Memorizing is not learning! — 6 tricks to prevent overfitting in machine learning.

Julien Despois Follow

Mar 22, 2018 · 7 min read ★

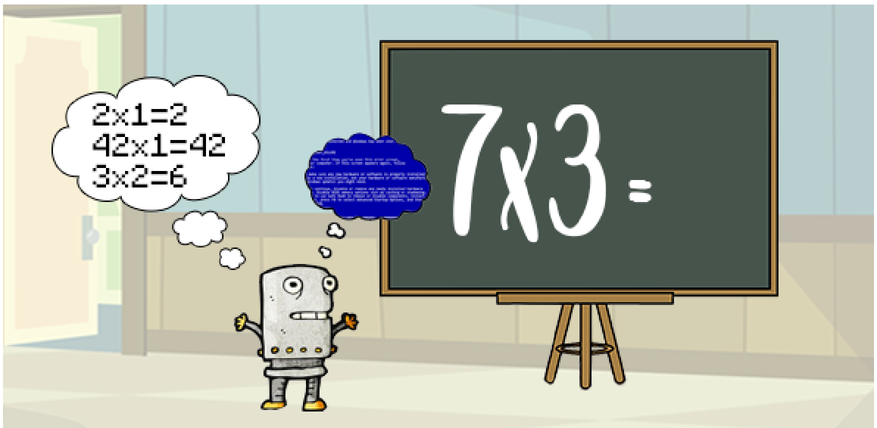
Introduction

Overfitting may be the most frustrating issue of *Machine Learning*. In this article, we're going to see **what it is**, **how to spot it**, and most importantly **how to prevent it from happening**.

What is overfitting?

The word **overfitting** refers to a model that models the training data too well. Instead of learning the genral **distribution** of the data, the model learns the *expected output* for every data point.

This is the same a **memorizing the answers** to a maths quizz instead of **knowing the formulas**. Because of this, the model cannot *generalize*. Everything is all good as long as you are in *familiar territory*, but as soon as you step outside, you're lost.



Looks like this little guy **doesn't know how** to do a multiplication. He only **remembers** the answers to the questions he has already seen.

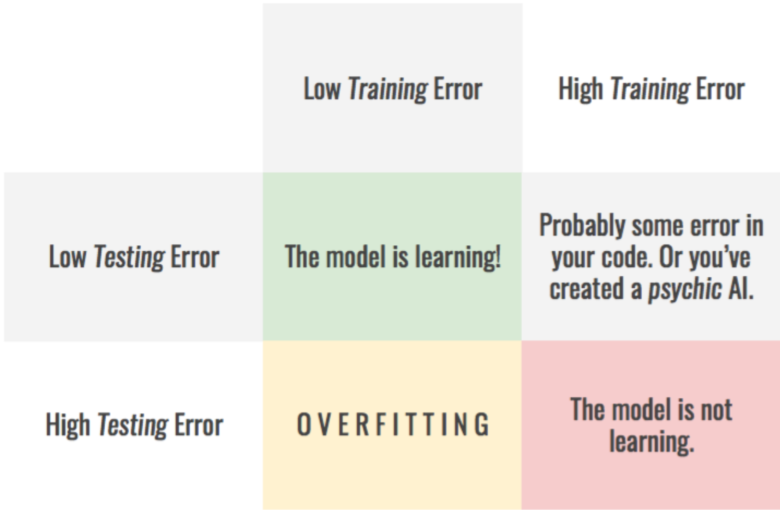
The tricky part is that, at first glance, it **may seem** that your model is performing well because it has a very **small error** on the *training* data. However, as soon as you ask it to **predict new data points,** it will **fail**.

How to detect overfitting

As stated above, overfitting is characterized by the **inability** of the model **to generalize**. To test this ability, a simple method consists in splitting the dataset into two parts: the **training set** and the **test set.** *When selecting models, you might want to split the dataset in three, I explain why here*.

1. The *training* set represents about **80%** of the *available* data, and is used to train the model (you don't say?!).
2. The *test* set consists of the remaining **20%** of the dataset, and is used to *test* the **accuracy** of the model on data it has **never seen before**.

With this split we can check the performance of the model on **each set** to gain insight on **how** the *training* process is going, and spot *overfitting* when it happens. *This table* shows the different cases.



	Low *Training* Error	High *Training* Error
Low *Testing* Error	The model is learning!	Probably some error in your code. Or you've created a *psychic* AI.
High *Testing* Error	OVERFITTING	The model is not learning.

Overfitting can be seen as the **difference** between the **training** and **testing** error.

Note: for this technique to work, you need to make sure both parts are **representative** of your data. A *good practice* is to **shuffle** the order of the dataset before *splitting*.

Overfitting can be pretty *discouraging* because it **raises** your **hopes** just before *brutally crushing* them. Fortunately, there are a few tricks to **prevent** it from happening.

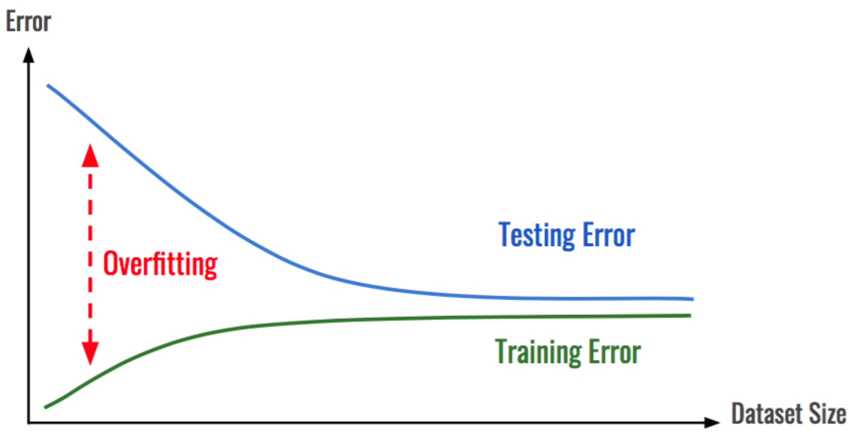
How to prevent overfitting - Model & Data

First, we can try to look at the *components* of our system to find solutions. This means changing *data* we are using, or which *model*.

Gather more data

You model can only *store* so much information. This means that the **more training data** you feed it, the **less likely** it is to **overfit**. The reason is that, as you **add** more **data**, the model becomes **unable** to **overfit** all the samples, and is **forced** to **generalize** to make progress.

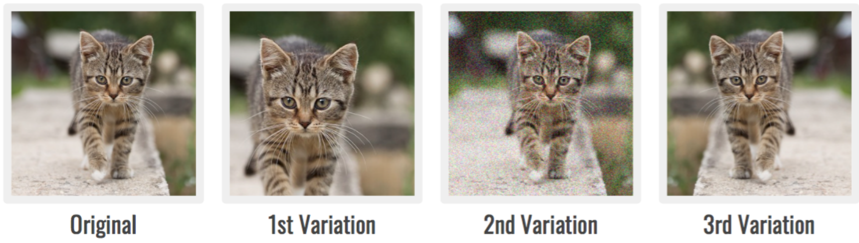
Collecting more examples should be the *first step* in every data science task, as more data will result in an *increased accuracy* of the model, while reducing the chance of *overfitting*.



The **more data** you get, the **less** likely the model is to **overfit.**

Data augmentation & Noise

Collecting more data is a *tedious* and **expensive** process. If you can't do it, you should try to make your data *appear* as if it was **more diverse**. To do that, use data augmentation techniques so that each time a sample is processed by the model, it's slightly different from the previous time. This will make it **harder** for the model to *learn parameters* for each sample.



Each iteration sees as **different variation** of the original sample.

Another good practice is to add **noise:**

- **To the input**: This serves the same purpose as data augmentation, but will also work toward making the **model robust** to *natural perturbations* it could encounter **in the wild**.
- **To the output**: Again, this will make the training more diversified.

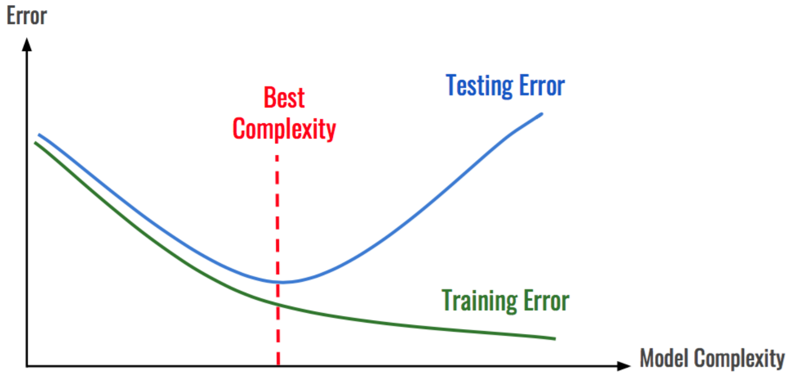
Note: In both cases, you need to make sure that the **magnitude of the noise** is not too *great*. Otherwise, you could end up respectively *drowning* the information of the input in the noise*, or* make the output *incorrect.* Both will hinder the training process.

Simplify the model

If, even with all the data you now have, your model *still* manages to overfit your training dataset, it may be that the model is **too powerful**. You could then try to **reduce the complexity** of the model.

As stated previously, a model can only overfit *that much* data. By progressively reducing its complexity—*# of **estimators** in a **random forest**, # of **parameters** in a **neural network** etc.*—you can make the model *simple* enough that it *doesn't overfit,* but *complex* enough to *learn* from your data. To do that, it's convenient to look at the **error** on **both datasets** depending on the model complexity.

This also has the advantage of making the model **lighter**, **train faster** and **run faster**.



On the left, the model is too simple. On the right it overfits.

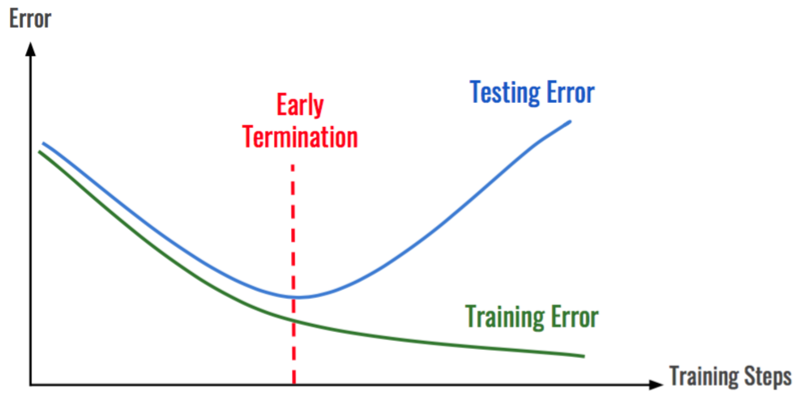
. . .

How to prevent overfitting - Training Process

A *second* possibility it to change the way the **training** is done. This includes altering the **loss function,** or the way the model *functions* during training.

Early Termination

In most cases, the model **starts** by learning a correct distribution of the data, and, at some point, starts to overfit the data. By identifying the *moment* where this **shift occurs**, you can **stop the learning process** *before* the overfitting happens. As before, this is done by looking at the *training error* over time.



When the **testing error** starts to **increase**, it's time to stop!

How to prevent overfitting— Regularization

Regularization is a process of **constraining** the **learning** of the model to *reduce* **overfitting**. It can take many different forms, and we will see a couple of them.

L1 and L2 regularization

One of the most *powerful* and well-known technique of regularization is to **add a penalty** to the **loss function**. The most common are called *L1* and *L2*:

1. The **L1 penalty** aims to minimize the **absolute value** of the weights
2. The **L2 penalty** aims to minimize the **squared magnitude** of the weights.

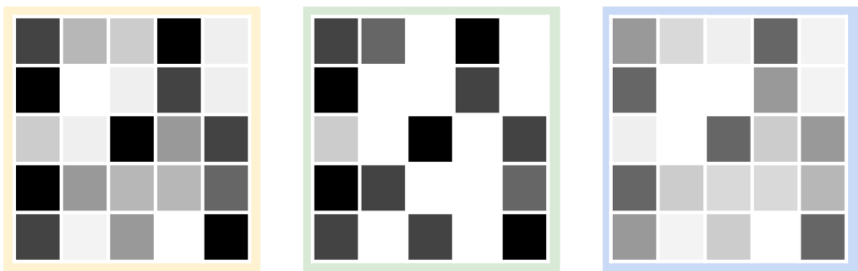
$$\lambda \sum_{j=0}^{M} |W_j| \qquad \lambda \sum_{j=0}^{M} W_j^2$$

L1 Penalty L2 Penalty

With the penalty, the model is forced to *make compromises* on its weights, as it can no longer make them **arbitrarily large**. This makes the model **more general**, which helps combat overfitting.

The *L1* penalty has the added advantage that it enforces **feature selection**, which means that it has a tendency to set to 0 the *less useful* parameters. This helps identify the **most relevant features** in a *dataset*. The downside is that it is often **not** as **computationally efficient** as the *L2* penalty.

Here is what the weight matrixes would look like. Note how the **L1** matrix is **sparse** with many zeros, and the **L2** matrix has *slightly* **smaller weights**.



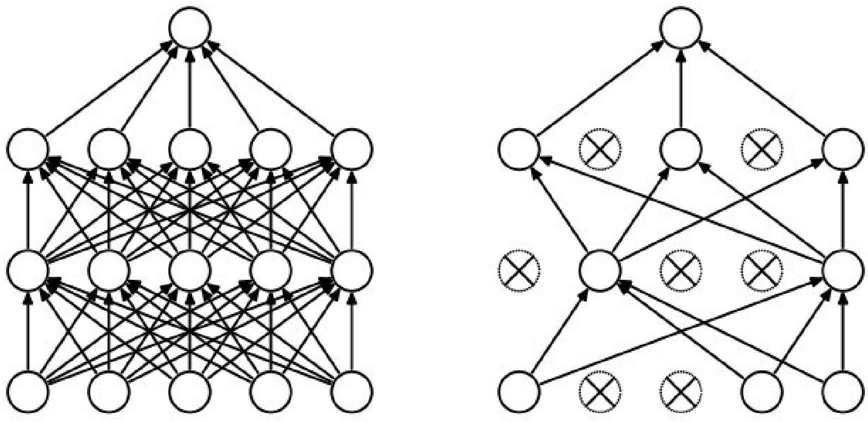
Baseline L1 Regularization L2 Regularization

Another possibility is to add noise to the *parameters* during the training, which helps **generalization**.

#6

For Deep Learning: Dropout and Dropconnect

This **extremely effective** technique is specific to **Deep Learning,** as it relies on the fact that *neural networks* process the information from one **layer** to the next. The idea is to randomly deactivate either **neurons** (*dropout*) or **connections** (*dropconnect*) during the training.



(a) Standard Neural Net (b) After applying dropout.

This forces the network to become **redundant**, as it can no longer **rely** on *specific* **neurons** or **connections** to extract *specific* **features**. Once the training is done, all neuraons and connections are restored. It has been shown that this technique is *somewhat equivalent* to having an **ensemble** approach, which **favorises generalization,** thus reducing overfitting.

Conclusion

As you know by now, overfitting is one of the main issues the *Data Scientist* has to face. It can be a *real pain* to deal with if you don't know how to *stop* it. With the techniques presented in this article, you should now be able to *prevent* your models from **cheating** the learning process, and get the **results** you deserve!

🎉 You've reached the end! I hope you enjoyed this article. If you did, feel free to like it, share it, explain it to your cat, follow me on medium, or do whatever you feel like doing! 🎉

> ***If you like Data Science and Artificial Intelligence, subscribe to the newsletter to receive updates on articles and much more!***