Learn OpenCV

Bias-Variance Tradeoff in Machine Learning

FEBRUARY 28, 2017 BY SATYA MALLICK (HTTPS://WWW.LEARNOPENCV.COM/AUTHOR/SPMALLICK/)

In this post, we will develop an intuitive sense for an important concept in Machine Learning called the Bias-Variance Tradeoff.

Before we dive into the subject, allow me to go off on a tangent about human learning for a little bit.

Practice alone does not make you better at a skill. We all know people who practice very hard but never seem to accomplish much. The reason is that they do not direct their effort appropriately. For a newbie who is learning the Piano, it is tempting to play the tune she has mastered, over and over again because it feels comfortable and it provides a lot of joy and sense of accomplishment. This behavior, though, does not help her improve her skill. The right way to practice is to identify your weakest areas and direct a massive effort on improving those areas without

worrying about areas in which you are already good. Psychologists call this **Deliberate Practice**. This form of practice is not very enjoyable. It is slow, frustrating and arduous. But Deliberate Practice is extremely effective in improving performance.

The same principle applies in Machine Learning. Enormous performance gains are made when you direct all your effort toward understanding your errors and minimizing them using known workflows. Otherwise, you will spend a lot of time trying different things without systematically reducing your errors.

Meet N00b — A Machine Learning Newbie

Let's first meet N00b. He is a smart programmer but a Machine Learning newbie. N00b is looking at his first real world machine learning problem. He has tried to find a solution on the internet, but on this rare occasion, the internet has disappointed him. His solution would need to be something better than *git clone* .

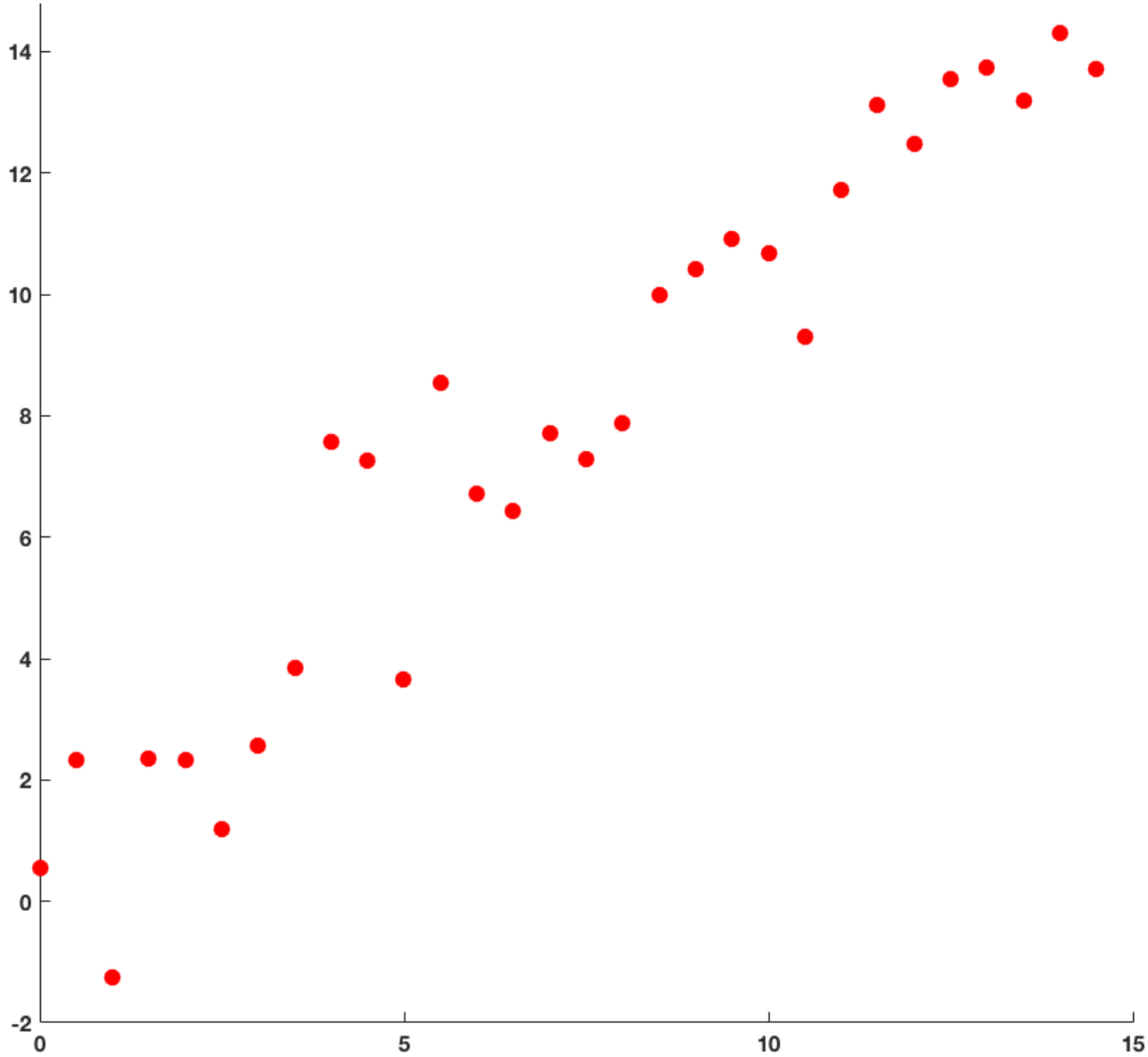
N00b thinks this is an opportunity disguised as a problem! He will find a good solution and post it online. He is afraid that if his solution is not excellent, users on Google+ will ignore him, users on StackOverflow will not upvote him and users on Reddit will hang him by his balls!

So N00b is determined to find the most awesome solution of all.

Dataset Preparation

N00b's data consists of 2D points (**x** and **y** coordinates) as shown below.


16


14
12
10
8
6
4
2
0
-2
0
5
10
15

(/wp-content/uploads/2017/02/data-points.png)

His goal is to build a model that can predict **y** for new, unseen values of **x**. He knows a few things about Machine Learning and uses the following steps to prepare his data.

1. **Shuffle data** : N00b randomly shuffles the order of his data first. It is an excellent step because many times the data we receive is ordered in some way. For example, it might be sorted by date. N00b knows these kinds of orders will invariably lead to funny biases in our models.
2. **Split data into training and test sets** : Noob splits his shuffled data into two parts — the **training set** consisting of 70% of the data and a **test set** consisting of 30% of the data. He will use the training set to fit a model and the test set to see how well he is doing. There are a surprisingly large number of engineers with the title “machine learning engineer” who make the rookie mistake of not separating their data into training and test sets. N00b knows this fact and feels superior. Unfortunately, N00b is doing it wrong. He should actually split the data into three sets — **training (60%)**, **validation (a.k.a development) (20%)** and **test (20%)**. He should be using the training set to train different models, the validation set to select a model and finally report performance on the test set.

3. **Model selection**: N00b plots his 2D data and spends some time just looking at this data. This again is an excellent step to follow. Staring at your data can provide surprising insights. Looking at the data, N00b thinks he can fit the data using a polynomial shown below.

$$y = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n = \sum_{i=0}^{n} a_i x_i \quad (1)$$

Training a model simply means finding good values for parameters $a_0, ..., a_n$ using the training set and testing how the model is doing using the test set. But what n should he choose? While finding the answer to this question, N00b would discover the Bias-Variance Tradeoff.

Machine Learning versus Curve Fitting

A friend of mine, who has interviewed many applicants for machine learning jobs, starts with the following simple question that trips off a surprising number of candidates.

What is the difference between Machine Learning and Curve Fitting?

In both Machine Learning and Curve Fitting, you want to come up with a model that explains (fits) the data. However, the difference in the end goal is both subtle and profound.

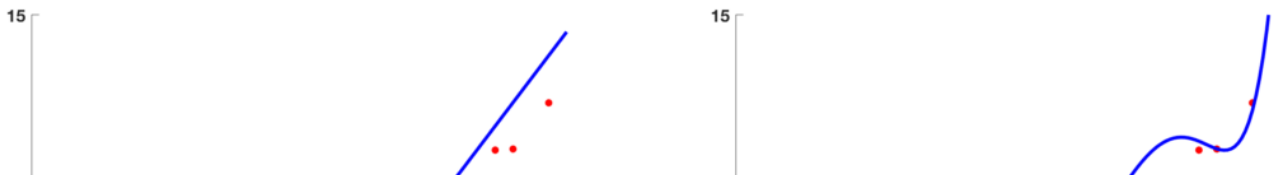
In Curve Fitting, we have all the data available to us at the time of fitting the curve. We want to fit the curve as best as we can.

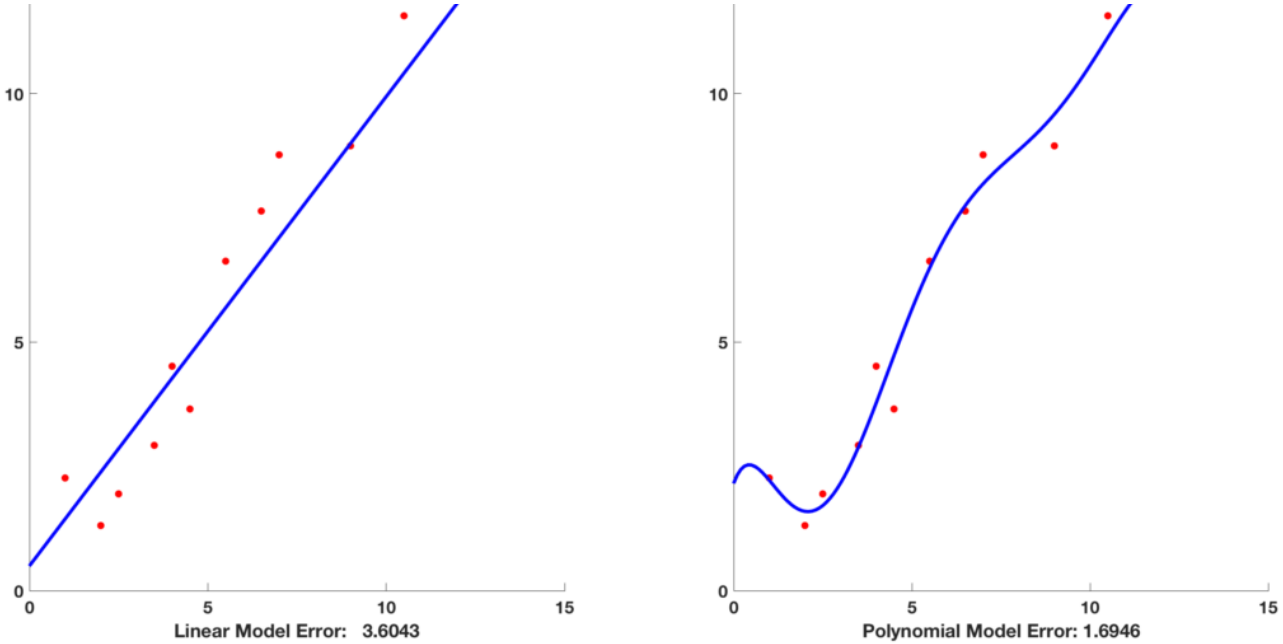
In Machine Learning, only a small set (the training set) of data is available at the time of training. We obviously want a model that fits the data well, but more importantly, we want the model to generalize to unseen data points. In Machine Learning, this presents a trade-off called the Bias-Variance Tradeoff.

Understanding the Bias-Variance Tradeoff

N00b needs to decide what degree, n, of the polynomial to use. He does not know what to choose, and so he conducts an experiment.

In the Figure below the red dots are 2D data points in the training set.





(/wp-content/uploads/2017/02/bias-variance-tradeoff.png)

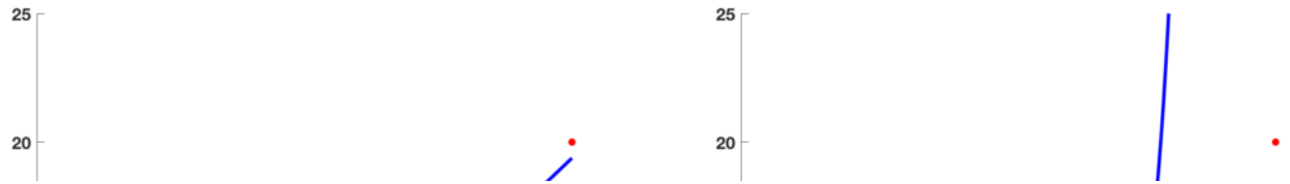
On the left, N00b fit a line to the data points. A line is just a polynomial of degree 1. Naturally, the line cannot pass through all the points, and there is an error between data (red dots) and the predicted value (blue line). In this example, the error is approximately 3.6. If this error is too large, we say the model is **underfitting** the data.

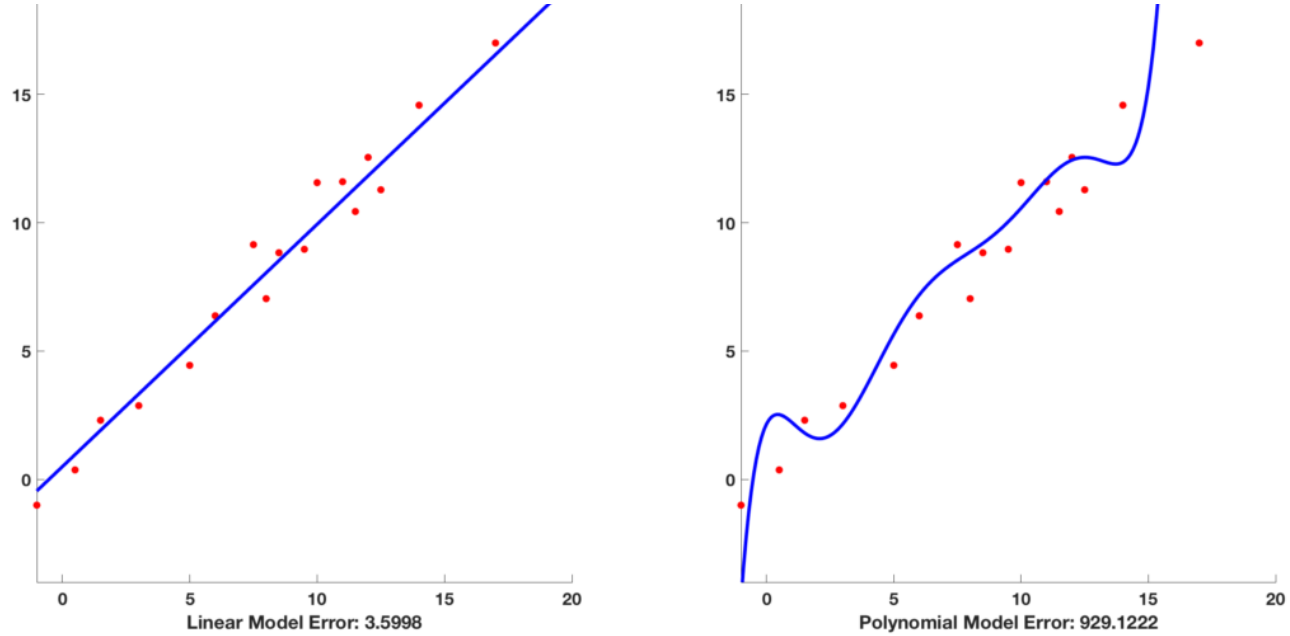
Can N00b do better? He can see that a straight line will never fit the data. He needs squiggly lines. He uses a polynomial of degree 8 and gets a squiggly line that fits the data much better. The error goes down to approximately 1.69.

The Linear model does not fit the data very well and is therefore said to have a higher **bias** than the polynomial model. N00b is excited by his new polynomial model and is tempted to use an even higher degree polynomial to obtain a squigglier curve to drive down the error to zero.

Before doing that N00b checks the performance of the two models on his test set. As you may recall, the test set was not used to train the model. The model has to perform better on this unseen dataset because that is what separates Machine Learning from Curve Fitting.

N00b plots the results and is shocked! His moment of ecstasy after seeing the error down on the training set was short lived.





(/wp-content/uploads/2017/02/bias-variance-tradeoff-test-error.png)

For the linear model, the error on this test set is very close to the error he had seen on the training set. In such cases, we say the model has **generalized** well to unseen data.

For the polynomial model, the error is astronomical (929.12)! The model he thought was excellent is pretty bad. This problem, where the model does very well on the training data but does poorly on the test data is called **overfitting**.

I had mentioned earlier, the Linear model had a higher **bias**. The polynomial model, on the other hand, suffers from a different problem. The model depends a lot on the choice of training data. If you change the data slightly, the shape of the curve will look very different, and the error will swing widely. Therefore, the model is said to have high **variance**.

N00b just got a taste of Bias-Variance Tradeoff. To keep the bias low, he needs a complex model (e.g. a higher degree polynomial), but a complex model has a tendency to overfit and increase the variance. He just learned an important lesson in Machine Learning —

> *Machine Learning is not a pursuit of perfection (i.e. zero error), but it is about seeking the best tradeoff.*

N00b is at the end of his rope here. So, let's help him out with some education.

Machine Learning Errors : Bias, Variance and the Optimum Error Rate

First, when you receive your data, divide it into three parts.

1. **Training set**: The training set is typically 60% of the data. As the name suggests, this is used for training a

machine learning model.

2. **Validation set**: The validation is also called the the **development** set. This is typically 20% of the data. This set is not used during training. It is used to test the quality of the trained model. Errors on the validation set are used to guide the choice of model (e.g. what value of n to use in our polynomial model). Even though this set is not used for training, the fact it was used for model selection makes it a bad choice for reporting the final accuracy of the model.
3. **Test set**: This set is typically 20% of the data. Its only purpose is to report the accuracy of the final model.

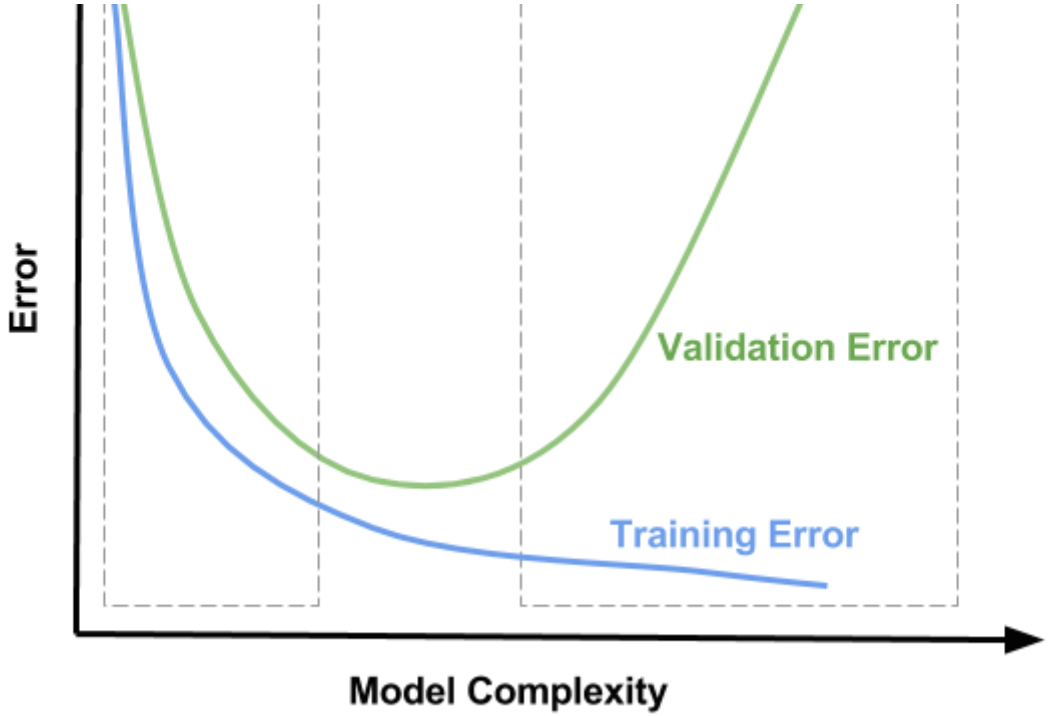
In Machine Learning, the errors made by your model is the sum of three kinds of errors — error due to bias in your model, error due to model variance and finally error that is irreducible. The following equation summarizes the sources of errors.

Total Error = Bias + Variance + Irreducible Error

Even if you had a perfect model, you might not be able to remove the errors made by a learning algorithm completely. This is because the training data itself may contain noise. This error is called **Irreducible error** or **Bayes' error rate** or **the Optimum Error rate**. While you cannot do anything about the Optimum Error Rate, you

can reduce the errors due to bias and variance.

If your machine learning model is not performing well, it is usually a high bias or a high variance problem. The figure below graphically shows the effect of model complexity on error due to bias and variance.



(/wp-content/uploads/2017/02/Bias-Variance-Tradeoff-In-Machine-Learning-1.png)

The region on the left, where both training and validation errors are high, is the region of high bias. On the other hand, the region on the right where validation error is high, but training error is low is the region of high variance. We want to be in the sweet spot in the middle.

How to detect a high bias problem?

A high bias problem has the following characteristics

1. High training error.
2. Validation error is similar in magnitude to the training error.

How to detect a high variance problem?

A high variance problem on the other hand has the following characteristics

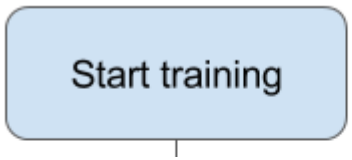
1. Low training error
2. Very high Validation error

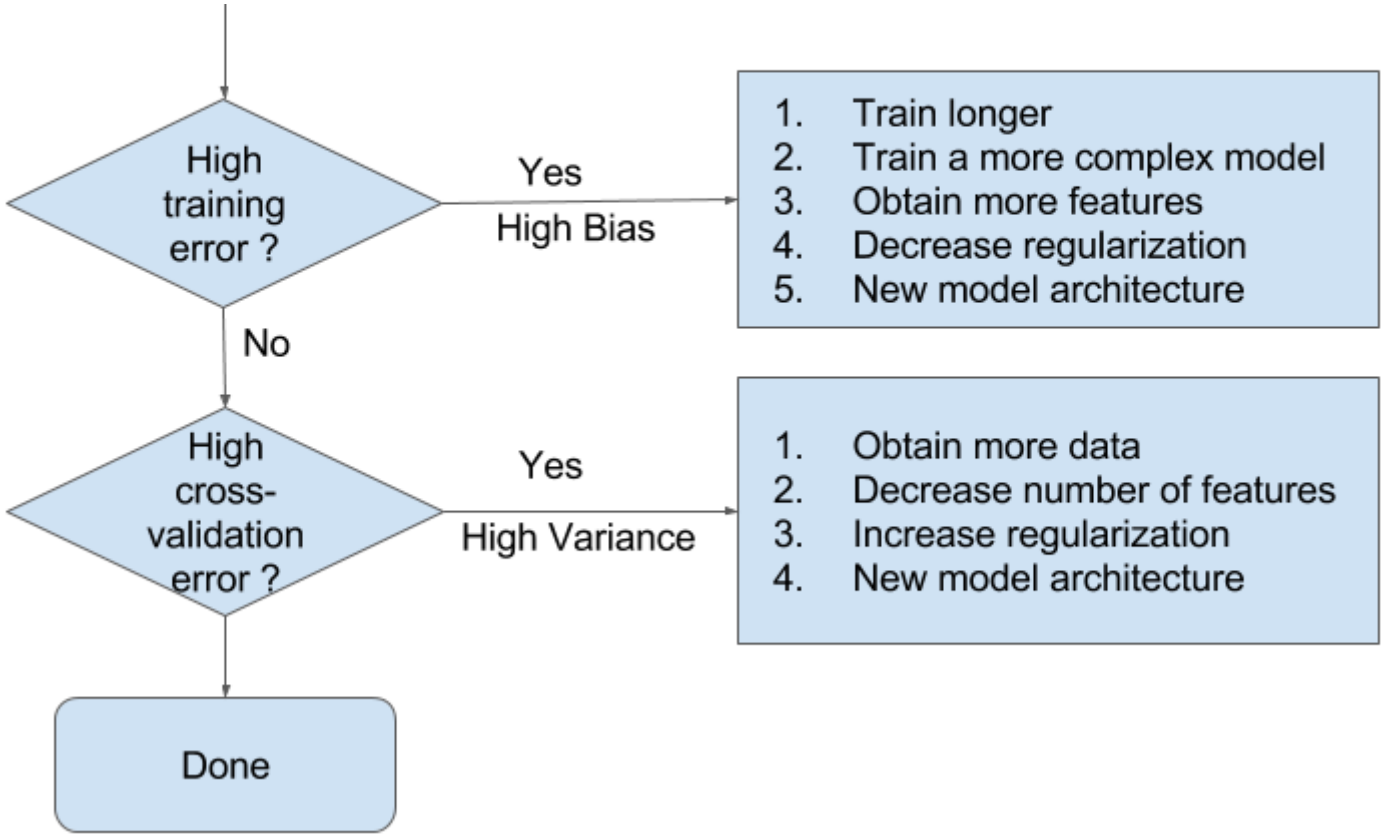
How to fix a high bias or a high variance problem?

When you are new to machine learning, you may feel lost when the model you have trained does not perform well. Often people waste a lot of time trying out different things based on what they feel is right. For example, N00b may be tempted to collect more data. Unfortunately, if he is dealing with a high bias problem, more data will not help at

all. N00b may also mindlessly use all the features available to him. Using all the features may hurt him if he has a high variance problem.

Fortunately, N00b can stand on the shoulders of giants and learn from a simple flowchart shown below. The inspiration for this diagram comes from a few videos in which Dr. Andrew Ng shares how to attack the Bias-Variance problem in a systematic way.





(/wp-content/uploads/2017/02/Machine-Learning-Workflow.png)

How to fix a high bias problem?

The following tricks are employed to fix a high bias problem.

1. **Train longer**: Many machine learning algorithms are set up as iterative optimization problems where the

training error (or a function of the training error) is minimized. Just letting the algorithm run for more hours or days can help reduce the bias. In Neural Networks, you can change a parameter called the “learning rate” which will help the training error go down faster.

2. **Train a more complex model**: A more complex model will fit the training data better. In the problem N00b was looking at, he could increase the degree of the polynomial to get a more complex model. In the case of Neural Networks, one can add more layers. Finally, in the case of an SVM, you can use a non-linear SVMs instead of a linear one.
3. **Obtain more features**: Sometimes you just do not have enough information to train a model. For example, if you are trying to train a model that can predict the gender of a person based on the color of their hair, the problem might be impossible to solve. But if you add a new feature — the length of the hair — the problem becomes more tractable.
4. **Decrease regularization**: Recall that N00b was trying to fit a polynomial to his data. In his naive implementation the parameters $a_0, ..., a_n$ could take any values. This makes the model very flexible. However, we may want to constrain the flexibility of the model to prevent overfitting. So usually a regularization term ($\lambda(a_0^2 + ... + a_n^2)$) is added to the cost function we are trying to minimize. But in this case, we do not have an overfitting problem. We have an under-fitting problem, and we can reduce it by reducing the value of λ.
5. **New model architecture**: This is just another way of saying that if nothing works, start over.

How to fix a high variance problem?

After you have addressed the high bias problem, you need to check if you have a high variance issue.

In this situation, your model is complex enough that it overfits your data. The following tricks should be employed to deal with overfitting.

1. **Obtain more data**: Because the validation error is large, it means that the training set and the validation set that were randomly chosen from the same dataset, somehow have different characteristics. This usually means that you do not have enough data and you need to collect more.
2. **Decrease number of features**: Sometimes collecting more data is not an option. In that case, you can reduce the number of features. You may have to remove features manually. For example, in our previous example of identifying the gender of a person based on hair color and hair length, you may decide to drop hair color and keep hair length.
3. **Increase regularization**: When we have a high variance problem the model is fitting the training data. In fact, the model is probably fitting even the noise in training set and therefore not performing as well on the validation set. We can reduce the flexibility of the model by using regularization that puts constraints on the magnitude of the parameters. This is done by adding a regularization term to the cost function. When we are fitting a polynomial model, the regularization term is of the form $\lambda(a_0^2 + ... + a_n^2)$. So we are not only trying to

   reduce the training set error, we are also trying to make sure the magnitudes of the parameters are small, thus limiting the flexibility of the model. Increasing λ increases regularization.
4. **New model architecture**: Try something else. Better luck next time!

Subscribe & Download Code

I hope you liked this article. If you would like to download code (C++ and Python) and example images used in other posts of this blog, please subscribe (https://bigvisionllc.leadpages.net/leadbox/143948b73f72a2%3A173c9390c346dc/5649050225344512/) to our newsletter. You will also receive a free Computer Vision Resource (https://bigvisionllc.leadpages.net/leadbox/143948b73f72a2%3A173c9390c346dc/5649050225344512/) Guide. In our newsletter, we share OpenCV tutorials and examples written in C++/Python, and Computer Vision and Machine Learning algorithms and news.

Subscribe Now
(https://bigvisionllc.leadpages.net/leadbox/143948b73f72a2%3A173c9390c346dc/5649050225344512/)