Overfitting vs. Underfitting: A Complete Example

Will Koehrsen | Follow
Jul 24, 2018 · 11 min read

Exploring and solving a fundamental data science problem

When you study data science you come to realize there are no truly complex ideas, just many simple building blocks combined together. A recent network may seem extremely advanced, but it's really just a combination of numerous small ideas. Rather than trying to learn everything at once when you start out in data science, it's more beneficial to pick a fundamental idea and through one block at a time. This ensures you have a solid idea of the fundamentals *and* avoid many common mistakes that will hold up others. Moreover each piece opens up new concepts allowing you to continually build up knowledge and you can create a useful machine learning system and, just as importantly, understand how it works.

Out of sample data values passed to pipeline | Source

This post walks through a complete example illustrating an essential data science building block: the underfitting vs overfitting problem. We'll implement a simple model, see how it performs on unseen data (validation), another important principle of model development, if you're looking for a conceptual framework on the topic, see my previous post. All of the graphs and results presented in this post are written in Python code which is on GitHub. I encourage anyone to go check out the code and make their own changes!

Model Basics

In order to talk about underfitting vs overfitting, we need to start with the basics of a model. A model is a function that maps inputs to outputs. For example, if we want to predict house prices, we could make a model that takes in the square footage of a house and outputs a price. A model represents a data set in a problem: there is some connection between the square footage and the price and we make a model to learn this connection. We can then use the model to predict the price for new values of square footage for data points which we haven't seen.

A model learns relationships between the inputs, called features, and outputs, called labels, from a training dataset. During training the model is given both the features and the labels so it can learn how to map from features to labels. A trained model is evaluated on a testing set, where we only give it the features and it makes predictions. We compare the predictions with the known labels for the testing set to calculate accuracy. Models can take many shapes, from simple linear regressions to deep neural networks, but all supervised models are based in the fundamental idea of learning relationships between inputs and outputs from training data.

Training and Testing Data

To make a model, we first need data that has an underlying relationship. For this example, we will create our own simple dataset with a single (features) and a value (labels). An important part of our data generation is adding random noise to the labels. In any real-world process, whether natural or man-made, the data does not exactly fit a trend. There is always some or other variability in the relationship we cannot measure. In the house price example, the trend between area and price is linear, but the prices do not exactly on a line because of other factors influencing house prices.

Sample of 1 and some of the corresponding | Source

Our data similarly has a trend (which we call the true function) and random noise to make it more realistic. After creating the data, we split it into random training and testing sets. The model will attempt to learn the relationship on the training data and be evaluated on the test data. In this case, 70% of the data is used for training and 30% for testing. The following graph shows the data we will explore.

Data and True Underlying Relationship

We can see that our data are distributed with some variation around the true function (a partial sine wave) because of the random noise we added to the data. Our goal is to make a model that uses our model to learn the true function without being "distracted" by the noise.

Model Building

Choosing a model can seem intimidating, but a good rule is to start simple and then build your way up. The simplest model is a linear regression, where the outputs are a linearly weighted combination of the inputs. In our case, we have one input variable (square footage) and one output (price). Scikit-Learn implements a linear relationship between 1 and 5 Polynomial regression, where the inputs are raised to different powers, is still considered a form of "linear" regression even though the graph does not form a straight line (this confused me at first as well!) The general equation for a polynomial is below.

$$y = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_5 x^5$$

Here y represents the label and x is the feature. The beta terms are the model parameters which will be learned during training, and the epsilon is the error present in any model. We fit the model has learned the beta values, we can plug in any value for x and get a corresponding prediction for y. A polynomial of degree 1 is a line, a degree 2 is a parabola, and so on for higher equations. A simple line is a polynomial of degree 1 while a parabola has 2 degrees.

Polynomials of Varying Degree | Source

Overfitting vs. Underfitting

The problem of overfitting vs. underfitting finally appears when we talk about the polynomial degree. The degree represents how much flexibility is in the model, with higher degrees allowing more complex functions to fit many data points so a parabola, for example, can model two data points and cannot account for the data. The best way to understand the issue is to take a look at models demonstrating both situations.

Under-fit 1-degree polynomial model on training (left) and testing (right) datasets

Our model passes straight through the training set with no regard for the data? This is because an underfit model has low variance and high bias. Variance refers to how much the model is dependent on the training data. For the case of a 1-degree polynomial, the model depends very little on the training data because it barely pays any attention to the points' learned, the line fits in the middle of the training points regardless of the position of the data. For this example, for assumption is that the data is linear, which is evidently quite wrong. When the model makes new predictions, the line leads it to make inaccurate estimates. The model failed to learn the relationship between x and y because of this bias, a clear example of underfitting.

Over-fit 15-degree polynomial model on training (left) and testing (right) datasets

We make a low degree leads to underfitting. A natural conclusion would be to have the training data, we should increase the degree of the model to capture every change in the data. This however is not the best decision!

Over-fit 15-degree polynomial model on training (left) and testing (right) datasets

With such a high degree of flexibility, the model does its best to account for every single training point. This might seem like a good idea — don't we want to learn from the data? Therefore, the model has a great score on the training data because it gets every point of the points. While this would be acceptable if the training observations perfectly represented the true function, because there is noise in the data, our model ends up fitting the noise. This is a model with a high variance, because it will change significantly depending on the training data. The model has learned the noise so well that when the new data (test data) arrives, the errors for the model will does not learn the relationship because it essentially memorizes the training data and the noise.

Our problem is that we want a model that does not "memorize" the training data, but learns the actual relationship between the inputs and the labels. Model with the right polynomial degree? If we choose the model with the best score on the training set, we will just select the overfitting model but that cannot generalize well to testing data. Fortunately, there is a well-established data science technique for diversifying the optimal model: validation.

Validation

We need to create a model with the best settings (the degree), but we don't want to have to keep going through training and testing. There are no hyperparameters in the model to optimize manually, however, in other applications where we might be performing a critical task such as diagnosing cancer, there would be serious downsides to deploying a faulty model. We need some sort of way to see our model optimization and validation. This pre-test is known as a validation set.

A basic approach would be to use a validation set in addition to the training and testing set. This presents a few problems though: we could just end up overfitting to the validation set and we would have less training data. A smarter implementation of the validation concept is k-fold cross-validation.

The idea is straightforward: rather than using a separate validation set, we divide the training data into k folds (typically $k = 4$ or $k = 5$). We then fit k folds as an example. We perform a series of train and evaluate cycles where each time we train on 4 of the folds and test on the 5th, called the hold-out set. We repeat this cycle 5 times, each time using a different fold for evaluation. At the end, we average the scores for each of the folds to determine the overall performance of a given model. This allows us to determine the model before deployment without having to use additional data.

5-Fold Cross-Validation | Source

For our problem, we can use cross-validation to select the best model by creating models with a range of different degrees, and evaluate each one using 5-fold cross-validation. The model with the lowest cross-validation score will perform best on the testing data and will achieve a balance between underfitting and overfitting. We can implement this in Scikit-Learn from 1 to 40 to cover a wide range. To compare models, we compute the mean squared error, the average distance between the prediction and the real value squared. The following table shows the cross validation results ordered by lowest error and the graph shows all the results with error on the y axis.

Degree	Average Validation MSE
1	0.0253
2	0.0252
3	0.0251
4	0.0251
5	0.0251
6	0.0251
7	0.0251
8	0.0251
9	0.0251
10	0.0251
11	0.0251
12	0.0251
13	0.0251
14	0.0251
15	0.0251
16	0.0251
17	0.0251
18	0.0251
19	0.0251
20	0.0251
21	0.0251
22	0.0251
23	0.0251
24	0.0251
25	0.0251
26	0.0251
27	0.0251
28	0.0251
29	0.0251
30	0.0251
31	0.0251
32	0.0251
33	0.0251
34	0.0251
35	0.0251
36	0.0251
37	0.0251
38	0.0251
39	0.0251
40	0.0251

Cross Validation Results

The cross validation error with the underfit and overfit models is off the charts, so we can only see the differences between the 1 and 5 degree results, we can make a 5-degree model and view the training and testing predictions.

Training and Testing Curves

There's nothing more beautiful than a model that fits the data! Moreover, we know that our model not only closely follows the training data, it has actually learned the relationship between x and y.

To verify we have the optimal model, we can also plot what are known as training curves. These show the training and testing error as a function of the training set size. The following graph shows the curves for a model that is underfitting, a model with have high training error and high testing error while an overfit model will have extremely low training error but a high testing error.

Training and Testing Curves

This graph nicely summarizes the problems of overfitting and underfitting. As the flexibility in the model increases (the meaning the polynomial degree) the training error continually decreases due to increased flexibility. However, the error on the testing set initially does decrease as we add flexibility up to a certain point. In this case, that occurs at 5 degrees in the flexibility of the model. Beyond this point, the model is learning the training data too well has memorized the training data and does not generalize. Cross-validation yielded the second best model on this training data, but in the long run we expect our cross-validation model to perform best. The exact metrics depend on the testing set, but on average, the best model from cross-validation will outperform all other models.

Conclusions

Overfitting and underfitting is a fundamental problem that trips up even experienced practitioners. It is easy to fall into the trap of building a complex model with extremely low error to train data and then eagerly write a paper with the results. Their model looks good, but the problem is they never even used a testing set let alone a validation set! The model is nothing more than an overly representation of the training data, a lesson the student must learn when someone else tries to apply their model to new data.

Fortunately, this is a mistake that we can easily avoid now that we have seen the importance of model validation and optimization using cross-validation. Once we understand the basic problems in data science and how to address them, we can feel confident in building up more complex models and helping others avoid mistakes. This post covered a lot of topics, but we started with the basic concepts of a model, training and testing, overfitting vs. underfitting, bias vs. variance, and model optimization with cross-validation. Data science is all about being willing to learn and continually adding more tools to your skillset. The field is exciting both for its potential beneficial impacts and for the opportunity to constantly learn new techniques.

I welcome feedback and constructive criticism. I can be reached on Twitter at @wkoehrsen, web. I would like to thank the contributors to Scikit-Learn for their excellent example on this subject.

Sign up for The Daily Pick
By Towards Data Science
Hands-on real world examples, research, tutorials, and cutting edge techniques delivered Monday, Wednesday, Friday, straight to your inbox. See a sample
Get this newsletter
Continue this Medium account to get the Daily Pick in your inbox.
or continue with:
Machine Learning · Data Science · Statistics · Technology
75 claps · 2 responses

Written by
Will Koehrsen
Data Scientist at Cortex Intel, Data Science Communicator

Follow

Towards Data Science
A medium publication sharing concepts, ideas, and code.

Follow

More From Medium

Amazon Wants to Make You an ML Practitioner— For Free

Anthony Agnone in Towards Data Science

Stop persisting pandas data frames in CSVs

Vaclav Dekanovsky in Towards Data Science

You're living in 1985 if you don't use Docker for your Data Science Projects

Sohaib Ahmad in Towards Data Science

How I'd start learning machine learning again (3-years in)

Daniel Bourke in Towards Data Science

Full Stack Data Science: The Next Gen of Data Scientists Cohort

Jay Kachhadia in Towards Data Science

The Best Data Science Certification You've Never Heard Of

Nicole Janeway Bills in Towards Data Science

What makes a data analyst excellent?

Cassie Kozyrkov in Towards Data Science

Here is What I've Learned in 2 Years as a Data Scientist

Admond Lee in Towards Data Science

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just $5/month. Upgrade

Medium

About Help Legal