**Machine Learning Overview**

**Learning Objectives**

* Explore the fundamentals of machine learning and gain an understanding of the taxonomy of different types of ML algorithms
* Develop an understanding of best practices and common challenges that data scientists deal with when working on machine learning applications

**Work to Complete**

In this unit, you'll:

* Work through a series of introductory resources. This unit is intended to introduce you to machine learning fundamentals.
* Complete a quiz to test your understanding of the topics introduced in the unit.

Machine learning (ML) algorithms enable computers to learn from data, and even improve themselves, without being explicitly programmed. This unit takes a thorough look at the different facets of this practice and lays the foundation that you'll build on your work through subsequent units about supervised learning, unsupervised learning, and advanced ML practices. In addition to learning about the fundamentals of machine learning, you'll also become familiar with different ML use-cases and explore the Python libraries and packages most relevant to machine learning work. Let's get started!

**Introduction to Machine Learning**

When it comes to the taxonomy of types of machine learning, a good account breaks up methods across six dimensions:

* Supervised vs Unsupervised learning
* Batch vs Online learning
* Instance-Based vs Model-based learning

This subunit offers up an introduction to machine learning and explores each of these topics.

Machine learning has been much buzzed about in the last few years. Go beyond the hype by reading this article, which offers a thoughtful introduction to this exciting and nuanced topic.

[[](https://medium.com/@NotAyushXD?source=post_page-----eed6024fdb08--------------------------------)](https://medium.com/@NotAyushXD?source=post_page-----eed6024fdb08--------------------------------)

[Ayush Pant](https://medium.com/@NotAyushXD?source=post_page-----eed6024fdb08--------------------------------)

Jan 7, 2019

·

6 min read

·

# Introduction to Machine Learning for Beginners

Diagram

Description automatically generated

# Introduction to Machine Learning for Beginners

We have seen Machine Learning as a buzzword for the past few years, the reason for this might be the high amount of data production by applications, the increase of computation power in the past few years and the development of better algorithms.

Machine Learning is used anywhere from automating mundane tasks to offering intelligent insights, industries in every sector try to benefit from it. You may already be using a device that utilizes it. For example, a wearable fitness tracker like Fitbit, or an intelligent home assistant like Google Home. But there are much more examples of ML in use.

* Prediction — Machine learning can also be used in the prediction systems. Considering the loan example, to compute the probability of a fault, the system will need to classify the available data in groups.
* Image recognition — Machine learning can be used for face detection in an image as well. There is a separate category for each person in a database of several people.
* Speech Recognition — It is the translation of spoken words into the text. It is used in voice searches and more. Voice user interfaces include voice dialing, call routing, and appliance control. It can also be used a simple data entry and the preparation of structured documents.
* Medical diagnoses — ML is trained to recognize cancerous tissues.
* Financial industry and trading — companies use ML in fraud investigations and credit checks.

# A Quick History of Machine Learning

A picture containing graphical user interface

Description automatically generated

Image: Linked In | Machine Learning vs Deep learning

It was in the 1940s when the first manually operated computer system, ENIAC (Electronic Numerical Integrator and Computer), was invented. At that time the word “computer” was being used as a name for a human with intensive numerical computation capabilities, so, ENIAC was called a numerical computing machine! Well, you may say it has nothing to do with learning?! WRONG, from the beginning the idea was to build a machine able to emulate human thinking and learning.



EIMC — Electronic Numerical Integrator and Computer | Image: www.computerhistory.org

In the 1950s, we see the first computer game program claiming to be able to beat the checkers world champion. This program helped checkers players a lot in improving their skills! Around the same time, Frank Rosenblatt invented the Perceptron which was a very, very simple classifier but when it was combined in large numbers, in a network, it became a powerful monster. Well, the monster is relative to the time and in that time, it was a real breakthrough. Then we see several years of stagnation of the neural network field due to its difficulties in solving certain problems.

Thanks to statistics, machine learning became very famous in the 1990s. The intersection of computer science and statistics gave birth to probabilistic approaches in AI. This shifted the field further toward data-driven approaches. Having large-scale data available, scientists started to build intelligent systems that were able to analyze and learn from large amounts of data. As a highlight, IBM’s Deep Blue system beat the world champion of chess, the grand-master Garry Kasparov. Yeah, I know Kasparov accused IBM of cheating, but this is a piece of history now and Deep Blue is resting peacefully in a museum.

# What is Machine Learning?

A picture containing text, clipart

Description automatically generated

According to Arthur Samuel, Machine Learning algorithms enable the computers to learn from data, and even improve themselves, without being explicitly programmed.

Machine learning (ML) is a category of an algorithm that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available.

# Types of Machine Learning?

Machine learning can be classified into 3 types of algorithms.

1. Supervised Learning — [Link coming soon in a future blog]
2. Unsupervised Learning — [Link coming soon in a future blog]
3. Reinforcement Learning — [Link coming soon in a future blog]

Diagram

Description automatically generated

3 Types of Learning

# Overview of Supervised Learning Algorithm

In Supervised learning, an AI system is presented with data which is labeled, which means that each data tagged with the correct label.

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

Graphical user interface, application, timeline

Description automatically generated

Example of Supervised Learning

As shown in the above example, we have initially taken some data and marked them as ‘Spam’ or ‘Not Spam’. This labeled data is used by the training supervised model, this data is used to train the model.

Once it is trained we can test our model by testing it with some test new mails and checking of the model is able to predict the right output.

## Types of Supervised learning

* **Classification**: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
* **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

# Overview of Unsupervised Learning Algorithm

In unsupervised learning, an AI system is presented with unlabeled, uncategorized data and the system’s algorithms act on the data without prior training. The output is dependent upon the coded algorithms. Subjecting a system to unsupervised learning is one way of testing AI.

A picture containing graphical user interface

Description automatically generated

Example of Unsupervised Learning

In the above example, we have given some characters to our model which are ‘Ducks’ and ‘Not Ducks’. In our training data, we don’t provide any label to the corresponding data. The unsupervised model is able to separate both the characters by looking at the type of data and models the underlying structure or distribution in the data in order to learn more about it.

## Types of Unsupervised learning

* **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
* **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

# Overview of Reinforcement Learning

A reinforcement learning algorithm, or agent, learns by interacting with its environment. The agent receives rewards by performing correctly and penalties for performing incorrectly. The agent learns without intervention from a human by maximizing its reward and minimizing its penalty. It is a type of dynamic programming that trains algorithms using a system of reward and punishment.

Diagram

Description automatically generated

Example of Reinforcement Learning

In the above example, we can see that the agent is given 2 options i.e. a path with water or a path with fire. A reinforcement algorithm works on reward a system i.e. if the agent uses the fire path then the rewards are subtracted and agent tries to learn that it should avoid the fire path. If it had chosen the water path or the safe path then some points would have been added to the reward points, the agent then would try to learn what path is safe and what path isn’t.

It is basically leveraging the rewards obtained, the agent improves its environment knowledge to select the next action.

# Summary

In this blog, I have presented you with the basics concepts of Machine learning and I hope this blog was helpful and would have motivated you enough to get interested in the topic.

In a supervised learning model, an algorithm is given a labeled training dataset and evaluates its ability to understand the data using an answer key. An unsupervised learning model features an algorithm that tries to make sense of an unlabeled dataset on its own without training. Learn more by working through this resource.

[[](https://medium.com/@devins?source=post_page-----14f68e32ea8d--------------------------------)](https://medium.com/@devins?source=post_page-----14f68e32ea8d--------------------------------)

[Devin Soni](https://medium.com/@devins?source=post_page-----14f68e32ea8d--------------------------------)

Mar 22, 2018

# Supervised vs. Unsupervised Learning

## Understanding the differences between the two main types of machine learning methods

Within the field of machine learning, there are two main types of tasks: supervised, and unsupervised. The main difference between the two types is that supervised learning is done using a **ground truth**, or in other words, we have prior knowledge of what the output values for our samples should be. Therefore, the goal of supervised learning is to learn a function that, given a sample of data and desired outputs, best approximates the relationship between input and output observable in the data. Unsupervised learning, on the other hand, does not have labeled outputs, so its goal is to infer the natural structure present within a set of data points.

## Supervised Learning

Chart, scatter chart

Description automatically generated

Supervised learning is typically done in the context of classification, when we want to map input to output labels, or regression, when we want to map input to a continuous output. Common algorithms in supervised learning include logistic regression, naive bayes, support vector machines, artificial neural networks, and random forests. In both regression and classification, the goal is to find specific relationships or structure in the input data that allow us to effectively produce correct output data. Note that “correct” output is determined entirely from the training data, so while we do have a ground truth that our model will assume is true, it is not to say that data labels are always correct in real-world situations. Noisy, or incorrect, data labels will clearly reduce the effectiveness of your model.

When conducting supervised learning, the main considerations are model complexity, and the bias-variance tradeoff. Note that both of these are interrelated.

Model complexity refers to the complexity of the function you are attempting to learn — similar to the degree of a polynomial. The proper level of model complexity is generally determined by the nature of your training data. If you have a small amount of data, or if your data is not uniformly spread throughout different possible scenarios, you should opt for a low-complexity model. This is because a high-complexity model will **overfit** if used on a small number of data points. Overfitting refers to learning a function that fits your training data very well, but does not **generalize** to other data points — in other words, you are strictly learning to produce your training data without learning the actual trend or structure in the data that leads to this output. Imagine trying to fit a curve between 2 points. In theory, you can use a function of any degree, but in practice, you would parsimoniously add complexity, and go with a linear function.

The bias-variance tradeoff also relates to model generalization. In any model, there is a balance between bias, which is the constant error term, and variance, which is the amount by which the error may vary between different training sets. So, high bias and low variance would be a model that is consistently wrong 20% of the time, whereas a low bias and high variance model would be a model that can be wrong anywhere from 5%-50% of the time, depending on the data used to train it. Note that bias and variance typically move in opposite directions of each other; increasing bias will usually lead to lower variance, and vice versa. When making your model, your specific problem and the nature of your data should allow you to make an informed decision on where to fall on the bias-variance spectrum. Generally, increasing bias (and decreasing variance) results in models with relatively guaranteed baseline levels of performance, which may be critical in certain tasks. Additionally, in order to produce models that generalize well, the variance of your model should scale with the size and complexity of your training data — small, simple data-sets should usually be learned with low-variance models, and large, complex data-sets will often require higher-variance models to fully learn the structure of the data.

## Unsupervised Learning

A picture containing graphical user interface

Description automatically generated

The most common tasks within unsupervised learning are clustering, representation learning, and density estimation. In all of these cases, we wish to learn the inherent structure of our data without using explicitly-provided labels. Some common algorithms include k-means clustering, principal component analysis, and autoencoders. Since no labels are provided, there is no specific way to compare model performance in most unsupervised learning methods.

Two common use-cases for unsupervised learning are exploratory analysis and dimensionality reduction.

Unsupervised learning is very useful in exploratory analysis because it can automatically identify structure in data. For example, if an analyst were trying to segment consumers, unsupervised clustering methods would be a great starting point for their analysis. In situations where it is either impossible or impractical for a human to propose trends in the data, unsupervised learning can provide initial insights that can then be used to test individual hypotheses.

Dimensionality reduction, which refers to the methods used to represent data using less columns or features, can be accomplished through unsupervised methods. In representation learning, we wish to learn relationships between individual features, allowing us to represent our data using the latent features that interrelate our initial features. This sparse latent structure is often represented using far fewer features than we started with, so it can make further data processing much less intensive, and can eliminate redundant features.

## TLDR:

Table

Description automatically generated

Data Science Career Track

Batch Learning vs Online Learning  
Batch Learning

An ML algorithm performs Batch Learning if the system can't learn incrementally, and  
must be trained using all the available data. Since this takes both time and computation  
power, the learning process typically occurs offline (this is known as 'offline learning').  
The system is trained and then launched; it doesn't continue to learn after it has  
launched. If new data is acquired, a new version of the system needs to be trained to  
replace the predecessor. Data scientists can automate the training, evaluation, and  
launch of ML systems that use Batch Learning.  
There are shortcomings to an automated Batch approach. Often, you'll need your ML  
algorithm to respond to changes in vast quantities of data in real-time. It can be quite  
costly to make a fresh Batch system from this data every day.  
Online Learning  
Online Learning, on the other hand, trains the system by breaking the data up into small  
groups and feeding the system those groups over a longer period of time. The learning  
is broken up into individually cheap and fast steps, which allows for receiving large  
amounts of data in real-time. Online Learning systems can also adapt quickly, even if  
This document is authorized for use only by Charito Mascio (charrie@k5ryu.com). Copying or posting is an infringement of  
copyright.

one has limited computational resources. You can set the learning rate (i.e; the speed  
with which the system adapts to changing data) yourself, but you need to be judicious.  
If you make this value too high, your system will quickly adapt to the new data at the  
expense of learning done on previous data; if you set it too low, it will learn about the  
new data too slowly to be effective.  
We must also be careful about the possibility, with Online Learning, that the system's  
performance is slowed by low-quality data being fed to it unbeknownst to us. Whereas  
with Batch Learning, you have a relatively high amount of control over the quality of the  
data your system learns on, with Online Learning, you snooze, you lose (or your model  
does)!

This document is authorized for use only by Charito Mascio (charrie@k5ryu.com). Copying or posting is an infringement of  
copyright.

[[](https://medium.com/@maxpagels?source=post_page-----515556ff72c5--------------------------------)](https://medium.com/@maxpagels?source=post_page-----515556ff72c5--------------------------------)

[Max Pagels](https://medium.com/@maxpagels?source=post_page-----515556ff72c5--------------------------------)

Apr 20, 2018

# What is Online Machine Learning?

During the start of my career, I was fortunate enough to work on a subfield of machine learning known as **online learning** (also known as **incremental** or **out-of-core learning**). Compared to “traditional” ML solutions, online learning is a fundamentally different approach, one that embraces the fact that learning environments can (and do) change from second to second. It’s tricky to get right, but when applied correctly, the results you can achieve with online learning are nothing short of remarkable. In this post, I’ll give a quick introduction to the technique.

Update 27/09/2019: lots of people have asked if there exist any purpose-built incremental learning libraries. Yes! [Vowpal Wabbit](https://github.com/VowpalWabbit/vowpal_wabbit) is extremely powerful, and has been around for quite a while. For those that prefer scikit-inspired APIs, take a look at [creme](https://creme-ml.github.io).

Our <insert name of AI product> gets better the more you use it!

Ever read about about some AI product or platform that claims the above, promising to get better over time? In the vast majority of cases, what’s actually under the bonnet is, well, a bit underwhelming.

In a general sense, you need two things for machine learning: data and a suitable learning algorithm. The learning algorithm learns from/trains on your data and produces a (hopefully) accurate model, typically used for prediction on new data. I’m oversimplifying things, but that’s the core idea.

ML models, save for some exceptions, are static things. They are essentially collections of parameters. After you’ve trained a model, its parameters don’t change. From a technical perspective, that’s good news: if you want to serve predictions over an API, you can instantiate several instances of a model, place a load balancer on top of them, and pop round the pub for a pint to congratulate yourself on a job well done. Since model parameters don’t change, you don’t need to synchronise between model instances. It’s horizontally scalable, almost trivially so. And if we’re being honest, horizontally scalable is the best type of scalable.

But what about new data? If we just train a model once and never touch it again, we’re missing out the information more data could provide us. This is especially important in environments where behaviours change quickly. Online shopping is one such environment: a product that is popular today may be all but forgotten tomorrow.

In order to react to new data and make an AI that learns over time, ML practitioners typically do one of two things:

1. They manually train on newer data, and deploy the resulting model once they are happy with its performance
2. They schedule training on new data to take place, say, once a week and automatically deploy the resulting model

99,99 per cent of the time, when someone claims “our AI gets better the more you use it”, what they really mean is that they’ve gone for approach 2), scheduling the training of new models. In its most simple form, this could very literally be one line in a crontab file.

All this is well and good, apart from one glaring problem: even if you train new models each week, or even each day, you’re still lagging behind. Your model is never fully up-to-date with current events, because it’s trained on stale data. Ideally, what you want is **a model that can learn from new examples in something close to real time**. Not only predict in real time, but learn in real time, too.

I love using scikit-learn to play around with ML. All of the different algorithms in scikit-learn implement the same simple API, making it very easy to get up and running. For regression problems, I usually start with the SGDRegressor class. Here’s how to train a simple model on dummy data (taken straight from the scikit documentation):

import numpy as np  
from sklearn import linear\_model  
n\_samples, n\_features = 10, 5  
y = np.random.randn(n\_samples)  
X = np.random.randn(n\_samples, n\_features)  
clf = linear\_model.SGDRegressor()  
clf.fit(X, y)

The fit() method does all the training magic, resulting in a model we can use for prediction (in this case, predicting on one example):

clf.predict(np.random.randn(1, n\_features))

In addition to the fit()method, the SGDRegressor also provides a partial\_fit() method, so that you can incrementally train on small batches of data. In fact, all learning algorithms that are compatible with standard optimisation algorithms like (stochastic) gradient decent, adam, RMSprop, and so on have this capability.

Out of curiosity, let’s see how long it takes to train on a single example using partial\_fit():

import numpy as np  
from sklearn import linear\_modeln\_samples, n\_features = 1, 500  
y = np.random.randn(n\_samples)  
X = np.random.randn(n\_samples, n\_features)  
clf = linear\_model.SGDRegressor()import timestart\_time = time.time()  
clf.partial\_fit(X, y)  
elapsed\_time = time.time() - start\_time  
print(elapsed\_time)>>> 0.0008502006530761719

0.0009 seconds on my machine. That’s quite fast. In fact, if we were to put our SGDRegressor behind a REST API and train on an example each time an HTTP request was made, factoring in, say, 10ms for request processing, we could handle about 520 requests a second, or about 45 million requests a day.

This prompts an interesting question: given these numbers, would it be possible to learn from new examples in something close to real time? **Therein lies the potential of online learning: the second we see a new example, let’s learn from it as fast as we can.** The faster, the better. In fact, because speed trumps everything else in online learning, we typically use simple learning algorithms over complex ones like neural networks. We strive for millisecond-level learning; everything else comes second.

ML purists might scoff at the idea of online algorithms for real time learning. Training a model can go wrong in lots of different ways: the algorithm itself might not be suitable, the model might fail to generalise well, the learning rate might be wrong, the regularisation might be too low or too high…the list goes on. Why on earth would we even attempt to learn immediately when there are no guarantees on what might happen?

The answer is simple: no matter how good a model is, or how much data you feed it, a model is still an imperfect representation of an environment. To make the best possible decisions right now, we can’t afford to have a model that only knows about things that happened yesterday.

Consider the following example. Let’s say we run a news website. We personalise our news by collecting data on what was clicked or not clicked, and by whom. Based on this information, we predict the types of news different visitors might like, and serve them relevant items.

One day, out of the blue, word gets out that the government is issuing a state of emergency, and will hold a press conference in an hour. Suddenly, everyone is interested in domestic affairs — even those who typically only read about sports or look at the funnies. When presented with a news piece about the conference, a huge percentage of the audience clicks it to learn more.

If you had gone the traditional route and batch trained your recommendation engine once a day, it would still be stuck offering the same type of content, even though the underlying world changed dramatically¹. You should be serving up domestic news right now, but aren’t because your system is too slow.

It gets worse: the following day, after the press conference and following a new training cycle, your engine would start actively recommending domestic news which, after 24 hours, isn’t necessarily interesting any more. It’s made two mistakes, both because it can’t react fast enough.

That’s the power of online learning: done properly, it can react in minutes or even seconds. With it, there is no such thing as “yesterday’s news”.

Bolding for emphasis, **implementing real time learning isn’t easy**. If you place some learning algorithm behind an API and, god forbid, open it up to the Internet, there’s an almost limitless number of ways it can go wrong. You might get lots of feedback (examples) from one thing but not another, leading to a skewed classes problem. You might’ve set your learning rate too high, causing your model to forget everything that happened more than a second ago. You might overfit, or underfit. Someone might DDoS your system, screwing up learning in the process. Online learning is prone to [catastrophic interference](https://en.wikipedia.org/wiki/Catastrophic_interference) — more so than most other techniques.

Online learning also requires an entirely different approach in terms of technical architecture. Since a model can, and will, change from second to second, you can’t just instantiate several instances like you can with traditional techniques. It’s not horizontally scalable. Instead, you are forced to have a single model instance that eats new data as fast as it can, spitting out sets of learned parameters behind an API. And the second that one set in one process gets replaced by a new one, all other processes must follow suit immediately. It’s an engineering challenge, because the most important part (the model) is only vertically scalable. It may not even be feasible to distribute between threads.

Learning immediately also requires fast access to new data. If you’re lucky enough to get all the data you need for a single training example as part of an API call, you’re good to go. But if something is not available client-side, you need to be able to grab that data from somewhere in milliseconds. Typically, that means using an in-memory store like Redis. “Big data” processing frameworks aren’t of much help. If you want to to both batch and online learning, Spark isn’t enough. If you do only online learning, Spark is useless.

I could go on for hours about the technical aspects, but the bottom line is this: online learning is an ML solution unlike any other, and it requires a technical approach unlike any other.

Summing up online learning isn’t easy. It’s not a single learning algorithm: in fact, lots of algorithms can learn online. It’s also not fundamentally different in terms of how learning happens: you can use the same optimisation steps you always do. It doesn’t even have a bombastic, sci-fi sounding name.

What online learning is is a fundamentally different way of approaching machine learning. It’s an approach that embraces change, no matter how drastic. Its existence is predicated on the belief that since everything is in flux, we should stop seeking stationarity and instead start living in the moment.

A special thanks to Data Scientist Jarno Kiviaho, who I consider to be one of Finland’s top authorities on this topic.

If you fancy playing around with the (admittedly simple) code in this post, it’s available as a GitHub gist: <https://gist.github.com/maxpagels/b9c9001f7e5b28a5742b81b02f7704e2>

[1]: You could, theoretically, have a feature indicating a drastic event like this, but it’s impossible to account for everything.

Instance-Based and Model-Based Learning systems differ in their approach to generalization. The former type of system learns the examples 'by heart' before generalizing to new cases using a metric of similarity. The latter generalizes from a set of examples to build a model of those examples before using that model to make predictions.

Because the Model-Based Learning form is much more common, this resource will look at some examples of the more little known instance-based systems. Model-Based Learning will become more familiar to you as you work through the ML-focused units. As you'll find, machine learning often involves scrutinizing the data, selecting a model, training it on the training data, and applying the model to make predictions about new cases, trusting that model generalizes effectively.

[[](https://medium.com/@Shaier?source=post_page-----4349224ed4f3--------------------------------)](https://medium.com/@Shaier?source=post_page-----4349224ed4f3--------------------------------)

[Sagi Shaier](https://medium.com/@Shaier?source=post_page-----4349224ed4f3--------------------------------)

Feb 1, 2019

# ML Algorithms: One SD (σ)- Instance-based Algorithms

## An intro to machine learning instance-based algorithms

A close-up of a stethoscope

Description automatically generated with medium confidence

TThe obvious questions to ask when facing a wide variety of machine learning algorithms, is “which algorithm is better for a specific task, and which one should I use?”

Answering these questions vary depending on several factors, including: (1) The size, quality, and nature of data; (2) The available computational time; (3) The urgency of the task; and (4) What do you want to do with the data.

This is one section of the many algorithms I wrote about in a [previous article](https://towardsdatascience.com/ml-algorithms-one-sd-σ-74bcb28fafb6).   
In this part I tried to display and briefly explain the main algorithms (though not all of them) that are available for instance-based tasks as simply as possible.

# **Instance-based Algorithms:**

These algorithms don’t perform explicit generalization, instead they compare new problem instances with instances seen in training, which have been stored in memory.

· **K-Nearest Neighbor (KNN)**

Can be used for both classiﬁcation and regression problems. KNN stores all available cases and classiﬁes new cases by a majority vote of its K neighbors. Predictions are made for a new data point by searching through the entire training set for the K most similar instances (the neighbors) and summarizing the output variable for those K instances. For instance, if we take K=3 and we want to decide which class does a new example belongs to, we consider the 3 closest (Euclidian distance usually) points to the new example.

Diagram

Description automatically generated

For regression problems, this might be the mean output variable:

Chart, line chart, scatter chart

Description automatically generated

Some things to consider:

Choosing the optimal value for K is best done by first inspecting the data (you can use the elbow method).

It is a supervised learning algorithm.

· **Learning Vector Quantization (LVQ)**

Developed as a classification algorithm. It is capable of supporting both binary (two-class) and multi-class classification problems. A downside of K-Nearest Neighbors is that you need to hang on to your entire training dataset. The LVQ is an artificial neural network algorithm that allows you to choose how many training instances to hang onto and learns exactly what those instances should look like. The value of the number of instances is optimized during learning process.

Chart, bubble chart

Description automatically generated

Some things to consider:

It is a supervised learning method

If you discover that KNN gives good results on your dataset try using LVQ to reduce the memory requirements of storing the entire training dataset.

· **Self-Organizing Map (SOM)**

An unsupervised deep learning model, mostly used for feature detection or dimensionality reduction. SOM differ from other artificial neural networks as it apply competitive learning as opposed to error-correction learning (like backpropagation with gradient descent), and in the sense that they use a neighborhood function to preserve the topological properties of the input space. SOM performs a topologically ordered mapping from high dimensional space onto two-dimensional space. In other words, it produces a two dimensional representation of the input space of the set of training samples.

For example, let’s look at the handwritten digits dataset. The inputs for SOM are high dimensional since each input dimension represents the grayscale value of one pixel on a 28 by 28 image, which makes the inputs 784-dimensional (each dimension is a value between 0 and 255).

If we’ll map them to a 20x20 SOM and color them based on their true class (a number from 0 to 9) we’ll get the following:

Chart, scatter chart

Description automatically generated

The true classes are labelled according to the colors in the bottom left.

Take a look at the yellow region. That is where the 6s were mapped to, and notice that there is a little overlap with other categories. In comparison, take a look at the bottom left, where the green and brown points overlap. That is where the SOM was “confused” between 4s and 9s.

Another example of SOM is NLP. We can use it for a classification of let’s say 2 million medical papers. SOM will create a cluster of similar meaning words:

A picture containing text

Description automatically generated

The bottom right words are related to brain, and the top right words are related to medical imaging.

Some things to consider:

SOM outputs a 2D map for any number of indicators.

We could use the SOM for clustering data without knowing the class memberships of the input data.

· **Locally Weighted Learning (LWL)**

The basic idea behind LWL is that instead of building a global model for the whole function space, for each point of interest a local model is created based on neighboring data of the query point.

Chart

Description automatically generated

For this purpose, each data point becomes a weighting factor which expresses the influence of the data point for the prediction. In general, data points which are in the close neighborhood to the current query point are receiving a higher weight than data points which are far away. Basically, say you want to predict what is going to happen in the future. You can simply reach into a database of all your previous experiences, you then grab some similar experiences, combine them (perhaps by a weighted average that weights more similar experiences more strongly) and use the combination to make a prediction.

Some things to consider:

LWL methods are non-parametric.

Please create a free account to view this resource.

This LeetCode resource features a great explanation of the What, the How, and the Why behind Machine Learning. The course gives a solid mathematical justification for its claims. If you have a tough time understanding a particular topic covered in this resource, be sure to make a note of it and bring it up during your next mentor call.

**Data Scientists and Machine Learning**

There are two categories of challenges that machine learning practitioners and data scientists face: bad algorithms and bad data.

**Bad algorithms** can come from:

* Poor feature engineering
* Poor feature selection or poor feature extraction
* Overfitting the training data — the model performs well on the training data alone but doesn't generalize well to unseen data
* Underfitting the training data — your model is too basic to learn the underlying structure of the data
* Failure to properly cross-validate — cross-validation gives you sensible hyperparameters, but you can easily misapply this technique — you'll learn more about this later

**Bad data** can arise for many reasons. These include:

* Nonrepresentative training data — the training data is not representative of the new cases to which you want to generalize
* Sampling bias exhibited by the data
* Poor-quality data — errors, outliers, and noise (due to poor-quality measurements).

We will look at some examples of these challenges in this subunit.

[[](https://medium.com/@alexandregonfalonieri?source=post_page-----725f2abd2b92--------------------------------)](https://medium.com/@alexandregonfalonieri?source=post_page-----725f2abd2b92--------------------------------)

[Alexandre Gonfalonieri](https://medium.com/@alexandregonfalonieri?source=post_page-----725f2abd2b92--------------------------------)

May 17, 2019

·

10 min read

·

·

# Dealing with the Lack of Data in Machine Learning

A body of water with mountains in the background

Description automatically generated with medium confidence

In many projects I carried out, companies, despite having fantastic AI business ideas, display a tendency to slowly become frustrated when they realize that they do not have enough data… However, solutions do exist! **The purpose of this article is to briefly introduce you to some of them (the ones that are proven effective in my practice) rather than to list all existing solutions.**

The problem of data scarcity is very important since data are at the core of any AI project. The size of a datasetis often responsible for poor performances in ML projects.

Most of the time, data related issues are the main reason why great AI projects cannot be accomplished. In some projects, you come to the conclusion that there is no relevant data or the collection process is too difficult and time-consuming.

Supervised machine learning models are being successfully used to respond to a whole range of business challenges. However, these models are data-hungry and their performance relies heavily on the size of training data available. In many cases, it is difficult to create training datasets that are large enough.

Another issue I could mention is that project analysts tend to underestimate the amount of data necessary to handle common business problems. I remember myself struggling to collect big training datasets. It is even more complicated to gather data when working for a large company…

**How much data do I need?**

Well, you need roughly 10 times as many examples as there are degrees of freedom in your model. The more complex the model, the more you are prone to overfitting, but that can be avoided by validation. **However, much fewer data can be used based on the use case.**

**Overfitting:** refers to a model that models the training data too well. It happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.

It is also worth discussing the issue of handling the missing values. Especially, if the number of missing values in your data is big enough (above 5%).

Once again, dealing with missing values will depend on certain ‘success’ criteria.Moreover, these criteria vary for different datasets and even for different applications such as recognition, segmentation, prediction, classification, etc. (given the same dataset) even for different applications (recognition, segmentation, prediction, classification).

It is important to understand that there is no perfect way to deal with missing data.

Different solutions exist but it depends on the kind of problem — Time-series Analysis, ML, Regression, etc.

When it comes to predictive techniques, they shall be used only when missing values are not observed completely at random and the variables were chosen to impute such missing values have some relationship with it, else it could yield imprecise estimates.

In general, different machine learning algorithms can be used to determine the missing values. This works by turning missing features to labels themselves and now using columns without missing values to predict columns with missing values.

Based on my experience, you will be confronted with a lack of data or missing data at some point if you decide to build an AI-powered solution, **but fortunately, there are ways to turn that minus into a plus.**

# Lack of data?

As noted above, it is impossible to precisely estimate the minimum amount of data required for an AI project. Obviously, the very nature of your project will influence significantly the amount of data you will need. For example, texts, images, and videos usually require more data. **However, many other factors should be considered in order to make an accurate estimate.**

* **Number of categories to be predicted**   
  What is the expected output of your model? Basically, the fewest number or categories the better.
* **Model Performance**If you plan on getting a product in production, you need more. **A small dataset might be good enough for a proof of concept but in production, you’ll need way more data.**

In general, small datasets require models that have low complexity (or [high bias](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff)) to avoid [overfitting](https://en.wikipedia.org/wiki/Overfitting) the model to the data.

# Non-Technical Solutions

Before exploring technical solutions, let’s analyze what we can do to enhance your dataset. It might sound obvious but before getting started with AI, please try to obtain as much data as possible by developing your external and internal tools with data collection in mind. If you know the tasks that a machine learning algorithm is expected to perform, you can create a data-gathering mechanism in advance.

Try to establish a real data culture within your organization.

To initiate ML execution, you could rely on open source data. There are a lot of data available for ML and some companies are ready to give it away.

If you need external data for your project, it can be beneficial to form partnerships with other organizations in order to get relevant data. Forming partnerships will cost obviously cost you some time, but the proprietary data gained will build a natural barrier to any rivals.

**Build a useful application, give it away, use the data**

Another approach that I used in my previous project was to give away access to a cloud application to customers. The data that makes it into the app can be used to build machine learning models. My previous client built an application for hospitals and made it free. We gathered a lot of data thanks to it and managed to create a unique dataset for our ML solution. It really helps to tell customers or investor that you have built your own and unique dataset.

A picture containing graphical user interface

Description automatically generated

# Small datasets

Based on my experience, some common approaches that can help with building predictive models from small data sets are:

A picture containing text

Description automatically generated

In general, the simpler the machine learning algorithm the better it will learn from small data sets. From an ML perspective, **small** data requires models that have low complexity (or high bias) to avoid overfitting the model to the data. I noticed that the Naive Bayes algorithm is among the simplest classifiers and as a result learns remarkably well from relatively small data sets.

**Naive Bayes methods:** set of supervised learning algorithms based on applying Bayes’ theorem with the “naive” assumption of conditional independence between every pair of features given the value of the class variable.

You can also rely on other linear models and decision trees. Indeed, they can also perform relatively well on small data sets. Basically, simple models are able to learn from small data sets better than more complicated models (neural networks) since they are essentially trying to learn less.

For very **small datasets**, Bayesian methods are generally the best in class, although the results can be sensitive **to your choice of prior.** I think that the naive Bayes classifier and ridge regression are the best predictive models.

When it comes to **small** datasets, you need models that have few parameters (low complexity) and/or a strong prior. You can also interpret the “prior” as an assumption you can make on how the data behaves.

Graphical user interface, text, application, email

Description automatically generated

**Many other solutions do exist depending on the exact nature of your business issues and the size of your dataset.**

# Transfer learning

**Definition:** a framework that leverages existing relevant data or models while building a machine learning model.

Transfer learning uses knowledge from a learned task to improve the performance on a related task, typically reducing the amount of required training data.

Transfer learning techniques are useful because they allow models to make predictions for a new domain or task (known as the target domain) using knowledge learned from another dataset or existing machine learning models (the source domain).

**Transfer learning techniques should be considered when you do not have enough target training data, and the source and target domains have some similarities but are not identical.**

Diagram

Description automatically generated

Naively aggregating models or different datasets would not always work! If the existing datasets are very different from the target data then the new learner can be negatively impacted by existing data or models.

Transfer learning works well when you have other datasets you can use to infer knowledge, but what happens when you have no data at all? This is where data generation can play a role. It is used when no data is available, or when you need to create more data than you could amass even through aggregation.

In this case, the small amount of data that does exist is modified to create variations on that data to train the model. For example, many images of a car can be generated by cropping, cropping, downsizing, one single image of a car.

Unfortunately, the lack of quality labeled data is also one of the largest challenges facing data science teams, but by using techniques such as transfer learning and data generation it is possible to overcome data scarcity.

Another common application of transfer learning is to train models on cross-customer datasets to overcome the cold-start problem. I noticed that SaaS companies often have to deal with when onboarding new customers to their ML products. Indeed, until the new customer has collected enough data to achieve good model performance (which could take several months) it’s hard to provide value

# Data Augmentation

Data augmentation means increasing the number of data points. In my latest project, we used data augmentation techniques to increase the number of images in our dataset. In terms of traditional row/column format data, it means increasing the number of rows or objects.

We had no choice but to rely on data augmentation for two reasons: Time and Accuracy. Every data collection process is associated with a cost. This cost can be in terms of dollars, human effort, computational resources and of course time consumed in the process.

Graphical user interface, text, application, email

Description automatically generated

As a consequence, we had to augment existing data to increase the data size that we feed to our ML classifiers and to compensate for the cost involved in further data collection.

There are many ways to augment data.

In our case, you can rotate the original image, change lighting conditions, crop it differently, so for one image you can generate different sub-samples. **This way you can reduce overfitting your classifier.**

However, if you are generating artificial data using over-sampling methods such as SMOTE, then there is a fair chance you may introduce over-fitting.

**Over-fitting:** An overfitted model is a model with a trend line that reflects the errors in the data that it is trained with, instead of accurately predicting unseen data.

**This is something you must take into consideration when developing your AI solution.**

A dog lying on the floor

Description automatically generated with low confidence

# Synthetic Data

Synthetic data means fake data that contains the same schema and statistical properties as its “real” counterpart. Basically, it looks so real that it’s nearly impossible to tell that it’s not.

**So what’s the point of synthetic data, and why does it matter if we already have access to the real thing?**

I have seen synthetic data applied especially when we were dealing with private data (banking, healthcare, etc.), this makes the use of synthetic data a more secure approach to development in certain instances.

Synthetic data is used mostly when there is not enough real data or there is not enough real data for specific patterns you know about. Usage mostly the same for training and testing datasets.

Synthetic Minority Over-sampling Technique (SMOTE) and Modified- SMOTE are two such techniques which generate synthetic data. Simply put, SMOTE takes the minority class data points and creates new data points which lie between any two nearest data points joined by a straight line.

The algorithm calculates the distance between two data points in the feature space, multiplies the distance by a random number between 0 and 1 and places the new data point at this new distance from one of the data points used for distance calculation.

In order to generate synthetic data, you have to use a Training Set to define a model, which would require validation, and then by changing the parameters of interest, you can generate synthetic data, through simulation. The domain/data type is significant since it affects the complexity of the entire process.

Diagram

Description automatically generated with low confidence

In my opinion, asking yourself if you have enough data will reveal inconsistencies that you have probably never spotted before. It will help to highlight issues in your business processes that you thought were perfect and make you understand why it is the key to creating a successful data strategy within your organization.

[[](https://medium.com/@emrerencberoglu?source=post_page-----3a5e293a5114--------------------------------)](https://medium.com/@emrerencberoglu?source=post_page-----3a5e293a5114--------------------------------)

[Emre Rençberoğlu](https://medium.com/@emrerencberoglu?source=post_page-----3a5e293a5114--------------------------------)

Apr 1, 2019

·

# Fundamental Techniques of Feature Engineering for Machine Learning

## All required methods for comprehensive data preprocessing with Pandas examples.

A picture containing wrench, tool

Description automatically generated

# Introduction

What is a feature and why we need the engineering of it? Basically, all machine learning algorithms use some input data to create outputs. This input data comprise features, which are usually in the form of structured columns. Algorithms require features with some specific characteristic to work properly. Here, the need for **feature engineering** arises. I think feature engineering efforts mainly have two goals:

* Preparing the proper input dataset, compatible with the machine learning algorithm requirements.
* Improving the performance of machine learning models.

The features you use influence more than everything else the result. No algorithm alone, to my knowledge, can supplement the information gain given by correct **feature engineering**.

— Luca Massaron

According to a survey in Forbes, data scientists spend **80%** of their time on **data preparation:**

Chart, sunburst chart

Description automatically generated

[Source: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/](https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#1594bda36f63)

This metric is very impressive to show the importance of feature engineering in data science. Thus, I decided to write this article, which summarizes the main techniques of feature engineering with their short descriptions. I also added some basic python scripts for every technique. You need to import **Pandas** and **Numpy** library to run them.

import pandas as pd  
import numpy as np

Some techniques above might work better with some algorithms or datasets, while some of them might be beneficial in all cases. This article does not aim to go so much deep in this aspect. Tough, it is possible to write an article for every method above, I tried to keep the explanations brief and informative. I think the best way to achieve expertise in feature engineering is practicing different techniques on various datasets and observing their effect on model performances.

## List of Techniques

* [1.Imputation](https://medium.com/p/3a5e293a5114#3abe)
* [2.Handling Outliers](https://medium.com/p/3a5e293a5114#1c08)
* [3.Binning](https://medium.com/p/3a5e293a5114#7559)
* [4.Log Transform](https://medium.com/p/3a5e293a5114#199b)
* [5.One-Hot Encoding](https://medium.com/p/3a5e293a5114#7c18)
* [6.Grouping Operations](https://medium.com/p/3a5e293a5114#ad97)
* [7.Feature Split](https://medium.com/p/3a5e293a5114#3149)
* [8.Scaling](https://medium.com/p/3a5e293a5114#83e6)
* [9.Extracting Date](https://medium.com/p/3a5e293a5114#8068)

# 1.Imputation

A person holding a heart

Description automatically generated with low confidence

Missing values are one of the most common problems you can encounter when you try to prepare your data for machine learning. The reason for the missing values might be human errors, interruptions in the data flow, privacy concerns, and so on. Whatever is the reason, missing values affect the performance of the machine learning models.

Some machine learning platforms automatically drop the rows which include missing values in the model training phase and it decreases the model performance because of the reduced training size. On the other hand, most of the algorithms do not accept datasets with missing values and gives an error.

The most simple solution to the missing values is to drop the rows or the entire column. There is not an optimum threshold for dropping but you can use **70%** as an example value and try to drop the rows and columns which have missing values with higher than this threshold.

threshold = 0.7**#Dropping columns with missing value rate higher than threshold**  
data = data[data.columns[data.isnull().mean() < threshold]]  
  
**#Dropping rows with missing value rate higher than threshold**  
data = data.loc[data.isnull().mean(axis=1) < threshold]

## Numerical Imputation

Imputation is a more preferable option rather than dropping because it preserves the data size. However, there is an important selection of what you impute to the missing values. I suggest beginning with considering a possible default value of missing values in the column. For example, if you have a column that only has **1** and **NA**, then it is likely that the **NA** rows correspond to **0**. For another example, if you have a column that shows the **“customer visit count in last month”**, the missing values might be replaced with **0** as long as you think it is a sensible solution.

Another reason for the missing values is joining tables with different sizes and in this case, imputing **0** might be reasonable as well.

Except for the case of having a default value for missing values, I think the best imputation way is to use the **medians** of the columns. As the averages of the columns are sensitive to the outlier values, while medians are more solid in this respect.

**#Filling all missing values with 0**  
data = data.fillna(0)**#Filling missing values with medians of the columns**  
data = data.fillna(data.median())

## Categorical Imputation

Replacing the missing values with the **maximum occurred value** in a column is a good option for handling categorical columns. But if you think the values in the column are distributed uniformly and there is not a dominant value, imputing a category like “**Other**” might be more sensible, because in such a case, your imputation is likely to converge a random selection.

**#Max fill function for categorical columns**  
data['column\_name'].fillna(data['column\_name'].value\_counts()  
.idxmax(), inplace=True)

# 2.Handling Outliers

Before mentioning how outliers can be handled, I want to state that the best way to detect the outliers is to demonstrate the data visually. All other statistical methodologies are open to making mistakes, whereas visualizing the outliers gives a chance to take a decision with high precision. Anyway, I am planning to focus visualization deeply in another article and let’s continue with statistical methodologies.

Statistical methodologies are less precise as I mentioned, but on the other hand, they have a superiority, they are fast. Here I will list two different ways of handling outliers. These will detect them using **standard deviation**, and **percentiles**.

## Outlier Detection with Standard Deviation

If a value has a distance to the average higher than **x \* standard deviation,** it can be assumed as an outlier. Then what **x** should be?

There is no trivial solution for x, but usually, a value between 2 and 4 seems practical.

**#Dropping the outlier rows with standard deviation**  
factor = 3  
upper\_lim = data['column'].mean () + data['column'].std () \* factor  
lower\_lim = data['column'].mean () - data['column'].std () \* factor  
  
data = data[(data['column'] < upper\_lim) & (data['column'] > lower\_lim)]

In addition, **z-score** can be used instead of the formula above. **Z-score** (or standard score) standardizes the distance between a value and the mean using the standard deviation.

## Outlier Detection with Percentiles

Another mathematical method to detect outliers is to use percentiles. You can assume a certain percent of the value from the top or the bottom as an outlier. The key point is here to set the percentage value once again, and this depends on the distribution of your data as mentioned earlier.

Additionally, a common mistake is using the percentiles according to the range of the data. In other words, if your data ranges from **0** to **100**, your top **5%** is not the values between **96** and **100**. Top **5%** means here the values that are out of the **95th** percentile of data.

**#Dropping the outlier rows with Percentiles**  
upper\_lim = data['column'].quantile(.95)  
lower\_lim = data['column'].quantile(.05)  
  
data = data[(data['column'] < upper\_lim) & (data['column'] > lower\_lim)]

## An Outlier Dilemma: Drop or Cap

Another option for handling outliers is to **cap** them instead of dropping. So you can keep your data size and at the end of the day, it might be better for the final model performance.

On the other hand, capping can affect the distribution of the data, thus it better not to exaggerate it.

**#Capping the outlier rows with Percentiles**  
upper\_lim = data['column'].quantile(.95)  
lower\_lim = data['column'].quantile(.05)data.loc[(df[column] > upper\_lim),column] = upper\_lim  
data.loc[(df[column] < lower\_lim),column] = lower\_lim

# 3.Binning

Chart, histogram

Description automatically generated

Binning illustration of numerical data

Binning can be applied on both categorical and numerical data:

**#Numerical Binning ExampleValue Bin**   
0-30 -> Low   
31-70 -> Mid   
71-100 -> High**#Categorical Binning ExampleValue Bin**   
Spain -> Europe   
Italy -> Europe   
Chile -> South America  
Brazil -> South America

The main motivation of binning is to make the model more **robust** and prevent **overfitting**, however, it has a cost to the performance. Every time you bin something, you sacrifice information and make your data more regularized. (Please see [regularization in machine learning](https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a))

The trade-off between **performance** and **overfitting** is the key point of the binning process. In my opinion, for numerical columns, except for some obvious overfitting cases, binning might be redundant for some kind of algorithms, due to its effect on model performance.

However, for categorical columns, the labels with low frequencies probably affect the robustness of statistical models negatively. Thus, assigning a general category to these less frequent values helps to keep the robustness of the model. For example, if your data size is **100,000** rows, it might be a good option to unite the labels with a count less than **100** to a new category like **“Other”**.

**#Numerical Binning Example**data['bin'] = pd.cut(data['value'], bins=[0,30,70,100], labels=["Low", "Mid", "High"]) **value bin**  
0 2 Low  
1 45 Mid  
2 7 Low  
3 85 High  
4 28 Low**#Categorical Binning Example** **Country**  
0 Spain  
1 Chile  
2 Australia  
3 Italy  
4 Brazilconditions = [  
 data['Country'].str.contains('Spain'),  
 data['Country'].str.contains('Italy'),  
 data['Country'].str.contains('Chile'),  
 data['Country'].str.contains('Brazil')]  
  
choices = ['Europe', 'Europe', 'South America', 'South America']  
  
data['Continent'] = np.select(conditions, choices, default='Other') **Country Continent**  
0 Spain Europe  
1 Chile South America  
2 Australia Other  
3 Italy Europe  
4 Brazil South America

# 4.Log Transform

Logarithm transformation (or log transform) is one of the most commonly used mathematical transformations in feature engineering. What are the benefits of log transform:

* It helps to handle skewed data and after transformation, the distribution becomes more approximate to normal.
* In most of the cases the magnitude order of the data changes within the range of the data. For instance, the difference between ages **15** and **20** is not equal to the ages **65** and **70**. In terms of years, yes, they are identical, but for all other aspects, **5** years of difference in young ages mean a higher magnitude difference. This type of data comes from a multiplicative process and log transform normalizes the magnitude differences like that.
* It also decreases the effect of the outliers, due to the normalization of magnitude differences and the model become more robust.

**A critical note:** The data you apply log transform must have only positive values, otherwise you receive an error. Also, you can add **1** to your data before transform it. Thus, you ensure the output of the transformation to be positive.

**Log(x+1)**

**#Log Transform Example**  
data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})data['log+1'] = (data['value']+1).transform(np.log)**#Negative Values Handling  
#Note that the values are different**data['log'] = (data['value']-data['value'].min()+1) .transform(np.log) **value log(x+1) log(x-min(x)+1)**  
0 2 1.09861 3.25810  
1 45 3.82864 4.23411  
2 -23 nan 0.00000  
3 85 4.45435 4.69135  
4 28 3.36730 3.95124  
5 2 1.09861 3.25810  
6 35 3.58352 4.07754  
7 -12 nan 2.48491

# 5.**One-hot encoding**

**One-hot encoding** is one of the most common encoding methods in machine learning. This method spreads the values in a column to multiple flag columns and assigns **0** or **1** to them. These binary values express the relationship between grouped and encoded column.

This method changes your categorical data, which is challenging to understand for algorithms, to a numerical format and enables you to group your categorical data without losing any information. (For details please see the last part of **Categorical Column Grouping**)

Table

Description automatically generated

One hot encoding example on City column

**Why One-Hot?:** If you have **N** distinct values in the column, it is enough to map them to **N-1** binary columns, because the missing value can be deducted from other columns. If all the columns in our hand are equal to **0**, the missing value must be equal to **1**. This is the reason why it is called as **one-hot encoding**. However, I will give an example using the **get\_dummies** function of Pandas. This function maps all values in a column to multiple columns.

encoded\_columns = pd.get\_dummies(data['column'])  
data = data.join(encoded\_columns).drop('column', axis=1)

# 6.Grouping Operations

In most machine learning algorithms, every instance is represented by a row in the training dataset, where every column show a different feature of the instance. This kind of data called **“Tidy”**.

Tidy datasets are easy to manipulate, model and visualise, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table.

— Hadley Wickham

Datasets such as transactions rarely fit the definition of tidy data above, because of the multiple rows of an instance. In such a case, we group the data by the instances and then every instance is represented by only one row.

The key point of group by operations is to decide the aggregation functions of the features. For numerical features, average and sum functions are usually convenient options, whereas for categorical features it more complicated.

## Categorical Column Grouping

I suggest three different ways for aggregating categorical columns:

* The first option is to select the label with the **highest frequency**. In other words, this is the **max** operation for categorical columns, but ordinary max functions generally do not return this value, you need to use a lambda function for this purpose.

data.groupby('id').agg(lambda x: x.value\_counts().index[0])

* Second option is to make a **pivot table**. This approach resembles the encoding method in the preceding step with a difference. Instead of binary notation, it can be defined as aggregated functions for the values between grouped and encoded columns. This would be a good option if you aim to go beyond binary flag columns and merge multiple features into aggregated features, which are more informative.

Table

Description automatically generated

Pivot table example: Sum of Visit Days grouped by Users

**#Pivot table Pandas Example**data.pivot\_table(index='column\_to\_group', columns='column\_to\_encode', values='aggregation\_column', aggfunc=np.sum, fill\_value = 0)

* Last categorical grouping option is to apply a **group by** function after applying **one-hot encoding**. This method preserves all the data -in the first option you lose some-, and in addition, you transform the encoded column from categorical to numerical in the meantime. You can check the next section for the explanation of **numerical column grouping**.

## Numerical Column Grouping

Numerical columns are grouped using **sum** and **mean** functions in most of the cases. Both can be preferable according to the meaning of the feature. For example, if you want to obtain **ratio** columns, you can use the average of binary columns. In the same example, sum function can be used to obtain the total count either.

#sum\_cols: List of columns to sum  
#mean\_cols: List of columns to averagegrouped = data.groupby('column\_to\_group')  
  
sums = grouped[sum\_cols].sum().add\_suffix('\_sum')  
avgs = grouped[mean\_cols].mean().add\_suffix('\_avg')  
  
new\_df = pd.concat([sums, avgs], axis=1)

# 7.Feature Split

A picture containing building, outdoor, nature, stone

Description automatically generated

Photo by [Jaxon Lott](https://unsplash.com/@jaxonlott?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

Splitting features is a good way to make them useful in terms of machine learning. Most of the time the dataset contains string columns that violates [tidy data](http://vita.had.co.nz/papers/tidy-data.html) principles. By extracting the utilizable parts of a column into new features:

* We enable machine learning algorithms to comprehend them.
* Make possible to bin and group them.
* Improve model performance by uncovering potential information.

**Split** function is a good option, however, there is no one way of splitting features. It depends on the characteristics of the column, how to split it. Let’s introduce it with two examples. First, a simple split function for an ordinary name column:

**data.name**  
0 Luther N. Gonzalez  
1 Charles M. Young  
2 Terry Lawson  
3 Kristen White  
4 Thomas Logsdon#Extracting first names **data.name.str.split(" ").map(lambda x: x[0])**0 Luther  
1 Charles  
2 Terry  
3 Kristen  
4 Thomas#Extracting last names **data.name.str.split(" ").map(lambda x: x[-1])**0 Gonzalez  
1 Young  
2 Lawson  
3 White  
4 Logsdon

The example above handles the names longer than two words by taking only the first and last elements and it makes the function robust for corner cases, which should be regarded when manipulating strings like that.

Another case for split function is to extract a string part between two chars. The following example shows an implementation of this case by using two split functions in a row.

#String extraction example **data.title.head()**  
0 Toy Story (1995)  
1 Jumanji (1995)  
2 Grumpier Old Men (1995)  
3 Waiting to Exhale (1995)  
4 Father of the Bride Part II (1995)**data.title.str.split("(", n=1, expand=True)[1].str.split(")", n=1, expand=True)[0]**0 1995  
1 1995  
2 1995  
3 1995  
4 1995

# 8.Scaling

In most cases, the numerical features of the dataset do not have a certain **range** and they differ from each other. In real life, it is nonsense to expect **age** and **income** columns to have the same range. But from the machine learning point of view, how these two columns can be compared?

Scaling solves this problem. The continuous features become identical in terms of the range, after a scaling process. This process is not mandatory for many algorithms, but it might be still nice to apply. However, the algorithms based on **distance** calculations such as **k-NN** or **k-Means** need to have scaled continuous features as model input.

Basically, there are two common ways of scaling:

## Normalization



Normalization (or **min-max normalization**) scale all values in a fixed range between **0** and **1**. This transformation does not change the distribution of the feature and due to the decreased standard deviations, the effects of the **outliers** increases. Therefore, before normalization, it is recommended to handle the outliers.

data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})  
  
data['normalized'] = (data['value'] - data['value'].min()) / (data['value'].max() - data['value'].min()) **value normalized**  
0 2 0.23  
1 45 0.63  
2 -23 0.00  
3 85 1.00  
4 28 0.47  
5 2 0.23  
6 35 0.54  
7 -12 0.10

## Standardization

Standardization (or **z-score normalization**) scales the values while taking into account standard deviation. If the standard deviation of features is different, their range also would differ from each other. This reduces the effect of the outliers in the features.

In the following formula of standardization, the **mean** is shown as **μ** and the **standard** **deviation** is shown as **σ**.



data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})  
  
data['standardized'] = (data['value'] - data['value'].mean()) / data['value'].std() **value standardized**  
0 2 -0.52  
1 45 0.70  
2 -23 -1.23  
3 85 1.84  
4 28 0.22  
5 2 -0.52  
6 35 0.42  
7 -12 -0.92

# 9.Extracting Date

Though date columns usually provide valuable information about the model target, they are neglected as an input or used nonsensically for the machine learning algorithms. It might be the reason for this, that dates can be present in numerous formats, which make it hard to understand by algorithms, even they are simplified to a format like **"01–01–2017"**.

Building an ordinal relationship between the values is very challenging for a machine learning algorithm if you leave the date columns without manipulation. Here, I suggest three types of preprocessing for dates:

* Extracting the parts of the date into different columns: Year, month, day, etc.
* Extracting the time period between the current date and columns in terms of years, months, days, etc.
* Extracting some specific features from the date: Name of the weekday, Weekend or not, holiday or not, etc.

If you transform the date column into the extracted columns like above, the information of them become disclosed and machine learning algorithms can easily understand them.

from datetime import date  
  
data = pd.DataFrame({'date':  
['01-01-2017',  
'04-12-2008',  
'23-06-1988',  
'25-08-1999',  
'20-02-1993',  
]})  
  
**#Transform string to date**  
data['date'] = pd.to\_datetime(data.date, format="%d-%m-%Y")  
  
**#Extracting Year**  
data['year'] = data['date'].dt.year  
  
**#Extracting Month**  
data['month'] = data['date'].dt.month  
  
**#Extracting passed years since the date**  
data['passed\_years'] = date.today().year - data['date'].dt.year  
  
**#Extracting passed months since the date**  
data['passed\_months'] = (date.today().year - data['date'].dt.year) \* 12 + date.today().month - data['date'].dt.month  
  
**#Extracting the weekday name of the date**  
data['day\_name'] = data['date'].dt.day\_name() **date year month passed\_years passed\_months day\_name**  
0 2017-01-01 2017 1 2 26 Sunday  
1 2008-12-04 2008 12 11 123 Thursday  
2 1988-06-23 1988 6 31 369 Thursday  
3 1999-08-25 1999 8 20 235 Wednesday  
4 1993-02-20 1993 2 26 313 Saturday

# Conclusion

Diagram

Description automatically generated with low confidence

<https://xkcd.com/1838/>

I tried to explain fundamental methods that can be beneficial in the feature engineering process. After this article, proceeding with other topics of data preparation such as **feature selection, train/test splitting,** and **sampling** might be a good option.

You can check my [other article about Oversampling](https://towardsdatascience.com/how-to-calibrate-undersampled-model-scores-8f3319c1ea5b).

Lastly, I want to conclude the article with a reminder. These techniques are not magical tools. If your data tiny, dirty and useless, feature engineering may remain incapable. Do not forget “**garbage in, garbage out!”**

 Machine Learning Model

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

The term, Machine Learning, often mystifies its nature of computer science, as its name might suggest that the machine is learning as human does, or even better.

Despite the hope that one day we could have machines that think and learn the way that humans do, machine learning nowadays does not go beyond a computer program that performs the predefined procedures. What distinguishes a machine learning algorithm from a non-machine-learning algorithm, such as a program that controls traffic lights, is its ability to adapt its behaviors to new input. And this adaptation, which seems to have no human intervention, occasionally leads to the impression that the machine is actually learning. However, underneath the machine learning model, this adaptation of behaviors is as rigid as every bit of machine instructions that are programmed by humans.

***So what is a machine learning model ?***

A machine learning algorithm is the process that uncovers the underlying relationship within the data.

The outcome of a machine learning algorithm is called **machine learning model**, which can be considered as a function FF, which outputs certain results, when given the input.

Rather than a predefined and fixed function, a machine learning model is derived from historical data. Therefore, when fed with different data, the output of machine learning algorithm changes, i.e. the machine learning model changes.

For example, in the scenario of image recognition, one might train a machine learning model to recognize the object in the photos. In one case, one might feed thousands of images with and without cats to a machine learning algorithm, in order to obtain a model that is capable to tell whether there is a cat in a photo. As a result, the input of the generated model would be a digital photo, and the output is a boolean value indicating the existence of a cat on the photo.

Diagram

Description automatically generated

The machine learning model in the above case is a function that maps multiple dimensional pixel values to a binary value. Assume that we have a photo of 3 pixels, and the value of each pixel range from 0 to 255. Then the mapping space between the input and the output would be (256×256×256)×2(256×256×256)×2, which is around 33 million. We can convince ourselves that it must be a daunting task to learn this mapping (machine learning model) in a real-world case where a normal photo accounts for millions of pixels and each pixel is composed of three colors (RGB) instead of a single grey color.

The task of machine learning, is to ***learn*** the function, from the vast mapping space.

The process of discovering the latent mapping relationship between millions of pixels and a Yes/No answer, is what we call ***machine learning***, in this case. Most of the time, what we learn at the end, is an ***approximation*** to this underlying relationship. Due to its nature of approximation, one should not be disappointed to find that the results of a machine learning model is often not 100%100% accurate. Before the wide application of deep learning in 2012, the best machine learning model can only achieve around 75%75% accuracy in the [ImageNet visual recognition challenge](http://www.image-net.org/). Till nowadays, still, no machine learning model can claim 100%100% accuracy, although there are models that make fewer errors (<5%<5%) than humans in this task.

[ntest](https://leetcode.com/contest/)

[Discuss](https://leetcode.com/discuss/)

Store

1

[0](https://leetcode.com/problems/smallest-number-in-infinite-set/)

  Back to Chapter

 Supervised VS. Unsupervised

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

Given a machine learning problem, first of all, one can determine whether it is a supervised or unsupervised problem.

For any machine learning problem, we start from a data set, which consists of a group of samples. Each sample can be represented as a tuple of attributes.

For example, there is a famous classic data set called [Iris](https://archive.ics.uci.edu/ml/datasets/iris), which is first published in the paper of "*The use of multiple measurements in taxonomic problems*" - Ronald. A. Fisher (1936) **[1]**. The Iris data set consists of measurement for 150 samples of iris flower. Each sample contains the measurement for the length and the width of its petal and sepal, and a class attribute that indicates the category of iris flower, namely *setosa*, *versicolor* and *virginica*. Here are a few samples of the Iris data set.

Table

Description automatically generated

***Supervised Learning***

In a ***supervised*** learning task, the data sample would contain a target attribute yy, also known as ***ground truth***. And the task is to learn a function F, that takes the non-target attributes X and output a value that approximates the target attribute, *i.e.* F(X)≈yF(X)≈y**.** The target attribute yy serves as a teacher to guide the learning task, since it provides a benchmark on the results of learning. Hence, the task is called supervised learning.

In the Iris data set, the *class* attribute (category of iris flower) can serve as a target attribute. The data with a target attribute is often called "***labeled***" data. Based on the above definition, for the task of predicting the category of iris flower with the labeled data, one can tell that it is a supervised learning task.

***Unsupervised Learning***

In contrary to the supervised learning task, we do not have the ground truth in an ***unsupervised*** learning task. One is expected to learn the underlying patterns or rules from the data, without having the predefined ground truth as the benchmark.

One might wonder, without the *supervision* of the ground truth, can we still learn anything? The answer is yes. Here are a few examples of the unsupervised learning tasks:

* **Clustering**: given a data set, one can cluster the samples into groups, based on the similarities among the samples within the data set. For instance, a sample could be a customer profile, with attributes such as the number of items that the customer bought, the time that the customer spent on the shopping site *etc*. One can cluster the customer profiles into groups, based on the similarities of the attributes. With the clustered groups, one could devise specific commercial campaigns targeting each group, which might help attract and retain customers.
* **Association**:  given a data set, the association task is to uncover the hidden association patterns among the attributes of a sample. For instance,  a sample could be a shopping cart of a customer, where each attribute of the sample is a merchandise. By looking into the shopping carts, one might discover that customers who bought beers *often* bought diapers as well, *i.e.* there is a strong association between the beer and the diaper in the shopping cart. With this learned insight, the supermarket could rearrange those strongly associated merchandise into the nearby corners, in order to promote the sales of one or another.

***Semi-supervised Learning***

In a scenario where the data set is massive but the labeled sample are few, one might find the application of both supervised and unsupervised learning. We can call this task as ***semi-supervised learning***.

In many scenarios, it is prohibitively time-consuming and expensive to collect a large amount of labeled data, which often involves manual efforts. It takes two and a half years for a research team from Stanford to curate the famous [ImageNet](https://image-net.org/about.php) which contains millions of images with thousands of manually labeled categories. As a result, it is often the case that one has a large amount of data, yet few of them are accurately "labeled", e.g. videos without category or even a title.

By combining both the supervised and unsupervised learning in a data set with few labels, one could exploit the data set to a better extend and obtain a better result than just applying each of them individually.

For example, one would like to predict the label of images, but only 10%10% of the images are labeled. By applying supervised learning, we train a model with the labeled data, then we apply the model to predict the unlabeled data. It would be hard to convince ourselves that the model would be general enough, after all we learned from only the minority of data set. A better strategy could be to first cluster the images into groups (unsupervised learning), and then apply the supervised learning algorithm on each of the groups individually. The unsupervised learning in the first stage could help us to narrow down the scope of learning so that the supervised learning in the second stage could obtain better accuracy.

***References***

[1]. Fisher,R.A. "[*The use of multiple measurements in taxonomic problems*](https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x)" Annual Eugenics, 7, Part II, 179-188 (1936)

ew

[Contest](https://leetcode.com/contest/)

[Discuss](https://leetcode.com/discuss/)

Store

1

[0](https://leetcode.com/problems/smallest-number-in-infinite-set/)

  Back to Chapter

 Classification VS. Regression

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

In the previous section, we define a machine learning model as a function FF, that takes certain input and generates an output. Often we further distinguish the machine learning models as classification and regression, based on the type of output values.

If the output of a machine learning model is discrete values, *e.g.* a boolean value, we then call it a classification model. While we call the model that outputs continuous values as regression model.

***Classification Model***

For example, the model that tells whether a photo contains a cat or not can be considered as a classification model, since we can represent the output with a boolean value.

Diagram

Description automatically generated

To be more specific, the input can be represented as a matrix MM with dimensions of H×WH×W where HH is the height of the photo in pixels and WW is the width of the photo. Each element within the matrix is the grayscale value of each pixel in the photo, *i.e.* an integer value between [0,255][0,255] that indicates the intensity of color. The expected output of the model would be a binary value [1∣0][1∣0], indicating whether the photo shows a cat or not. To summarize, our cat-photo-recognition model FF can be formulated as follows:

F(M[H][W])=1∣0,where M[i][j]∈[0,255],0<i<H,0<j<WF(M[H][W])=1∣0,where M[i][j]∈[0,255],0<i<H,0<j<W

And the goal of machine learning is to discover a function that is as general as possible, which has a high probability to give the right answer for unseen data.

***Regression Model***

As for the examples of regression model, one can consider a model that estimates the price of a real estate, given the characteristics such as the surface, the type of real estate (*e.g.* house, apartment), and of course the location. In this case, we can consider the expected output as a real value p∈Rp∈R, therefore it is a regression model. Note, in this example, the raw data that we have is not all numeric, but certain of them are categorical, such as the type of real estate. This is often the case in real-world problems.

For each real estate that is under consideration, we can represent its characteristics as a tuple TT, where each element within the tuple is either a numeric value, or a categorical value that represents one of its attributes. The elements are also called features in many cases. To summarize, we can formulate our real-estate-price-estimation model as follows:

F(T)=p,where p∈RF(T)=p,where p∈R

To be more specific, let's consider a real estate with the following features:

surface = 120 m2120 m2,  type = ' apartment', location = ' NY downtown', year\_of\_construction = 2000

Now given the above features, if our model FF gives a value like 10,000$10,000$, then most likely that our model is not a good fit for the problem.

As an example, in the following graph, we show a regression model with the surface of the estate as the only variable, and the price of the estate as the output.

Chart

Description automatically generated

Speaking of features, we would also like to mention that some of the machine learning models (*e.g.* decision tree) can handle directly the non-numeric feature as it is, while more often one has to transform those non-numeric features into numeric one way or another.

***Problem Conversion***

Given a real-world problem, sometimes one can easily formulate it and quickly attribute it to either a classification or regression problem. However, sometimes the boundary between these two models is not clear, and one can transform a classification problem into a regression problem, and vice versa.

In the above example of real estate price estimation, it seems difficult to predict the exact price of a real estate. However, if we reformulate the problem as predicting the price range of real estate instead of a single price tag, then we could expect to obtain a more robust model. As a result, we transform the problem into a classification problem, instead of regression.

As to our cat-photo-recognition model, we can also transform it from classification to regression. Instead of giving a binary value as the output, we could define a model to give a probability value [0,100%][0,100%] on whether the photo shows a cat. In this way, one can compare the nuance between models and further tune the models. For instance, for a photo with a cat, a model A gives the probability 1%1%, while model B gives the probability 49%49% for the same photo. Although both models fail to give the right answer, one can tell that model B is closer to the truth. In this scenario, one often applies one of the machine learning models called Logistic Regression, which gives continuous probability values as output, but it is served to solve the classification problem.

1

[0](https://leetcode.com/problems/smallest-number-in-infinite-set/)

  Back to Chapter

 Data, Data, Data !

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

The ultimate goal of the machine learning workflow is to build a machine learning model. We obtain the model from the data. As a result, it is the data that determines the ***upper bound*** of performance that the model can achieve. There are numerous models that can fit a specific data. The best that we can do, is to find a model that can approach the most to the upper bound set by the data. We cannot really expect that a model can learn something else out of the scope of data.

**Rule of thumb**: garbage in, garbage out.

It might be appropriate to illustrate the above point with the parable of the blind men and an elephant. The story goes like this, a group of blind men, who have never come across an elephant before, would like to learn and conceptualize what an elephant is like by touching it. Each man touches a part of the body, such as leg, tusk or tail *etc*. While each of them got a part of the reality, none of them has the whole picture of an elephant. Therefore, none of them actually learned the true image of an elephant.

Text

Description automatically generated

Now, back to our machine learning task, the training data we got could be those images of legs or tusks from an elephant, while during the test processing, the testing data we got are the full portraits of elephants. It would not be surprising to find out that our trained model does not perform well in this case, since we do not have the ***high-quality*** training data that is closer to the reality in the first place.

One might wonder that if the data is really important, then why not feeding the "high-quality" data such as full portraits of elephants into the algorithm, instead of snapshots on parts of the elephant body. Because, facing a problem, we or the machine, like the "blind-men", often struggle to gather the data that captures the essential characteristics of the problem, either due to the technical issues (*e.g.* data privacy) or simply because we do not perceive the problem in the right way.

In the real world, the data we got reflects a part of reality in a favorable case, or it could be some noise in a less favorable case, or in the worst case, even a contradiction to the reality. Regardless of the machine learning algorithms, one would not be able to learn anything from data that contains too much noise or is too inconsistent with the reality.

 Machine Learning Workflow

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

In the previous section, we clarify the notion of machine learning model. In this section, we discuss a typical workflow to construct a machine learning model.

First of all, one cannot talk about machine learning, without mentioning about the data. The relationship between the data and the machine learning model, is as critical as the fuel to the engine of rocket.

***Data-Centric Workflow***

The workflow to build a machine learning model is centralized around the data.

It is not exaggerating to say that the data dictates how the machine learning model is built. In the following graph, we illustrate a typical workflow involved in a project of machine learning.

Diagram

Description automatically generated

Starting from the data, we first determine which type of machine learning problems we would like to solve, *i.e.* ***supervised*** vs. ***unsupervised***. We say that the data is *labeled*, if one of the attributes in the data is the desired one, *i.e.* the target attribute. For instance, in the task of telling whether there is a cat on a photo, the target attribute of the data could be a boolean value [Yes|No]. If this target attribute exists, then we say the data is labeled, and it is a supervised learning problem.

For the supervised machine learning algorithms, we further determine the type of the generated model: ***classification*** or ***regression***, based on the expected output of the model, ***i.e.*** discrete value for classification model and continuous value for the regression model.

Once we determine on the type of model that we would like to build out of the data, we then go ahead to perform the ***feature engineering***, which is a group of activities that transform the data into the desired format. Here are a few examples:

* For almost all cases, we ***split*** the data into two groups: training and testing. The training dataset is used during the process to train a model, while the testing dataset is then used to test or validate whether the model we build is generic enough that can be applied to the unseen data.
* The raw dataset is often ***incomplete*** with missing values. Therefore, one might need to fill those missing values with various strategies such as filling with the average value.
* The dataset often contains categorical attributes, such as country, gender *etc.* And it is often the case that one needs to ***encode*** those categorical string values into numerical one, due to the constraints of algorithm. For example, the Linear Regression algorithm can only deal with vectors of real values as input.

The process of feature engineering is ***not*** a ***one-off*** step. Often one needs to repeatedly come back to the feature engineering later in the workflow.

Once the data is ready, we then select one of the machine learning algorithms, and start to feed the algorithm with the prepared ***training data***. This is what we call the ***training process***.

Once we obtain our model at the end of the training process, we then test the model with the reserved ***testing data***. This is what we call the ***testing process***.

It is rarely the case that we are happy with our first trained model. One would then go back to the training process and tune some parameters that are exposed by the model that we selected. This is what we called the ***hyper-parameter tuning***. The reason that it is highlighted as 'hyper' is because the parameters that we tune are the outermost interface that we interact with the model, which would eventually have impacts on the underlying parameters of the model. For example, for the decision tree model, one of its hyper-parameters would be the maximum height of the tree. Once set manually before the training, it would limit the number of branches and leaves that a decision tree can grow at the end, which are the underlying parameters that a decision tree model consists of.

As one can see, the steps involved in the machine learning workflow form a ***cycle*** with a focus on the data.

 Underfitting VS. Overfitting

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

For supervised learning algorithms, *e.g.* classification and regression, there are two common cases where the generated model does not fit well the data: underfitting and overfitting.

An important measurement for supervised learning algorithms, is the ***generalization***, which measures how well that a model derived from the training data can predict the desired attribute of the unseen data. When we say a model is underfitting or overfitting, it implies that the model does not generalized well to the unseen data.

A model that fits well with the training data does not necessarily imply that it would generalize well to the unseen data. Because *1)*. the training data are just samples we collect from the real world, which represents only a proportion of reality. It could be the case that the training data is simply not representative, thus even the model fits perfectly the training data, it would not fit well with the unseen data. *2)*. the data that we collect contains noises and errors inevitably. The model that fits perfectly with the data, would also capture the undesired noises and errors by mistake, which would eventually lead to bias and errors in the prediction for the unseen data.

Before we dive down into the definition of underfitting and overfitting, here we show some examples of what underfitting and overfitting models look like, in the task of classification.

Chart, scatter chart

Description automatically generated

***Underfitting***

An underfitting model is the one that does not fit well with the training data, *i.e.* significantly deviated from the ground truth.

One of the causes of underfitting could be that the model is over-simplified for the data, therefore it is not capable to capture the hidden relationship within the data. As one can see from the above graph No. **(1)**, in order to separate the samples, *i.e.* classification, a simple linear model (a line) is not capable to clearly draw the boundary among the samples of different categories, which results in significant misclassification.

As a countermeasure to avoid the above cause of underfitting, one can choose an alternative algorithm that is capable to generate a more complex model from the training data set.

***Overfitting***

An overfitting model is the one that fits well with the training data, *i.e.* little or no error, however it does not generalized well to the unseen data.

Contrary to the case of underfitting, an over-complicated model that is able to fit every bit of the data, would fall into the traps of noises and errors. As one can see from the above graph No. **(3)**, the model managed to have less misclassification in the training data, yet it is more likely that it would stumble on the unseen data.

Similarly with the underfitting case, to avoid the overfitting, one can try out another algorithm that could generate a simpler model from the training data set. Or more often, one stays with the original algorithm that generated the overfitting model, but adds a regularization term to the algorithm, *i.e.* penalizing the model that is over-complicated so that the algorithm is steered to generate a less complicated model while fitting the data.

 Bias VS. Variance

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

In this article, we will talk about the notions of bias and variance, which provides another perspective to look at the phenomenon of **underfitting** and **overfitting** that we discussed before.

Before we start, we would like to put a statement below. It probably does not ring a bell for you at the moment, but hopefully you would get the answer yourself once you go through the article.

Bias is a learner’s tendency to consistently learn the same wrong thing. Variance is the tendency to learn random things unrelated to the real signal **[1]**.

In order to define bias and variance, we need to first define the notion of main prediction **[2]**. For those of you who are already familiar with the concepts of model and loss function, you can skip the part of definitions, and jump directly to the section of Main Prediction. For the sake of completeness, we first give the definitions of basic concepts before diving into the main subject of this article.

***Definitions***

Given a training set s={(x1⃗,t1),....(xn⃗,tn)}s={(x1​

​,t1​),....(xn​​,tn​)}, a learner (a machine learning algorithm) produces a model FF. Given a test example xk⃗xk​​, this model produces a **prediction** yk=F(xk⃗)yk​=F(xk​​). Each sample in the training set consists of two elements, xi⃗xi​

​ is a vector of attributes associated with the sample, and titi​ is the target attribute to predict for the sample.

For example, a train sample might look like: xi⃗xi​

​= (type='appartment', location='LA', surface='120m2'), titi​ = (price='420,000$'). The task of a learner is then to predict the price of an estate given its properties.

Given an training sample (xi⃗,ti)(xi​

​,ti​), the learner produces a prediction as F(xi⃗)F(xi​​), we then define the **loss function** L(F(xi⃗),ti)L(F(xi​​),ti​) as the cost that is incurred by the difference between the prediction F(xi⃗)F(xi​​) and the true value titi​ associated with the sample. The larger the difference, the bigger the loss. If the target attribute titi​ is of numerical value, a common loss function would be a **square error**, *i.e.* L(F(xi⃗),ti)=(F(xi⃗)−ti)2L(F(xi​​),ti​)=(F(xi​​)−ti​)2. Following the above example, if a model FF produces the price of the above example xi⃗xi​

​ as '410,000$', then the result of the loss function would be (420,000−410,000)2=108(420,000−410,000)2=108.

***Main Prediction***

Given a loss function LL and sets of training set S={s1,s2...sn}S={s1​,s2​...sn​}, the main prediction for a learner is defined as ym=argminy′ES(L(y,y′))ym​=argminy′​ES​(L(y,y′)).

For each training set sisi​, we train a model FiFi​ with a given learner. For a given training sample, we then produce a set of predictions Y={y1,y2...yn}Y={y1​,y2​...yn​} with yiyi​ corresponding to the result produced by the model FiFi​. The ***main prediction*** ymym​ is the prediction y′y′ whose average loss with regards to all the predictions in YY is minimum (*i.e.*, it is the prediction that “differs least” from all the predictions in YY according to LL). **[2]**

In a particular case, when the loss function is square error, *i.e.* L(y,y′)=(y−y′)2L(y,y′)=(y−y′)2, the main prediction with regards to the entire training sets SS, is then **the mean of the predictions**, *i.e.* ym=ES(Y)=1n∑i=1nyiym​=ES​(Y)=n1​∑i=1n​yi​. One can refer to the paper **[2]** in the references for the deduction process.

We can interpret the main prediction as the expect answer for a given training sample, from a given learner (a machine learning algorithm).

To better understand the concept of the main prediction, let's imagine that a learner is playing a dart-throwing game, *i.e.* a learner is a player. Each time a player throws a dart, it involves two corresponding activities: *1)*. the player poses and aims, *i.e.* the learner trains a model from a given training dataset. *2).* the player throws the dart, *i.e.* we say that the model trained by the learner produces a prediction. Intuitively, the bullseye is the target of the prediction. The closer a dart to the bullseyes, the better the player (learner) is.

Then each time before the player throws a dart, we can ask ourselves a question: *how many points that the player would score*. And this is where the concept of the main prediction comes into the picture. As it turns out that the best guess that one can bet on is the main prediction of this learner (player), which could be approximated by the average score that the player has achieved during all past games. For example, we can bet that a dart out of the hand of a good player would not stray too far away from the bullseye.

Intuitively, we can consider the main prediction as the general ***tendency*** (performance) of a learner, *i.e.* expected points that a player can score in a game.

With the notion of the main prediction, we now can define the notions of bias and variance.

***Bias***

The bias of a learner (algorithm) on the example (xi⃗,ti)(xi​

​,ti​) is defined as B(xi⃗)=L(ym,ti)B(xi​

​)=L(ym​,ti​), where ymym​ is the main prediction, titi​ is the target value, and LL is the loss function.

In a word, the bias of a learner, with regards to an example, is the loss incurred by the difference between the main prediction (the general tendency) and the actual value of the target attribute (target value).

The closer the main prediction to the target value, the less the loss. The more likely a learner produces a model that gives the right prediction, the less bias the learner has.

According to the definition, the main prediction of a learner is the mean prediction of all models that are produced by the learner from various training sets. Therefore, given a learner, with infinite training sets, one can assume that the main prediction would converge to a constant value that is linked with the nature of the learner. When the bias of a learner is zero, we can say that the learner can produce a number of models whose mean prediction is the desired target value **[2]**.

If a learner has a relatively high bias and the learner does indeed learn something from the data, then we can say that the model resulted from the learner has a high tendency to produce an erroneous prediction.

***Variance***

The variance of a learner (algorithm) on the example (xi⃗,ti)(xi​

​,ti​) is defined as V(xi⃗)=ES(L(ym,y))V(xi​

​)=ES​(L(ym​,y)), where ymym​ is the main prediction, yy is a prediction produced from a model that is trained on a training set s∈Ss∈S, LL is the loss function, and ESES​ is the average function over the list of loss values.

In a word, V(xi⃗)V(xi​

​) measures the loss incurred by its fluctuation around the main prediction in response to different training sets.

The more stable the performance of a learner, the less its variance. In the game of dart, a learner with *high variance* corresponds to a lousy player who rarely lands the dart in the same place. On the other hand, a learner with low variance could correspond to a decent player who rarely misses the bullseye.

The variance is independent of the true value of the predicted variable, and is zero for a learner that always makes the same prediction regardless of the training set **[2]**.

If a learner has high variance, then we would know that the learner is highly sensitive to the training data set, *i.e.* a minor noise in the training data could lead to an ill-formed model that produces wrong predictions.

***Bias-Variance Coordinate***

The bias and variance can serve as metrics to evaluate the performance of a learner.

Below we draw a plot with two dimensions: the X dimension indicates the variance of a learner, and the Y dimension indicates the bias of a learner. Following the dart game example that we discussed in previous sections, the learner assumes the role of a dart player. Each dot on the board corresponds to a prediction made by the learner. The closer is the dot to the bullseye, the closer is the prediction to the target value. We then can classify the learners into four different types as follows:

Diagram, shape, circle

Description automatically generated

- (1). An ***ideal learner*** would situate at the low-left part of the plot, where we find the low bias as well as low variance. This is the decent dart player that rarely misses the bullseye. A good player never makes mistakes, *i.e.* a good learner always makes good predictions.

- (2). Now, moving to the right-hand side of the ideal learner, we have a ***fair learner*** who manages to score some points (*i.e.* low bias), yet the darts are all over the places (*i.e.* high variance). The learners that are situated in this area are usually complex algorithms that could manage to train some decent models sometimes, but in general, the performance of the model is not too promising. The case where we do not obtain a good model, is also called **overfitting**, *i.e.* the model pays too much attention to the irrelevant noise.

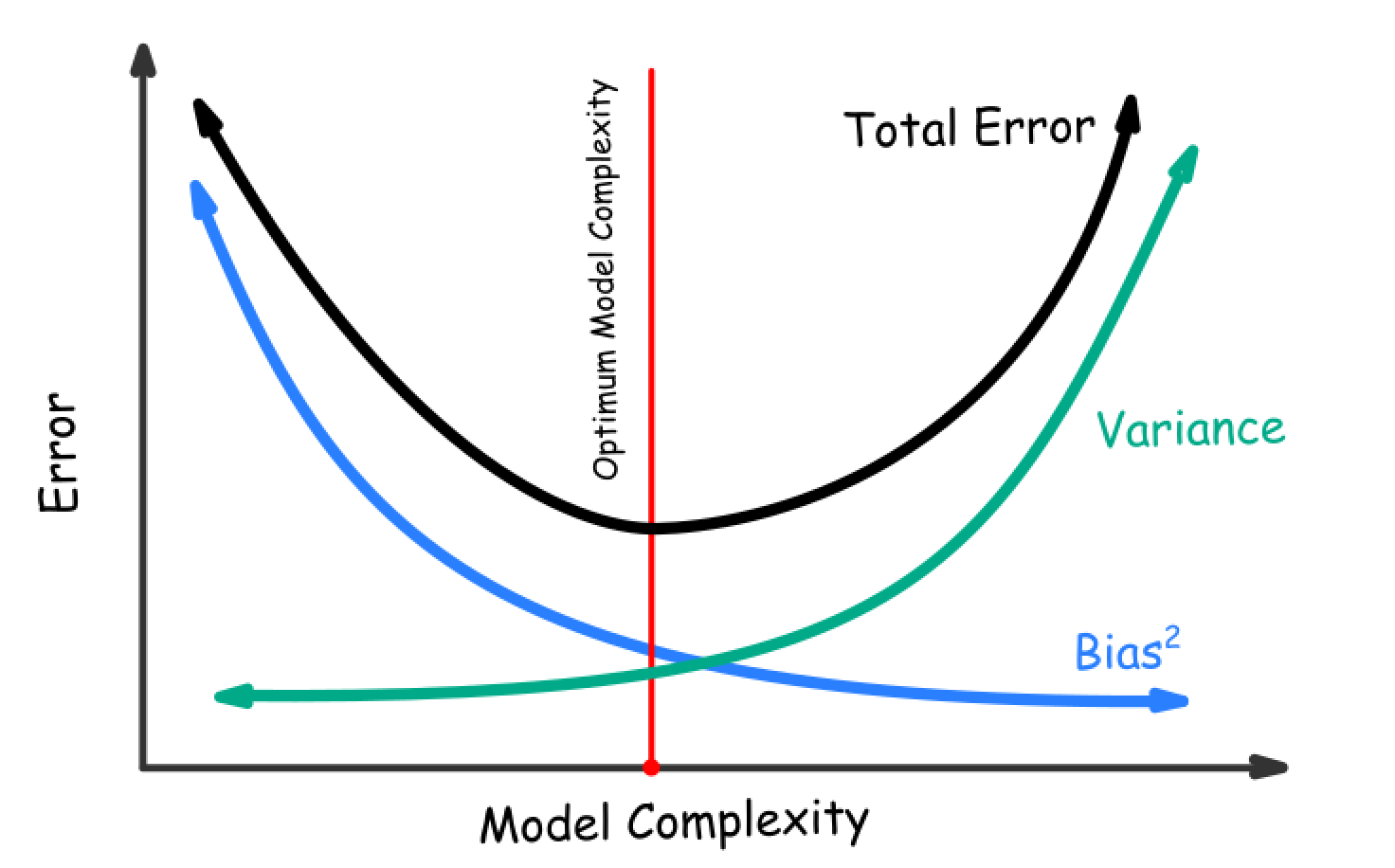
- (3). We then move to the up-right part of the plot, here we find the ***terrible learner***, who has both high bias and high variance. The terrible learner is not able to extract information from the data and basically learns nothing. The prediction that learner produces is not relevant (high bias), and what's even worse, the learner is not consistent with its strategy but making a random guess (high variance).

- (4). Right next to the terrible learner, we find the ***naive learner***, who has high bias yet low variance. The naive learner often adopts some straightforward strategies, which could explain why it produces stable outputs (low variance). But the strategy is too simple to capture the essential information from the data, which results in producing a **underfitted** model.

***Bias-Variance Tradeoff***

Bias is reduced and variance is increased in relation to the complexity of the models produced by a learner.

The correlation between the model complexity and bias-variance can be described in the following graph:



One can draw a few information from the above graph:

* When the model becomes more complex, it could potentially fit better the training data set, as a result, the bias is reduced.
* Meanwhile, when the model becomes more complex, the variance increases, since the model becomes more sensitive to the noise in the data.
* As one can see, the total error of a model is correlated to the components of Bias2Bias2 and VarianceVariance.

Indeed, when the loss function of the model is square error, its total error can be decomposed as:

Err(xi)=Bias2+Variance+Irreducible ErrorErr(xi​)=Bias2+Variance+Irreducible Error

We skip the deduction process for the above formula, but one can refer to [the page here](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff#Derivation) for more details.

From the above definitions, we can see that a good learner should have low bias and also low variance. However, it is difficult to have them both, since these two properties are contradicting to each other. As a result, one needs to find the sweet spot on the model complexity in order to obtain the best result.

Given a learner, often one can adjust its bias and variance, by tuning the parameters involved in the learner.

For example, if we construct a decision tree for a classification problem, without any constraint the decision tree could overgrow itself to fit every bit of the training data set, including the noisy data. As a result, we might obtain a decision tree model with a low bias for the given training data set. However, it could end up with high bias as well as high variance for the unseen data sets, since it ***overfits*** the training data set. To mitigate the problem, one can impose some constraints to limit the growth of a decision tree, *e.g.* setting the max depth a tree can grow, which might result in higher bias in the training data set. In exchange, we could obtain a model with lower variance in the unseen data set, as well as lower bias, since the model is trained to be more generalized.

***Conclusion***

In this article, we clarify the notions of bias and variance, which are the characteristics associated with a learner (*i.e.* a machine learning algorithm). These characteristics are exhibited in the scenarios of applying the learner to solve a particular machine learning problem. Therefore, to measure the bias and variance, one should apply the learner in a set of training datasets for a given problem.

The bias and variance of a learner are defined under the context of the problem, *i.e.* the training data sets, the learning task and the loss function *etc.*which vary from problem to problem.In general, it is not fair to say that a learner is of high bias or of high variance, without mentioning the context. For example, the linear regression algorithm might be a terrible learner (high bias and high variance) for the image classification problems, while it could excel in some simple classification problems whose data sets contain only a few attributes.

Given a problem, the bias and variance of a learner is often not fixed either. One could tune the parameters with the learner to strike a balance between the bias and variance. Overall, when the complexity of the models produced by a learner increases, the bias of the learner decreases, while its variance increases.

Finally, let's conclude the article with the statement that we put at the beginning:

Bias is a learner’s tendency to consistently learn the same wrong thing. Variance is the tendency to learn random things unrelated to the real signal [1].

Now, if you have reached this point, you should be able to make sense of the statement. If you have any doubts or questions, feel free to pose in the [Discussion](https://leetcode.com/discuss/explore/machine-learning-101) forum.

***Further Readings***

- [1]. [*A Few Useful Things to Know about Machine Learning*](https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf). Pedro Domingo. 2012. University of Washington.

- [2]. [*A Unified Bias-Variance Decomposition and its Applications*](https://homes.cs.washington.edu/~pedrod/papers/mlc00a.pdf). Pedro Domingo. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 231–238, Stanford, CA, 2000. Morgan Kaufmann.

 Why Machine Learning

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

After the previous chapters, one should be able to tell in general what Machine Learning (ML) algorithms are, and should have a brief idea on how to apply ML in a project.

Now, in this chapter, it would be the right moment to reflect a bit more on the question: *why do we need ML algorithms ?*

First of all, let's acknowledge that at this moment (2018) we do need the ML algorithms in many aspects of our lives. Noticeably, it is omnipresent in the Internet services (*e.g.* social networking, search engine *etc.*) that we are indulging daily. In fact, as revealed in a [recent paper from Facebook](https://research.fb.com/publications/applied-machine-learning-at-facebook-a-datacenter-infrastructure-perspective/), ML algorithms become so important that Facebook started to redesign their datacenters from hardware to software, to better cater to the requirements of applying ML algorithms.

*"At Facebook, machine learning provides key capabilities in driving nearly all aspects of user experiences... Machine learning is applied pervasively across nearly all services."*

Here are a few examples of how ML is applied in Facebook:

* Ranking of stories in the News Feed is done via ML.
* When, where and who to display Ads is determined by ML.
* The various search engines (*e.g.* photos, videos, people) are each powered by ML.

One could easily identify many other scenarios where ML is applied, in the services (*e.g.* Google search engine, Amazon e-commerce platform) that we are using nowadays. The ubiquitous presence of ML algorithms becomes a norm in the modern life, which justifies its raison d'etre at least for the moment and the near future to come.

**Why ML ?**

ML algorithms exist, because they can solve problems that non-ML algorithms are not able to, and because they offer advantages that non-ML algorithms do not have.

One of the most important characteristics that tells a ML algorithm apart from non-ML ones, is that it decouples the model from the data so that a ML algorithm can adapt to different business scenarios or the same business case but with different contexts. For instance, a classification algorithm can be applied to tell if there is a face shown on a photo. It can also be applied to predict if users are going to click on an Ads. In the case of face detection, the same classification algorithm can be used to train a model that can tell whether or not there is a face presented on a photo, as well as training another model that tell precisely who is presented on the photo.

Through the separation of model and data, ML algorithms can solve many problems in a more *flexible*, *generic* and *autonomous* manner, *i.e.* much like a human, the ML algorithms seem to be able to learn from the environment (*i.e.* the data) and adjust its behaviors (*i.e.* the model) accordingly in order to solve a specific problem. Without explicitly coding the rules (*i.e.* the model) in the ML algorithms, we construct a sort of meta-algorithm that is able to *learn* the rules/patterns from the data, in a supervised or even unsupervised manner.

 ML, Silver Bullet ?

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

Once one starts to learn various kinds of Machine Learning (ML) algorithms, and how versatile they are to handle challenging tasks such as image recognition and language translation *etc*, one might be indulged to apply ML to every problem that they face, regardless of whether it fits or not. Because often the case, once one acquires a hammer, every problem might seem to be just another nail.

As a result, in this section, we would like to stress on some negative notes of ML. Like all other solutions, ML is **no silver bullet**.

Like humans, ML models make mistakes.

For instance, one might notice that sometimes Facebook fails to tag a face from a photo. Unfortunately, people seem to accept the current state-of-the-art that ML algorithm is usually not 100%100% accurate. One can probably defend for ML algorithms, with the argument that the problems that ML deals with are indeed difficult, even for humans, *e.g.* image recognition. However, it is contrasting to the general conception that machines make no mistake or at least less than humans. For a moment (before 2012), people could easily claim the championship of [ImageNet](http://www.image-net.org/) challenge with a model of 75%75% accuracy. One should bear in mind that the challenge was considered to the Olympic game in the domain of image recognition. So one can consider the results in ImageNet challenge as the *state-of-the-art* in the domain. Yet till now (2018), still no model can achieve 100%100% of accuracy. In general, a ML model that can reach ~80%80% accuracy, is considered to have a decent performance. ***Therefore, in the scenarios where the accuracy of the algorithm is critical, one should carefully examine their decision of adopting ML algorithms.***

It is hard, if not impossible, to correct the mistakes made by ML, in the case-by-case manner.

One might wonder, if we consider each mistake made by a ML model as a bug in the software, can't we just correct them one by one so that we can boost the accuracy step by step? The answer is no. The reason is twofold: *1).* In general, one does not explicitly manipulate a ML model, but apply a ML algorithm with a given data to generate a model. To improve a model, we either improve the algorithm or the quality of data, without modifying the model directly. *2).* Even we can manipulate a generated ML model afterward, it is not intuitive how one can change the output of the ML model in certain 'erroneous' cases, without impacting the other correct cases. For instance, for a decision-tree model, the output of the model is the conjunction of branching conditions at each node, following the path from root to leaf. One can change certain branching conditions in the nodes to alter the decision of erroneous cases. However, this change would also impact the outputs for every case that passes through the modified nodes. In summary, one can not treat the mistakes made by a ML model simply as bugs in the software. ***It requires a holistic approach to improve the model, rather than patching the model case by case.***

It is hard, if not impossible, to reason about certain ML models.

So far, one has learned that ML model makes mistakes and it is hard to correct the mistakes case by case. Perhaps things aren't so bad, since at least we could explain why it makes mistakes, such as the decision-tree model. Yet, in some cases, particularly for the ML models with neural networks, we cannot really reason about the models, *i.e.* it is hard to interpret the model, to identify the key parameters within a model. For instance, there is a state-of-the-art neural network model called [ResNet](https://arxiv.org/abs/1512.03385) **[1]** which achieves 96.43%96.43% accuracy in the [ImageNet](http://www.image-net.org/) challenge. The ResNet-50 model consists of 50 layers of neurons, including 25.6 million of parameters in total. Each of the parameters contributes to the final output of the model. Either the output is correct or not, it is the millions of parameters behind the model that accounts for. It is hard to attribute any logic to each of the parameters individually. ***Therefore, in the scenarios where one looks for interpretability for the model, one should think over the decision to apply any neural-network-based ML model.***

So to summarise, ML is no silver bullet, because it is often not 100%100% accurate, and we cannot correct the ML model case by case, and in certain cases we cannot even reason about the ML models.

***Further Readings***

- [1]. [*ResNet: Deep Residual Learning for Image Recognition*](https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf). He *et al.* CVPR 2016 Las Vegas, NV, USA.

- [2]. [*LIME: Explaining the Predictions of Any Classifier.*](https://www.kdd.org/kdd2016/papers/files/rfp0573-ribeiroA.pdf) Ribeiro *et al.* KDD 2016 San Francisco, CA, USA.

**Data Scientists and Machine Learning**

There are two categories of challenges that machine learning practitioners and data scientists face: bad algorithms and bad data.

**Bad algorithms** can come from:

* Poor feature engineering
* Poor feature selection or poor feature extraction
* Overfitting the training data — the model performs well on the training data alone but doesn't generalize well to unseen data
* Underfitting the training data — your model is too basic to learn the underlying structure of the data
* Failure to properly cross-validate — cross-validation gives you sensible hyperparameters, but you can easily misapply this technique — you'll learn more about this later

**Bad data** can arise for many reasons. These include:

* Nonrepresentative training data — the training data is not representative of the new cases to which you want to generalize
* Sampling bias exhibited by the data
* Poor-quality data — errors, outliers, and noise (due to poor-quality measurements).

We will look at some examples of these challenges in this subunit.

[[](https://williamkoehrsen.medium.com/?source=post_page-----d05dd7e19765--------------------------------)](https://williamkoehrsen.medium.com/?source=post_page-----d05dd7e19765--------------------------------)

[Will Koehrsen](https://williamkoehrsen.medium.com/?source=post_page-----d05dd7e19765--------------------------------)

Jan 28, 2018

·

A picture containing water, sunset, outdoor, clouds

Description automatically generated

# Overfitting vs. Underfitting: A Complete Example

**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 neural 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 want to develop a model, it’s more productive and less frustrating to work 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 until you can create a useful machine learning system and, just as importantly, understand how it works.

Diagram, engineering drawing

Description automatically generated

Out of simple ideas come powerful systems ([Source](https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721))

This post walks through a complete example illustrating an essential data science building block: the underfitting vs overfitting problem. We’ll explore the problem and then implement a solution called cross-validation, another important principle of model development. If you’re looking for a conceptual framework on the topic, [see my previous post](https://towardsdatascience.com/overfitting-vs-underfitting-a-conceptual-explanation-d94ee20ca7f9). All of the graphs and results generated in this post are written in Python code which is on [GitHub](https://github.com/WillKoehrsen/Data-Analysis). 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: what is a model? A model is simply a system for mapping 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 theory about a problem: there is some connection between the square footage and the price and we make a model to learn that relationship. Models are useful because we can use them to predict the values of outputs for new data points given the inputs.

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 and learns how to map the former to the latter. 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 on 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 x-values (features) and y-values (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 to a trend. There is always noise or other variables in the relationship we cannot measure. In the house price example, the trend between area and price is linear, but the prices do not lie exactly on a line because of other factors influencing house prices.

Chart

Description automatically generated

Example of a real-world relationship ([Source](https://rstudio-pubs-static.s3.amazonaws.com/64900_3f884598a6e84ea3aacef364cbfc54ba.html))

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.

Chart, scatter chart

Description automatically generated

Data and True Generating Funtion

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 ([see code for details](https://github.com/WillKoehrsen/Data-Analysis/blob/master/over_vs_under/Over%20vs%20Under%20Fitting%20Example.ipynb)). During training, we want 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 model, we will use an [extension of linear regression called polynomial regression](https://onlinecourses.science.psu.edu/stat501/node/324) to learn the relationship between x and y. 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.



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. Once the model has learned the beta values, we can plug in any value for x and get a corresponding prediction for y. A polynomial is defined by its order, which is the highest power of x in the equation. A straight line is a polynomial of degree 1 while a parabola has 2 degrees.

Diagram

Description automatically generated

Polynomials of Varying Degree ([Source](http://brandon.ai/2017/01/27/multivariate-polynomial-regression.html))

# Overfitting vs. Underfitting

The problem of [Overfitting vs Underfitting](https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html) finally appears when we talk about the polynomial degree. The degree represents how much flexibility is in the model, with a higher power allowing the model freedom to hit as many data points as possible. An underfit model will be less flexible and cannot account for the data. The best way to understand the issue is to take a look at models demonstrating both situations.

First up is an underfit model with a 1 degree polynomial fit. In the image on the left, model function in orange is shown on top of the true function and the training observations. On the right, the model predictions for the testing data are shown compared to the true function and testing data points.

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Underfit 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! Instead, the model has high bias, which means it makes a strong assumption about the data. For this example, the assumption is that the data is linear, which is evidently quite wrong. When the model makes test predictions, the bias 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.

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

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Overfit 25 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? Further, the model has a great score on the training data because it gets close to all 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 predictions on the test set are better than the one degree model, but the twenty five degree model still 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! How can we find a balanced 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 this cannot generalize well to testing data. Fortunately, there is a well-established data science technique for developing 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 consequences in our example from poor test performance, but in a real application 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 pre-test to use for model optimization and evaluate. 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 split the training set into a number of subsets, called folds. Let’s use five 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 optimize the model before deployment without having to use additional data.

Chart, bar chart, box and whisker chart

Description automatically generated

Five-Fold Cross-Validation ([Source](https://stackoverflow.com/questions/31947183/how-to-implement-walk-forward-testing-in-sklearn))

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. I choose to use models with degrees 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.

Table

Description automatically generatedChart, line chart

Description automatically generatedCross Validation Results

The cross-validation error with the underfit and overfit models is off the chart! A model with 4 degrees appears to be optimal. To test out the results, we can make a 4-degree model and view the training and testing predictions.

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Balanced Four degree polynomial model on training (left) and testing (right) datasets

There is 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 and testing curves. These show the model setting we tuned on the x-axis and both the training and testing error on the y-axis. **A model that is underfit will have high training and high testing error while an overfit model will have extremely low training error but a high testing error.**

Chart, line chart

Description automatically generated

Training and Testing Curves

This graph nicely summarizes the problem of overfitting and underfitting. As the flexibility in the model increases (by increasing the polynomial degree) the training error continually decreases due to increased flexibility. However, the error on the testing set only decreases as we add flexibility up to a certain point. In this case, that occurs at 5 degrees As the flexibility increases beyond this point, the training error increases because the model has memorized the training data and the noise. Cross-validation yielded the second best model on this testing 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 data analysts. In my lab, I have seen many grad students fit a model with extremely low error to their data and then eagerly write a paper with the results. Their model looks great, but the problem is they never even used a testing set let alone a validation set! The model is nothing more than an overfit representation of the training data, a lesson the student soon learns 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 evaluation 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 hopefully you now have an idea of the basics of modeling, 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 [@koehrsen\_will](https://twitter.com/koehrsen_will). I would like to thank the contributors to Scikit-Learn for their [excellent example on this subject](http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html).

**QUIZ**

According to Arthur Samuel, machine learning algorithms:

a. Enable the computers to learn from data, and even improve themselves, without being explicitly programmed.

b. Enable the computers to learn from data and improve themselves through explicit programming.

Learn from experience E, with respect to some task T and some performance P, if their performance on T as measured by P improves with E.

Are defined correctly by none of the above.

What is machine learning according to Arthur Samuel?

It was defined in the 1950s by AI pioneer Arthur Samuel as “**the field of study that gives computers the ability to learn without explicitly being programmed**.”

Which of the following claims is not true?

a. Machine learning shines for problems that are too complex for traditional approaches.

b. Machine learning is great for problems where no known algorithm can solve them.

c. Machine learning is great for problems where alternate solutions would involve the writing and maintenance of long lists of complex rules.

**d. Machine learning is great for even the simplest data analysis and data science problems**.

What is supervised learning?

a. Machine learning such that the training data is unlabeled.

b**. Machine learning such that the training data is labeled.**

c. Machine learning involving the writing and maintenance of long lists of complex rules.

d. None of the above

With Batch learning:

a. The system can learn incrementally and can be trained using just a small subset of the available data.

b. Training is done online and is computationally cheap.

**Training takes a lot of time and is computationally expensive. It is done offline.**

The training is divided into learning steps that are individually fast and cheap.

Batch size is a term used in machine learning and **refers to the number of training examples utilized in one iteration**. The batch size can be one of three options: batch mode: where the batch size is equal to the total dataset thus making the iteration and epoch values equivalent.

Why do we use batches in machine learning?

Batch Size is among the important hyperparameters in Machine Learning. It is the hyperparameter that defines the number of samples to work through before updating the internal model parameters. **It can one of the crucial steps to making sure your models hit peak performance**

The learning rate determines:

a. The rate at which training data is ingested.

b. **The rate at which the machine should adapt to changing data.**

c.The number of solutions output by the machine learning algorithm.

The amount of memory used by the machine learning algorithm.

Instance-based learning requires a measure of similarity

**TRUE**

FALSE

The main concerns with machine learning can be categorized into two sets: worries around ‘bad algorithms’ and worries about ‘bad data’

**TRUE**

FALSE

If the sample from which the training set is derived is too small:

a. **We will have sampling noise.**

b. We will have sampling bias.

c. We will have both.

d. We will have neither.

Under what conditions does a machine learning algorithm overfit?

a. When it generalizes well but performs poorly on the training data.

b. When garbage data is used in training, so garbage outputs are produced.

c. **When it performs well on the training data but generalizes poorly.**

d. We will have neither.

**Supervised Learning**

**Learning Objectives**

* Develop an understanding of supervised learning and its common applications
* Be able to perform regression and classification techniques to solve real-world problems

**Work to Complete**

In this unit, you'll:

* Complete a logistic regression case study
* Complete a Decision Trees case study
* Complete a Random Forest case study
* Complete a Gradient Boosting case study

Supervised Learning is the bread and butter of machine learning. Consider a teacher facilitating a student’s learning process by showing them what to do and then having them do that task repeatedly. Supervised learning is similar — a data scientist uses a training dataset to teach an algorithm how to use a mapping function from an input to an output. The algorithm learns to approximate the mapping function so well that the algorithm can predict output variables for that data. Put another way — you’ll use supervised learning when you give your machine labeled training data and encode procedures for the machine to learn to assign those labels itself.

Let’s take a look at a simple example. If you wanted to predict if a cat is a particular breed, you could load up a ton of cat information, such as height, body hair length, nose color, and more. These characteristics are called “features.” All of these features combined make up the training data that your algorithm can draw from — the algorithm can predict a cat’s breed based on the cat’s height, body hair length, nose color, etc.

This unit will explore several supervised learning methods, including logistic regression, decision trees, random forest, gradient descent, series analysis, and forecasting.

# [Choosing a Machine Learning Classifier](http://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/)

How do you know what machine learning algorithm to choose for your classification problem? Of course, if you really care about accuracy, your best bet is to test out a couple different ones (making sure to try different parameters within each algorithm as well), and select the best one by cross-validation. But if you’re simply looking for a “good enough” algorithm for your problem, or a place to start, here are some general guidelines I’ve found to work well over the years.

# How large is your training set?

If your training set is small, high bias/low variance classifiers (e.g., Naive Bayes) have an advantage over low bias/high variance classifiers (e.g., kNN), since the latter will overfit. But low bias/high variance classifiers start to win out as your training set grows (they have lower asymptotic error), since high bias classifiers aren’t powerful enough to provide accurate models.

You can also think of this as a generative model vs. discriminative model distinction.

# Advantages of some particular algorithms

**Advantages of Naive Bayes:** Super simple, you’re just doing a bunch of counts. If the NB conditional independence assumption actually holds, a Naive Bayes classifier will converge quicker than discriminative models like logistic regression, so you need less training data. And even if the NB assumption doesn’t hold, a NB classifier still often does a great job in practice. A good bet if want something fast and easy that performs pretty well. Its main disadvantage is that it can’t learn interactions between features (e.g., it can’t learn that although you love movies with Brad Pitt and Tom Cruise, you hate movies where they’re together).

**Advantages of Logistic Regression:** Lots of ways to regularize your model, and you don’t have to worry as much about your features being correlated, like you do in Naive Bayes. You also have a nice probabilistic interpretation, unlike decision trees or SVMs, and you can easily update your model to take in new data (using an online gradient descent method), again unlike decision trees or SVMs. Use it if you want a probabilistic framework (e.g., to easily adjust classification thresholds, to say when you’re unsure, or to get confidence intervals) or if you expect to receive more training data in the future that you want to be able to quickly incorporate into your model.

**Advantages of Decision Trees:** Easy to interpret and explain (for some people – I’m not sure I fall into this camp). They easily handle feature interactions and they’re non-parametric, so you don’t have to worry about outliers or whether the data is linearly separable (e.g., decision trees easily take care of cases where you have class A at the low end of some feature x, class B in the mid-range of feature x, and A again at the high end). One disadvantage is that they don’t support online learning, so you have to rebuild your tree when new examples come on. Another disadvantage is that they easily overfit, but that’s where ensemble methods like random forests (or boosted trees) come in. Plus, random forests are often the winner for lots of problems in classification (usually slightly ahead of SVMs, I believe), they’re fast and scalable, and you don’t have to worry about tuning a bunch of parameters like you do with SVMs, so they seem to be quite popular these days.

**Advantages of SVMs:** High accuracy, nice theoretical guarantees regarding overfitting, and with an appropriate kernel they can work well even if you’re data isn’t linearly separable in the base feature space. Especially popular in text classification problems where very high-dimensional spaces are the norm. Memory-intensive, hard to interpret, and kind of annoying to run and tune, though, so I think random forests are starting to steal the crown.

# But…

Recall, though, that better data often beats better algorithms, and designing good features goes a long way. And if you have a huge dataset, then whichever classification algorithm you use might not matter so much in terms of classification performance (so choose your algorithm based on speed or ease of use instead).

And to reiterate what I said above, if you really care about accuracy, you should definitely try a bunch of different classifiers and select the best one by cross-validation. Or, to take a lesson from the Netflix Prize (and Middle Earth), just use an ensemble method to choose them all.

# 8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset

By [Jason Brownlee](https://machinelearningmastery.com/author/jasonb/) on August 19, 2015 in [Imbalanced Classification](https://machinelearningmastery.com/category/imbalanced-classification/)

Last Updated on August 15, 2020

Has this happened to you?

You are working on your dataset. You create a classification model and get 90% accuracy immediately. “Fantastic” you think. You dive a little deeper and discover that 90% of the data belongs to one class. Damn!

This is an example of an imbalanced dataset and the frustrating results it can cause.

In this post you will discover the tactics that you can use to deliver great results on machine learning datasets with imbalanced data.

**Kick-start your project** with my new book [Imbalanced Classification with Python](https://machinelearningmastery.com/imbalanced-classification-with-python/), including step-by-step tutorials and the Python source code files for all examples.

Let’s get started.

[](https://machinelearningmastery.com/wp-content/uploads/2015/08/Class-Imbalance.jpg)

Find some balance in your machine learning.  
Photo by MichaEli, some rights reserved.

## Coming To Grips With Imbalanced Data

I get emails about class imbalance all the time, for example:

I have a binary classification problem and one class is present with 60:1 ratio in my training set. I used the logistic regression and the result seems to just ignores one class.

And this:

I am working on a classification model. In my dataset I have three different labels to be classified, let them be A, B and C. But in the training dataset I have A dataset with 70% volume, B with 25% and C with 5%. Most of time my results are overfit to A. Can you please suggest how can I solve this problem?

I write long lists of techniques to try and think about the best ways to get past this problem. I finally took the advice of one of my students:

Perhaps one of your upcoming blog posts could address the problem of training a model to perform against highly imbalanced data, and outline some techniques and expectations.

## Frustration!

Imbalanced data can cause you a lot of frustration.

You feel very frustrated when you discovered that your data has imbalanced classes and that all of the great results you thought you were getting turn out to be a lie.

The next wave of frustration hits when the books, articles and blog posts don’t seem to give you good advice about handling the imbalance in your data.

Relax, there are many options and we’re going to go through them all. It is possible, you can build predictive models for imbalanced data.

### Want to Get Started With Imbalance Classification?

Take my free 7-day email crash course now (with sample code).

Click to sign-up and also get a free PDF Ebook version of the course.

## What is Imbalanced Data?

Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally.

For example, you may have a 2-class (binary) classification problem with 100 instances (rows). A total of 80 instances are labeled with Class-1 and the remaining 20 instances are labeled with Class-2.

This is an imbalanced dataset and the ratio of Class-1 to Class-2 instances is 80:20 or more concisely 4:1.

You can have a class imbalance problem on two-class classification problems as well as multi-class classification problems. Most techniques can be used on either.

The remaining discussions will assume a two-class classification problem because it is easier to think about and describe.

### Imbalance is Common

Most classification data sets do not have exactly equal number of instances in each class, but a small difference often does not matter.

There are problems where a class imbalance is not just common, it is expected. For example, in datasets like those that characterize fraudulent transactions are imbalanced. The vast majority of the transactions will be in the “Not-Fraud” class and a very small minority will be in the “Fraud” class.

Another example is customer churn datasets, where the vast majority of customers stay with the service (the “No-Churn” class) and a small minority cancel their subscription (the “Churn” class).

When there is a modest class imbalance like 4:1 in the example above it can cause problems.

### Accuracy Paradox

The [accuracy paradox](https://en.wikipedia.org/wiki/Accuracy_paradox) is the name for the exact situation in the introduction to this post.

It is the case where your accuracy measures tell the story that you have excellent accuracy (such as 90%), but the accuracy is only reflecting the underlying class distribution.

It is very common, because classification accuracy is often the first measure we use when evaluating models on our classification problems.

### Put it All On Red!

What is going on in our models when we train on an imbalanced dataset?

As you might have guessed, the reason we get 90% accuracy on an imbalanced data (with 90% of the instances in Class-1) is because our models look at the data and cleverly decide that the best thing to do is to always predict “Class-1” and achieve high accuracy.

This is best seen when using a simple rule based algorithm. If you print out the rule in the final model you will see that it is very likely predicting one class regardless of the data it is asked to predict.

## 8 Tactics To Combat Imbalanced Training Data

We now understand what class imbalance is and why it provides misleading classification accuracy.

So what are our options?

### 1) Can You Collect More Data?

You might think it’s silly, but collecting more data is almost always overlooked.

Can you collect more data? Take a second and think about whether you are able to gather more data on your problem.

A larger dataset might expose a different and perhaps more balanced perspective on the classes.

More examples of minor classes may be useful later when we look at resampling your dataset.

### 2) Try Changing Your Performance Metric

Accuracy is not the metric to use when working with an imbalanced dataset. We have seen that it is misleading.

There are metrics that have been designed to tell you a more truthful story when working with imbalanced classes.

I give more advice on selecting different performance measures in my post “[Classification Accuracy is Not Enough: More Performance Measures You Can Use](https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/)“.

In that post I look at an imbalanced dataset that characterizes the recurrence of breast cancer in patients.

From that post, I recommend looking at the following performance measures that can give more insight into the accuracy of the model than traditional classification accuracy:

* **Confusion Matrix**: A breakdown of predictions into a table showing correct predictions (the diagonal) and the types of incorrect predictions made (what classes incorrect predictions were assigned).
* **Precision**: A measure of a classifiers exactness.
* **Recall**: A measure of a classifiers completeness
* **F1 Score (or F-score)**: A weighted average of precision and recall.

I would also advice you to take a look at the following:

* **Kappa (or** [**Cohen’s kappa**](https://en.wikipedia.org/wiki/Cohen%27s_kappa)**)**: Classification accuracy normalized by the imbalance of the classes in the data.
* **ROC Curves**: Like precision and recall, accuracy is divided into sensitivity and specificity and models can be chosen based on the balance thresholds of these values.

You can learn a lot more about using ROC Curves to compare classification accuracy in our post “[Assessing and Comparing Classifier Performance with ROC Curves](https://machinelearningmastery.com/assessing-comparing-classifier-performance-roc-curves-2/)“.

Still not sure? Start with kappa, it will give you a better idea of what is going on than classification accuracy.

### 3) Try Resampling Your Dataset

You can change the dataset that you use to build your predictive model to have more balanced data.

This change is called sampling your dataset and there are two main methods that you can use to even-up the classes:

1. You can add copies of instances from the under-represented class called over-sampling (or more formally sampling with replacement), or
2. You can delete instances from the over-represented class, called under-sampling.

These approaches are often very easy to implement and fast to run. They are an excellent starting point.

In fact, I would advise you to always try both approaches on all of your imbalanced datasets, just to see if it gives you a boost in your preferred accuracy measures.

You can learn a little more in the the Wikipedia article titled “[Oversampling and undersampling in data analysis](https://en.wikipedia.org/wiki/Oversampling_and_undersampling_in_data_analysis)“.

#### Some Rules of Thumb

* Consider testing under-sampling when you have an a lot data (tens- or hundreds of thousands of instances or more)
* Consider testing over-sampling when you don’t have a lot of data (tens of thousands of records or less)
* Consider testing random and non-random (e.g. stratified) sampling schemes.
* Consider testing different resampled ratios (e.g. you don’t have to target a 1:1 ratio in a binary classification problem, try other ratios)

### 4) Try Generate Synthetic Samples

A simple way to generate synthetic samples is to randomly sample the attributes from instances in the minority class.

You could sample them empirically within your dataset or you could use a method like Naive Bayes that can sample each attribute independently when run in reverse. You will have more and different data, but the non-linear relationships between the attributes may not be preserved.

There are systematic algorithms that you can use to generate synthetic samples. The most popular of such algorithms is called SMOTE or the Synthetic Minority Over-sampling Technique.

As its name suggests, SMOTE is an oversampling method. It works by creating synthetic samples from the minor class instead of creating copies. The algorithm selects two or more similar instances (using a distance measure) and perturbing an instance one attribute at a time by a random amount within the difference to the neighboring instances.

Learn more about SMOTE, see the original 2002 paper titled “[SMOTE: Synthetic Minority Over-sampling Technique](http://www.jair.org/papers/paper953.html)“.

There are a number of implementations of the SMOTE algorithm, for example:

* In Python, take a look at the “[UnbalancedDataset](https://github.com/fmfn/UnbalancedDataset)” module. It provides a number of implementations of SMOTE as well as various other resampling techniques that you could try.
* In R, the [DMwR package](https://cran.r-project.org/web/packages/DMwR/index.html) provides an implementation of SMOTE.
* In Weka, you can use the [SMOTE supervised filter](http://weka.sourceforge.net/doc.packages/SMOTE/weka/filters/supervised/instance/SMOTE.html).

### 5) Try Different Algorithms

As always, I strongly advice you to not use your favorite algorithm on every problem. You should at least be spot-checking a variety of different types of algorithms on a given problem.

For more on spot-checking algorithms, see my post “Why you should be Spot-Checking Algorithms on your Machine Learning Problems”.

That being said, decision trees often perform well on imbalanced datasets. The splitting rules that look at the class variable used in the creation of the trees, can force both classes to be addressed.

If in doubt, try a few popular decision tree algorithms like C4.5, C5.0, CART, and Random Forest.

For some example R code using decision trees, see my post titled “[Non-Linear Classification in R with Decision Trees](https://machinelearningmastery.com/non-linear-classification-in-r-with-decision-trees/)“.

For an example of using CART in Python and scikit-learn, see my post titled “[Get Your Hands Dirty With Scikit-Learn Now](https://machinelearningmastery.com/get-your-hands-dirty-with-scikit-learn-now/)“.

### 6) Try Penalized Models

You can use the same algorithms but give them a different perspective on the problem.

Penalized classification imposes an additional cost on the model for making classification mistakes on the minority class during training. These penalties can bias the model to pay more attention to the minority class.

Often the handling of class penalties or weights are specialized to the learning algorithm. There are penalized versions of algorithms such as penalized-SVM and penalized-LDA.

It is also possible to have generic frameworks for penalized models. For example, Weka has a [CostSensitiveClassifier](http://weka.sourceforge.net/doc.dev/weka/classifiers/meta/CostSensitiveClassifier.html) that can wrap any classifier and apply a custom penalty matrix for miss classification.

Using penalization is desirable if you are locked into a specific algorithm and are unable to resample or you’re getting poor results. It provides yet another way to “balance” the classes. Setting up the penalty matrix can be complex. You will very likely have to try a variety of penalty schemes and see what works best for your problem.

### 7) Try a Different Perspective

There are fields of study dedicated to imbalanced datasets. They have their own algorithms, measures and terminology.

Taking a look and thinking about your problem from these perspectives can sometimes shame loose some ideas.

Two you might like to consider are **anomaly detection** and **change detection**.

[Anomaly detection](https://en.wikipedia.org/wiki/Anomaly_detection) is the detection of rare events. This might be a machine malfunction indicated through its vibrations or a malicious activity by a program indicated by it’s sequence of system calls. The events are rare and when compared to normal operation.

This shift in thinking considers the minor class as the outliers class which might help you think of new ways to separate and classify samples.

[Change detection](https://en.wikipedia.org/wiki/Change_detection) is similar to anomaly detection except rather than looking for an anomaly it is looking for a change or difference. This might be a change in behavior of a user as observed by usage patterns or bank transactions.

Both of these shifts take a more real-time stance to the classification problem that might give you some new ways of thinking about your problem and maybe some more techniques to try.



### 8) Try Getting Creative

Really climb inside your problem and think about how to break it down into smaller problems that are more tractable.

For inspiration, take a look at the very creative answers on Quora in response to the question “[In classification, how do you handle an unbalanced training set?](https://www.quora.com/In-classification-how-do-you-handle-an-unbalanced-training-set)”

For example:

Decompose your larger class into smaller number of other classes…

…use a One Class Classifier… (e.g. treat like outlier detection)

…resampling the unbalanced training set into not one balanced set, but several. Running an ensemble of classifiers on these sets could produce a much better result than one classifier alone

These are just a few of some interesting and creative ideas you could try.

For more ideas, check out these comments on the reddit post “[Classification when 80% of my training set is of one class](https://www.reddit.com/r/MachineLearning/comments/12evgi/classification_when_80_of_my_training_set_is_of/)“.

## Pick a Method and Take Action

You do not need to be an algorithm wizard or a statistician to build accurate and reliable models from imbalanced datasets.

We have covered a number of techniques that you can use to model an imbalanced dataset.

Hopefully there are one or two that you can take off the shelf and apply immediately, for example changing your accuracy metric and resampling your dataset. Both are fast and will have an impact straight away.

***Which method are you going to try?***

## A Final Word, Start Small

Remember that we cannot know which approach is going to best serve you and the dataset you are working on.

You can use some expert heuristics to pick this method or that, but in the end, the best advice I can give you is to “become the scientist” and empirically test each method and select the one that gives you the best results.

Start small and build upon what you learn.

**QUIZ**

What do the examples of classification use cases have in common?

a. You start with the labels, then find data

b.**You start with the data, then assign labels**

c. You pattern recognize over unlabelled data

d. None of the above

Which of the following is NOT one of the mentioned reasons that image recognition is hard for computers?

a. Occlusion

b. Illumination conditions

**Computational complexity**

d. Inter-class variation

What was the example with the apples and oranges supposed to communicate?

a. The importance of choosing the right question

b. The power of classification

c. **The importance of appropriate feature selection**

d. The importance of doing PCA properly

One good way to do feature selection is:

a. To do PCA

b. To do multidimensional scaling

**c. To do cross-validation**

d. To do data cleaning

Suppose we’re using a 1-nearest neighbor classifier. What does the Voronoi diagram tell us?

a. **What the closest cells are to each cell on the plane**

b. How accurate our classification model is

c. Which features are relevant

d. None of the above

What does it mean for a 1-nearest neighbor classifier to have a ‘rough decision boundary’?

a**. Its outliers are represented as ‘islands’**

b. Its decision boundary is jagged

c. It is inaccurate

With the same classifier, what was the test complexity for M data points, with N training samples?

a. O(1)

b. O(M)

c. O(N)

d**. O(M\*N)**

Variance is characterized by the lecturer as:

a. **How much the decision boundary changes given new data**

b. The square of the standard deviation

c. How variant the classification is made by the model

d. None of the above

Which of the following is an easy way to reduce the variance, so as to reduce the roughness of the decision boundary?

a. Increase the number of features

b. Increase the number of models

c. **Increase the number of neighbors**

d. Decrease the number of neighbors

What is the cost of reducing variance by increasing K and making the decision boundary smoother?

a. Efficiency of the algorithm

b. Monetary cost of encoding the algorithm

c. Computational memory

d. **Increased bias**

**Please watch from 41:52 to the end** of the Classification, kNN, Cross-validation, and Dimensionality Reduction Harvard lecture. The lecturer covers transmuting images into long picture vectors, the difference between L1 (Manhattan) and L2 (Euclidean) distances, the curse of dimensionality, and how to execute dimensionality reduction with both PCA and the multidimensional scaling.

To view the slides associated with this lecture, [click here](https://github.com/cs109/2015/blob/master/Lectures/09-ClassificationPCA.pdf).

**QUIZ**

With the CIFAR-10 Dataset of 60,000 images, how many images did we have in the training and test sets respectively?

a. 10,000 and 50,000

**b.50,000 and 10,000**

c. 45,000 and 15,000

d. 15,000 and 45,000

How were the images turned into high-dimensional data?

**They were converted into long vectors**

b. They were stored in giant matrices

c. They were scrambled, and encrypted

d. None of the above

What was the Manhattan (or L1) distance defined as in the picture example?

a. The relative difference between the two picture vectors

b. The Euclidean distance between the two data points

c. **The absolute difference between the two picture vectors**

d. The sum of the squared differences between each pair of data points

How should we pick K with nearest-neighbor classifiers?

a. Intuition

b. The variance equation

c. **Cross-validation**

d. None of the above

What was the problem with making the features pixels in the picture classification example?

a. **This doesn’t account for any variation in viewpoint, lighting etc.**

b. Pixels are not interesting

c. There are too many pixels to store

d. Doing so corrupts the image

SIFT features are:

a. Well-defined and don’t require projection

b. **Rotation- and scale-invariant**

c. Always selected in any proper feature selection

d. None of the above

What is the data scientific reason we don’t we just add more and more features, for any problem that we have?

a. We cannot fathom all the world’s features

b**. The curse of dimensionality**

c. Our computers lack the memory to handle so many dimensions

d. None of the above

What are two questions, relevant to dimensionality reduction, that we have to ask ourselves before doing PCA?

a. **Does the data mostly lie on a hyperplane? If so what is its intrinsic dimensionality?**

b. Is this a supervised or unsupervised problem? What features should we use?

c. How many data points are there? Why are there so many?

d. None of the above

Which step of this PCA algorithm is wrong?

a. Subtract mean from each data point

b. **Typically scale each dimension by the median CORRECT ANSWER IS VARIANCE**

c. Compute covariance matrix S (captures, for each dimension, the extent of the spread of that dimension and their cross-products)

d. Compute k largest eigenvectors of S

The difference between PCA and multidimensional scaling is that, with the latter, instead of giving the algorithm the raw data, you give it a matrix with distance vectors between your data points

**TRUE**

FALSE