Chapter 2: Main Challenges of Machine Learning

In machine learning, challenges originate from two potential sources – 'bad data' and 'bad algorithm'.

2.1. Challenges from 'Bad Data'

2.1.1. Insufficient Quantity of Training Data

It takes a lot of data for most Machine Learning algorithms to work properly. Even for very simple problems, we typically need thousands of examples, and for complex problems such as image or speech recognition, we may need millions of examples (unless we can reuse parts of an existing model).

The Unreasonable Effectiveness of Data

In a famous paper published in 2001, Microsoft researchers Michele Banko and Eric Brill showed that very different Machine Learning algorithms, including fairly simple ones, performed almost identically well on a complex problem of natural language disambiguation[8] once they were given enough data (as you can see in Figure 1-20).

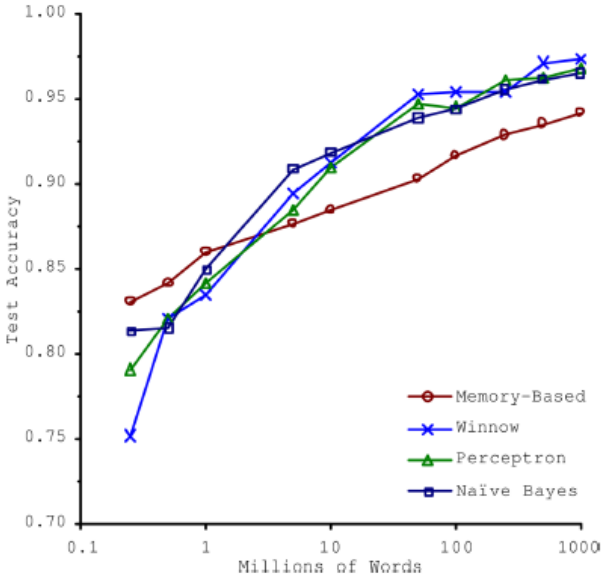


Figure 1-20. The importance of data versus algorithms[9]

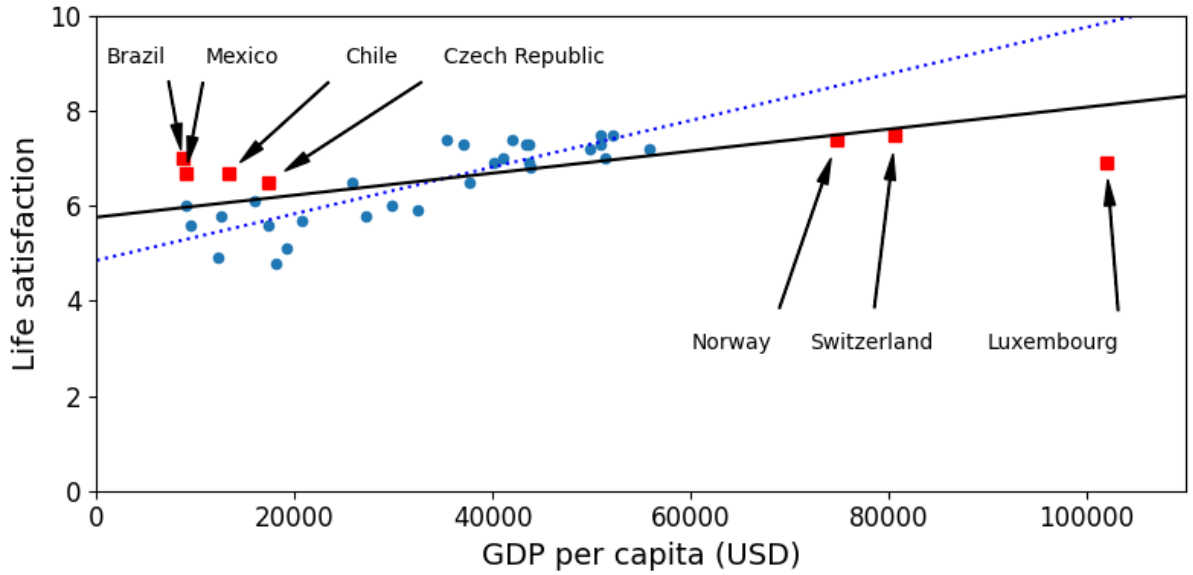
As the authors put it: "these results suggest that we may want to reconsider the trade-off between spending time and money on algorithm development versus spending it on corpus development."

The idea that data matters more than algorithms for complex problems was further popularized by Peter Norvig et al. in a paper titled "The Unreasonable Effectiveness of Data" published in 2009.[10] It should be noted, however, that small- and medium-sized datasets are still very common, and it is not always easy or cheap to get extra training data, so don't abandon algorithms just yet.

2.1.2. Nonrepresentative Training Data

Irrespective of whether you use instance-based learning or model-based learning, it is crucial to ensure that the training data be representative of the new cases we want to generalize to.

For example, the set of countries we used earlier for training the linear model ("`01_the_machine_learning_landscape.ipynb`") was not perfectly representative; a few countries were missing. Below figure shows what the data looks like when you add the missing countries.



If we train a linear model on this data, you get the solid line, while the old model is represented by the dotted line. As we can see, not only does adding a few missing countries significantly alter the model, but it makes it clear that such a simple linear model is probably never going to work well. It seems that very rich countries are not happier than moderately rich countries (in fact they seem unhappier), and conversely some poor countries seem happier than many rich countries.

By using a nonrepresentative training set, we trained a model that is unlikely to make accurate predictions, especially for very poor and very rich countries.

It is crucial to use a training set that is representative of the cases you want to generalize to. This is often harder than it sounds: if the sample is too small, you will have *sampling noise* (i.e., nonrepresentative data as a result of chance), but even very large samples can be nonrepresentative if the sampling method is flawed. This is called *sampling bias*.

A Famous Example of Sampling Bias

Perhaps the most famous example of sampling bias happened during the US presidential election in 1936, which pitted Landon against Roosevelt: The Literary Digest conducted a very large poll, sending mail to about 10 million people. It got 2.4 million answers, and predicted with high confidence that Landon would get 57% of the votes. Instead, Roosevelt won with 62% of the votes. The flaw was in the Literary Digest's sampling method:

- First, to obtain the addresses to send the polls to, the *Literary Digest* used telephone directories, lists of magazine subscribers, club membership lists, and the like. All of these lists tend to favour wealthier people, who are more likely to vote Republican (hence Landon).
- Second, less than 25% of the people who received the poll answered. Again, this introduces a *sampling bias*, by ruling out people who don't care much about politics, people who don't like the Literary Digest, and other key groups. This is a special type of sampling bias called *nonresponse bias*.

2.1.3. Poor-Quality Data

If the training data is full of errors, outliers, and noise (e.g., due to poor quality measurements), the system is less likely to detect the underlying patterns, resulting in erroneous performance.

Therefore, we need to spend significant amount of time to clean and prepare the data:

- If some instances are clearly outliers, simply discarding them or trying to fix the errors manually may be helpful.
- If some instances are missing a few features (e.g., 5% of customers did not specify their age), we can either:
    - Ignore this attribute altogether,
    - Ignore these instances,
    - Fill in the missing values (e.g., with the median age), or
    - Train one model with the feature and one model without it, and so on.

2.1.4. Irrelevant Features

A machine learning algorithm will be able to learn only if the training data contains enough relevant features and not too many irrelevant ones. So, we need to come up with a good set of features for the machine learning model to train on. This process is called *feature engineering*. It involves:

- *Feature selection*: selecting the most useful features to train on among existing features.
- *Feature extraction*: combining existing features to produce a more useful one (as we saw earlier, dimensionality reduction algorithms can help).
- Creating new features by gathering new data.

Now we will look at a couple of challenges originating from "bad algorithms".

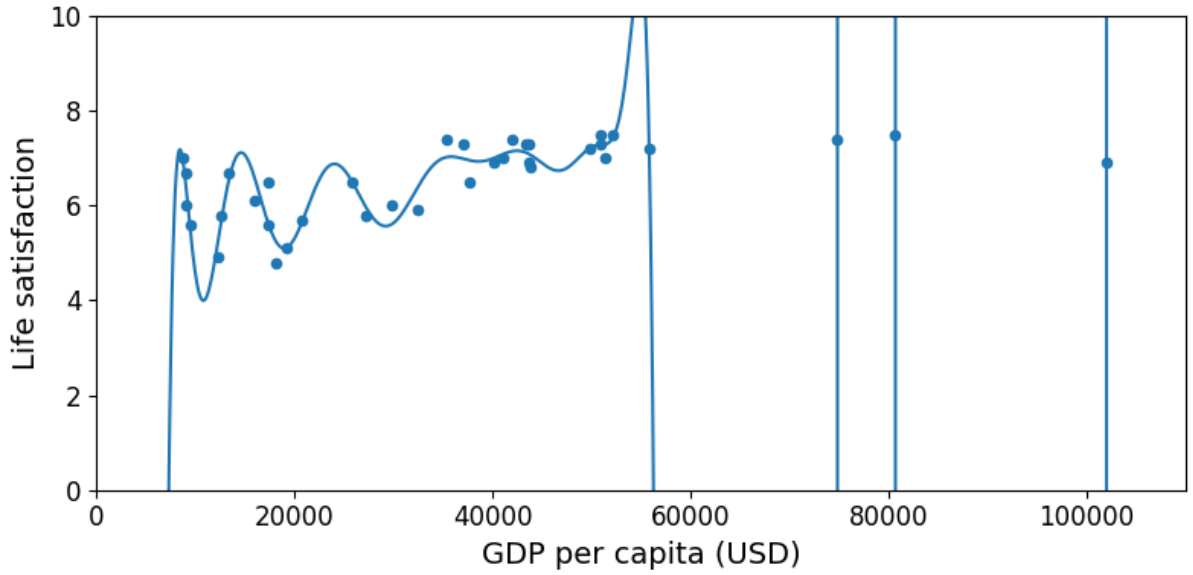
2.2. Challenges from 'Bad Algorithm'

2.2.1. Overfitting the Training Data

Overgeneralizing is something that we humans do all too often, and unfortunately machines can fall into the same trap as well, if we are not careful.

In Machine Learning this is called *overfitting* happens when the model performs well on the training data, but it does not generalize well to the new unseen data.

Below figure shows an example of a high-degree polynomial life satisfaction model that strongly overfits the training data. Even though it performs much better on the training data than the simple linear model, we cannot trust its predictions for new unseen data.



Complex models such as deep neural networks can detect subtle patterns in the data, but if the training set is noisy, or if it is too small (which introduces sampling noise), then the model is likely to detect patterns in the noise itself. This is *overfitting*. Obviously, these patterns will not generalize to new instances.

Overfitting happens when the model is too complex relative to the amount and noisiness of the training data. The possible solutions are:

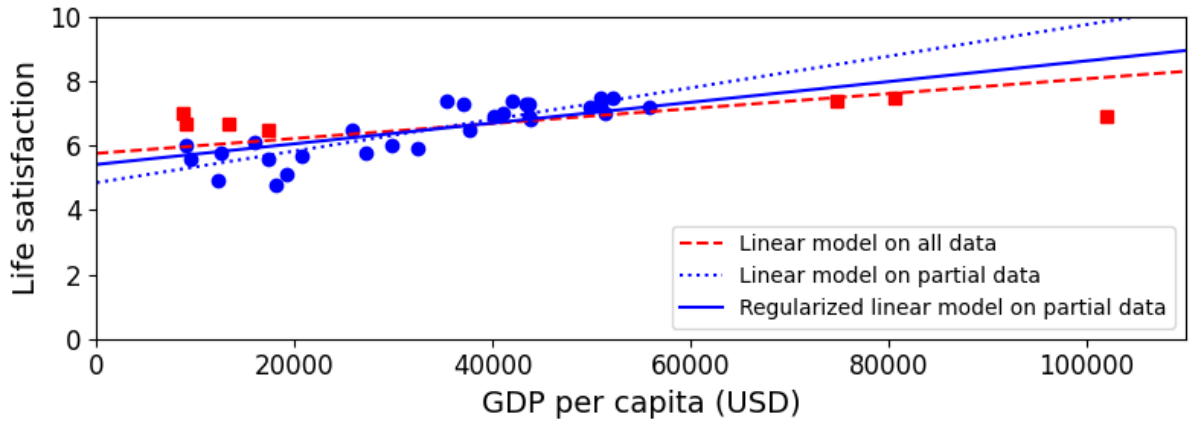
- To simplify the model by selecting one with fewer parameters (e.g., a linear model rather than a high-degree polynomial model), by reducing the number of attributes in the training data or by constraining the model
- To gather more training data
- To reduce the noise in the training data (e.g., fix data errors and remove outliers)

Constraining a model to make it simpler and reduce the risk of overfitting is called ***regularization*.**

For example, the linear model we defined earlier has two parameters, θ_0 and θ_1. This gives the learning algorithm two degrees of freedom to adapt the model to the training data: it can tweak both the height (θ_0) and the slope (θ_1) of the line. If we forced $\theta_1 = 0$, the algorithm would have only one degree of freedom and would have a much harder time fitting the data properly: all it could do is move the line up or down to get as close as possible to the training instances, so it would end up around the mean.

If we allow the algorithm to modify θ_1 but we force it to keep it small, then the learning algorithm will effectively have somewhere in between one and two degrees of freedom. It will produce a simpler model than with two degrees of freedom, but more complex than with just one. You want

to find the right balance between fitting the training data perfectly and keeping the model simple enough to ensure that it will generalize well.



From the above figure, we can see that regularization forced the model to have a smaller slope, which fits a bit less the training data that the model was trained on, but actually allows it to generalize better to new examples.

The amount of regularization to apply during learning can be controlled by a *hyperparameter*, a parameter of a learning algorithm (not of the model). That means, it is not affected by the learning algorithm itself; it must be set prior to training and remains constant during training.

If the regularization hyperparameter is set to a very large value, we will get an almost flat model (a slope close to zero); the learning algorithm will almost certainly not overfit the training data, but it will be less likely to find a good solution.

Tuning hyperparameters is an important part of building a Machine Learning system.

2.2.2. Underfitting the Training Data

Underfitting is the opposite of overfitting: **it occurs when your model is too simple to learn the underlying structure of the data**.

For example, a linear model of life satisfaction is prone to underfit; reality is just more complex than the model, so its predictions are bound to be inaccurate, even on the training examples. The possible solutions to underfitting are:

- Selecting a more powerful model, with more parameters (more complex model)
- Feeding better features to the learning algorithm (feature engineering)
- Reducing the constraints on the model (e.g., reducing the regularization hyperparameter)

The machine learning system will not perform well if your training set is too small, or if the data is not representative, noisy, or polluted with irrelevant features (garbage in, garbage out). Lastly, your model needs to be neither too simple (in which case it will underfit) nor too complex (in which case it will overfit).