**Machine Learning 101**

Introduction

In this explore card, we introduce some basic concepts in the domain of machine learning.

By completing this card, you will be able to:

1. Identify different types of machine learning problems;
2. Know what a machine learning model is;
3. Know the general workflow of building and applying a machine learning model;
4. Know the advantages and disadvantages about machine learning algorithms;

Before you start, bear in mind that should you have any questions or comments, you can always pose them in the [Discussion](https://leetcode.com/discuss/explore/machine-learning-101) forum that is located at the end of the card. We'll do our best to response to you as soon as we can.

## Machine Learning - What

In this chapter, we first define the notions of Machine Learning (ML) algorithm and model. Then we clarify several types of ML algorithm, based on various criteria.

**Machine Learning Model**

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 *F*, 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, 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% accuracy in the [ImageNet visual recognition challenge](http://www.image-net.org/). Till nowadays, still, no machine learning model can claim 100% accuracy, although there are models that make fewer errors (<5%) than humans in this task.

**Supervised VS. Unsupervised**

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 *y*, 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*)≈*y***.**The target attribute {y}*y* 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](http://image-net.org/about-stats) 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)

**Classification VS. Regression**

In the previous section, we define a machine learning model as a function {F}*F*, 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 *M* with dimensions of *H*×*W* where *H* is the height of the photo in pixels and *W* 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] that indicates the intensity of color. The expected output of the model would be a binary value [1∣0], indicating whether the photo shows a cat or not. To summarize, our cat-photo-recognition model *F* can be formulated as follows:



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*∈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 {T}*T*, 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:

Text, logo

Description automatically generated

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

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

Now given the above features, if our model *F* gives a value like 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%] 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%, while model B gives the probability 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.

## Machine Learning - How

In the previous chapter, we answer the general question: what is a machine learning model ?

In this chapter, we illustrate the common workflow on how to train a machine learning model. Then we explain some of the common problems that one would come across when training a machine learning model.

**Data, Data, Data !**

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**

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**

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**

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  a learner (a machine learning algorithm) produces a model *F*. Given a test example A picture containing text

Description automatically generated, this model produces a **prediction** *yk*​=*F*(A picture containing text

Description automatically generated). Each sample in the training set consists of two elements,  is a vector of attributes associated with the sample, and *ti*​ is the target attribute to predict for the sample.

For example, a train sample might look like: ****= (type='appartment', location='LA', surface='120m2'), *ti*​ = (price='420,000$'). The task of a learner is then to predict the price of an estate given its properties.

Given an training sample A picture containing text

Description automatically generated, the learner produces a prediction as Text

Description automatically generated with low confidence, we then define the **loss function** A picture containing text, gauge

Description automatically generatedas the cost that is incurred by the difference between the prediction Text

Description automatically generated with low confidence and the true value *ti*​ associated with the sample. The larger the difference, the bigger the loss. If the target attribute *ti*​ is of numerical value, a common loss function would be a **square error**, i.e. . Following the above example, if a model *F* produces the price of the above example ****​​ as '410,000$', then the result of the loss function would be (420,000 - 410,000)^2 = 108.

### ***Main Prediction***

Given a loss function *L* and sets of training set *S*={*s*1​,*s*2​...*sn*​}, the main prediction for a learner is defined as .

For each training set *si*​, we train a model *Fi*​ with a given learner. For a given training sample, we then produce a set of predictions *Y*={*y*1​,*y*2​...*yn*​} with *yi*​ corresponding to the result produced by the model *Fi*​. The ***main prediction*** *ym*​ is the prediction *y*′ whose average loss with regards to all the predictions in *Y* is minimum (i.e., it is the prediction that “differs least” from all the predictions in *Y* according to *L*). **[2]**

In a particular case, when the loss function is square error, i.e. Icon

Description automatically generated, the main prediction with regards to the entire training sets *S*, is then **the mean of the predictions**, i.e.  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 A picture containing text

Description automatically generated is defined as Icon

Description automatically generated, where *ym*​ is the main prediction, *ti*​ is the target value, and *L* 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 A picture containing text

Description automatically generated is defined as Text

Description automatically generated with medium confidence, where *ym*​ is the main prediction, *y* is a prediction produced from a model that is trained on a training set *s*∈*S*, *L* is the loss function, and *ES*​ is the average function over the list of loss values.

In a word, A picture containing text

Description automatically generated 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:

Diagram

Description automatically generated

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 Bias2 and Variance.

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



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.

## Machine Learning - Why

In this chapter, we would like to give some examples on why the Machine Learning (ML) algorithms are important nowadays. Then we highlight some of the advantages as well as disadvantages of ML.

**Why Machine Learning**

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 ?**

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% 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% 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% of accuracy. In general, a ML model that can reach ~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" \t "_blank) **[1]** which achieves 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% 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" \t "_blank)*. 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.