**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

·

14 min read

·

# 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!”**