the training set (we neglected to scrub duplicate entries for the same spam email from our input database before splitting the data). We've inadvertently trained on some of our test data, and as a result, we're no longer accurately measuring how well our model generalizes to new data.



Model fitting is a procedure that takes three steps: First you need a function that takes in a set of parameters and returns a predicted **data** set. Second you need an 'error function' that provides a number representing the difference between your **data** and the **model's** prediction for any given set of **model** parameters.



A **prediction** is what someone thinks will happen. A **prediction** is a forecast, but not only about the weather. Pre **means** “before” and diction has to **do** with talking. So a**prediction** is a statement about the future. It's a guess, sometimes based on facts or evidence, but not always.

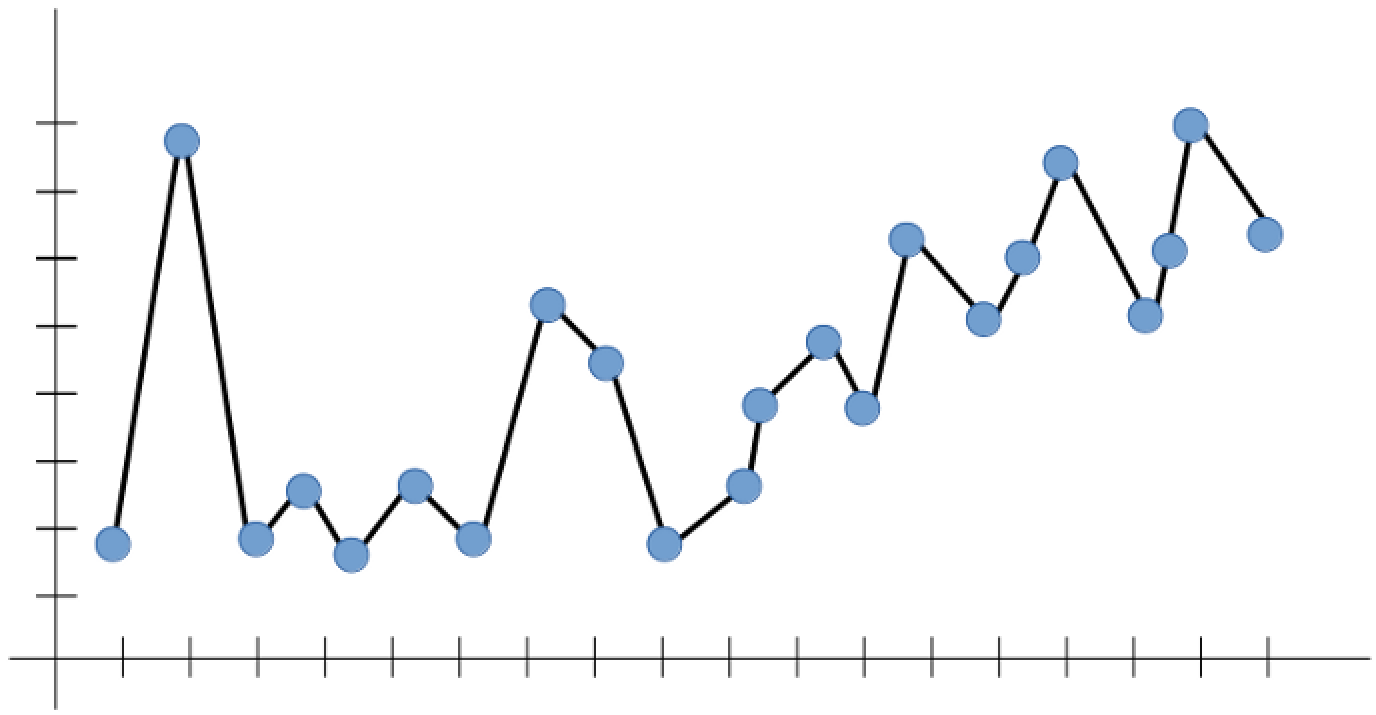


One thing I remember very clearly from writing my dissertation is how confused I initially was about which particular methods I could use to evaluate how often my models were correct or wrong. (A big part of my research was comparing human errors with errors from various machine learning models.) With that in mind, I thought it might be handy to put together a very quick equation-free primer of some different ways of measuring error.

Overfitting

When we run our training algorithm on the data set, we allow the overall cost (i.e. distance from each point to the line) to become smaller with more iterations. Leaving this training algorithm run for long leads to minimal overall cost. However, this means that the line will be fit into all the points (including noise), catching secondary patterns that may not be needed for the generalizability of the model.

Referring back to our example, if we leave the learning algorithm running for long, it cold end up fitting the line in the following manner:



This looks good, right? Yes, but is it reliable? Well, not really.

The essence of an algorithm like Linear Regression is to capture the dominant trend and fit our line within that trend. In the figure above, the algorithm captured all trends — but not the dominant one. If we want to test the model on inputs that are beyond the line limits we have (i.e. generalize), what would that line look like? There is really no way to tell. Therefore, the outputs aren't reliable. **If the model does not capture the dominant trend that we can all see (positively increasing, in our case), it can't predict a likely output for an input that it has never seen before — defying the purpose of Machine Learning to begin with!**

Overfitting is the case where the overall cost is really small, but the generalization of the model

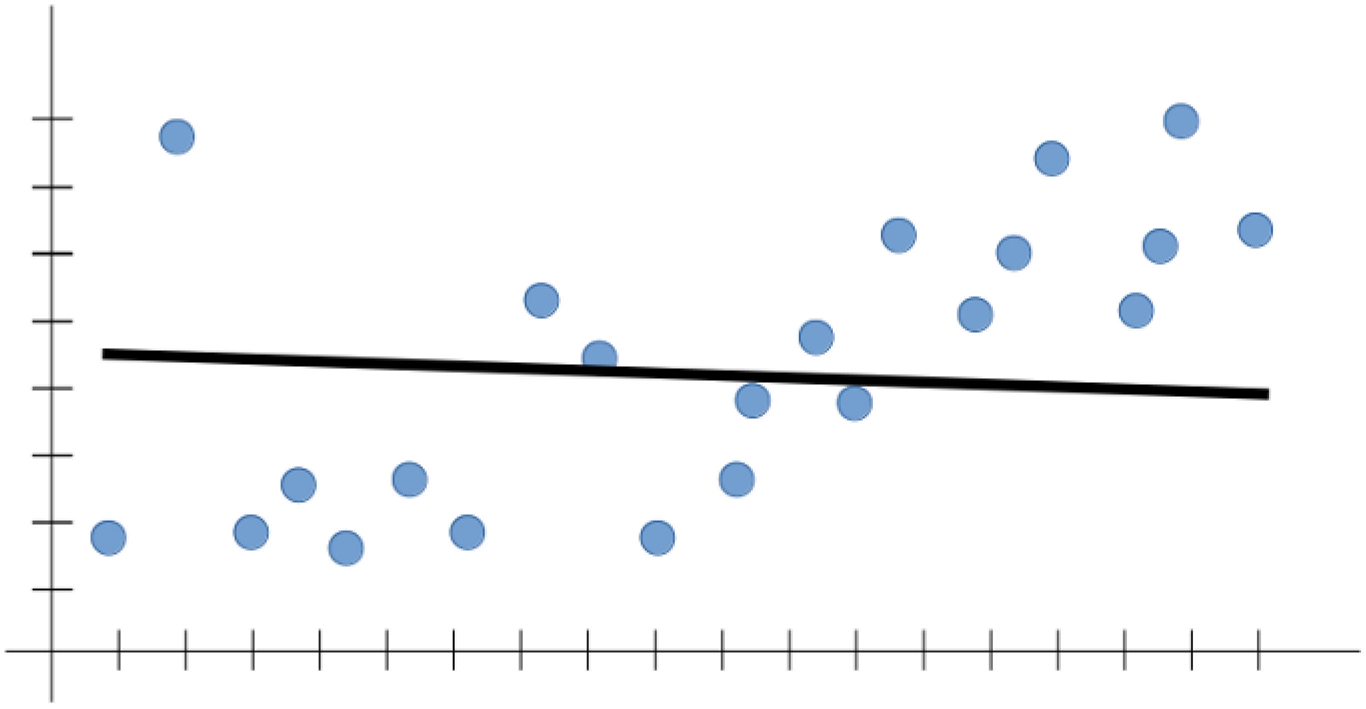
is unreliable. This is due to the model learning "too much" from the training data set.

This may sound preposterous, as why would we settle for a higher cost when we can just find the minimal one? Generalization.

The more we leave the model training the higher the chance of overfitting occurring. **We always want to find the trend, not fit the line to all the data points.** Overfitting (or high variance) leads to more bad than good. What use is a model that has learned very well from from the training data but still can't make reliable predictions for new inputs?

Underfitting

We want the model to learn from the training data, but we don't want it to learn too much (i.e. too many patterns). One solution could be to stop the training earlier. However, this could lead the model to not learn enough patterns from the training data, and possibly not even capture the dominant trend. This case is called underfitting.

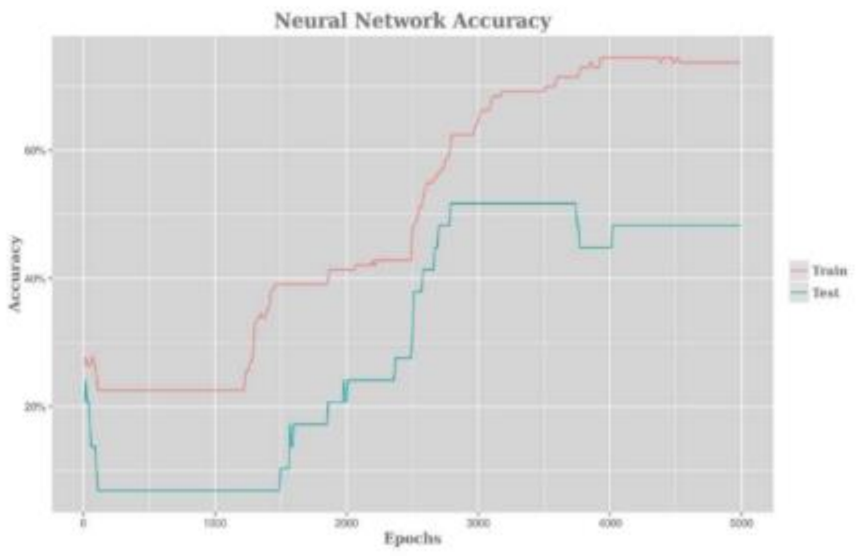


Underfitting is the case where the model has " not learned enough" from the training data,

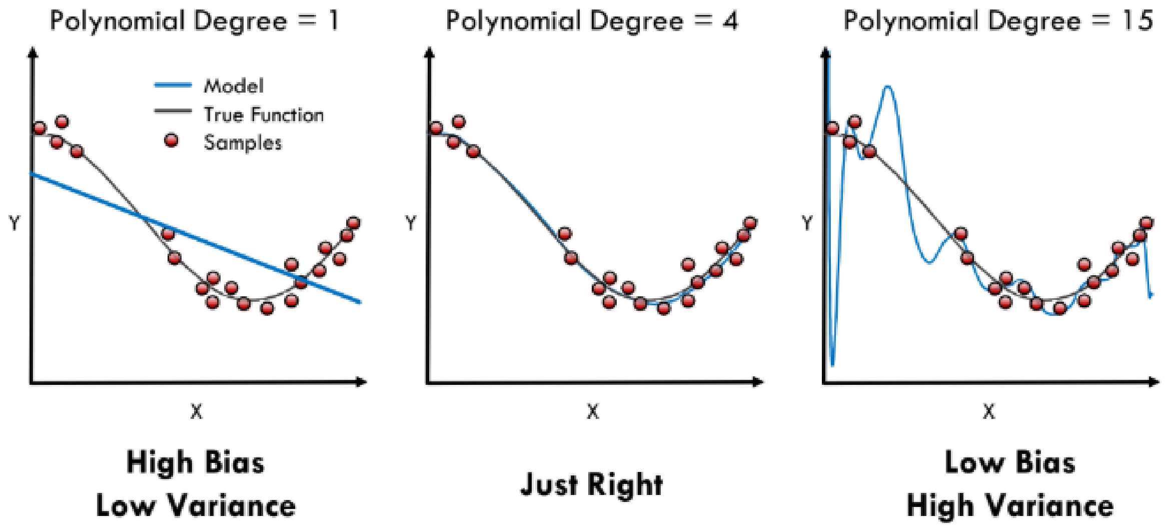
resulting in low generalization and unreliable predictions.

As you probably expected, underfitting (i.e. high bias) is just as bad for generalization of the model as overfitting. In high bias, the model might not have enough flexibility in terms of line fitting, resulting in a simplistic line that does not generalize well.

Underfitting vs Overfitting



Bias – Variance Tradeoff



Understanding the Bias-Variance Tradeoff

Whenever we discuss model prediction, it's important to understand prediction errors (bias and variance). There is a tradeoff between a model's ability to minimize bias and variance. Gaining a proper understanding of these errors would help us not only to build accurate models but also to avoid the mistake of overfitting and underfitting.

So let's start with the basics and see how they make difference to our machine learning Models.

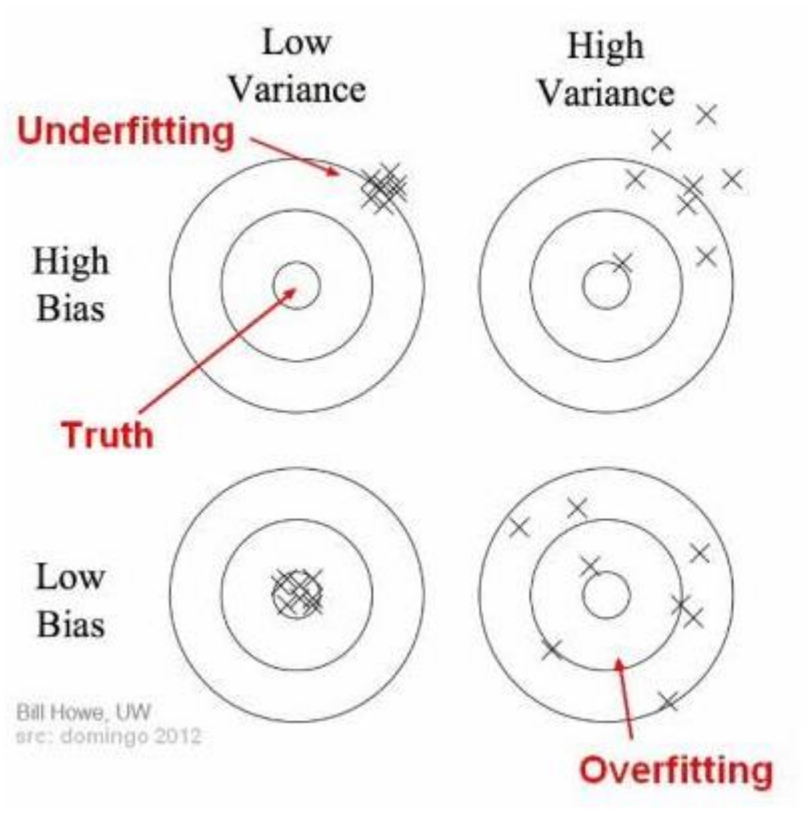
What is bias?

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

What is variance?

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

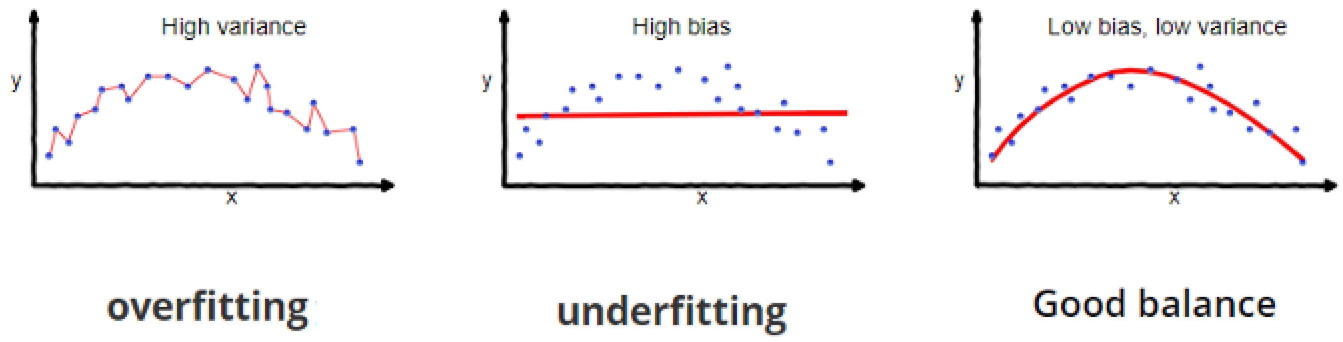
Bias and variance using bulls-eye diagram



In the above diagram, center of the target is a model that perfectly predicts correct values. As we move away from the bulls-eye our predictions become get worse and worse. We can repeat our process of model building to get separate hits on the target.

In supervised learning, **underfitting** happens when a model unable to capture the underlying pattern of the data. These models usually have high bias and low variance. It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data. Also, these kind of models are very simple to capture the complex patterns in data like Linear and logistic regression.

In supervised learning, **overfitting** happens when our model captures the noise along with the underlying pattern in data. It happens when we train our model a lot over noisy dataset. These models have low bias and high variance. These models are very complex like Decision trees which are prone to overfitting.



Why is Bias Variance Tradeoff?

If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.

This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time.