

Introduction to Machine Learning and Related Concepts



Introduction to Machine Learning

Wikipedia: Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. Machine Learning algorithms build a model based on sample data, known as "training data", in order to make **predictions** or **decisions** without being explicitly programmed to do so.

Voice Assistants

Stock Price Prediction

Email Filtering

Product Recommendations

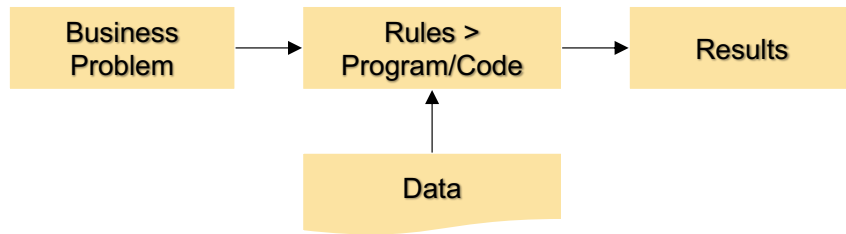
Predictive Maintenance

Self Driving Cars

**Some super cool
use cases of
Machine Learning**

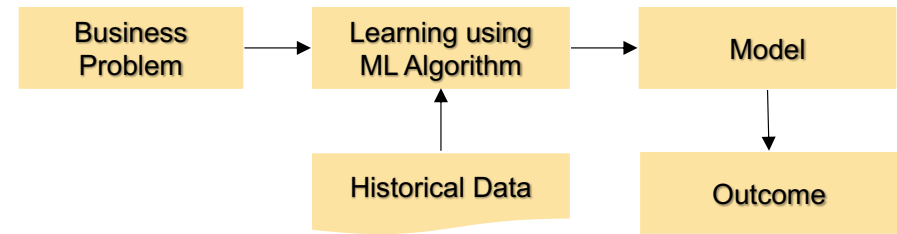
Machine Learning vs Traditional Programming

Traditional Programming



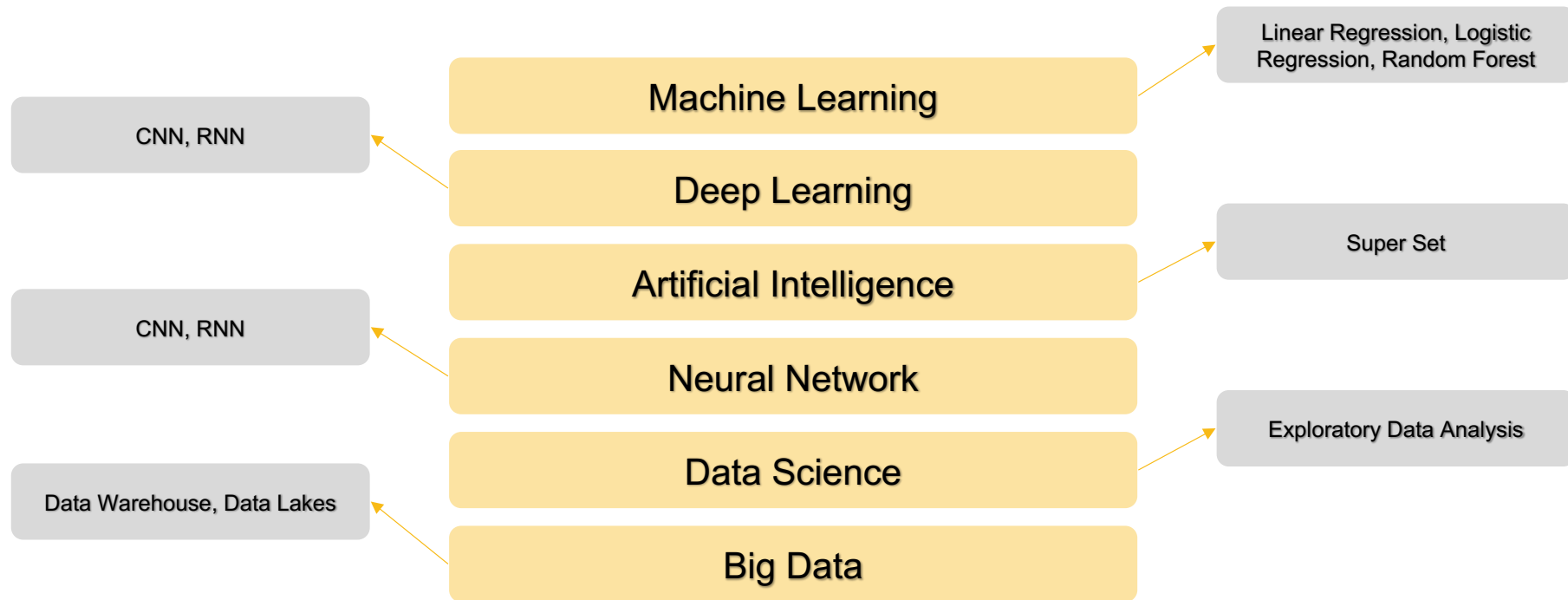
In traditional programming, business requirements are converted into “deterministic” rules. These rules are then applied on input data to produce outcomes. The rules are deterministic implying that they do not change or learn from the data they ingest or outcomes they produce.

Machine Learning



In Machine Learning, you will not write “deterministic” rules. Instead, you will use Machine Learning Algorithms to learn from the historical data fed and create a Model, which is essentially the set of rules that the algorithm has learnt from the data to solve a complex problem for which writing static rules is unsuitable.

Related Terminology



History of Machine Learning

- In 1943, Neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper about neurons, and how they work. They decided to create a model of this using an electrical circuit, and therefore the neural network was born.
 - In 1950, Alan Turing created the world-famous Turing Test – the challenge was for a computer to pass, it had to be able to convince a human that it is a human and not a computer.
 - 1952 saw the first computer program which could learn as it ran. It was a game which played checkers, created by Arthur Samuel.
 - Frank Rosenblatt designed the first artificial neural network in 1958, called Perceptron. The main goal of this was pattern and shape recognition.
 - In 1959, Bernard Widrow and Marcian Hoff created two models at Stanford University. The first was called ADELIN, and it could detect binary patterns. The next generation was called MADELINE, and it could eliminate echo on phone lines, so had a useful real-world application. It is still in use today.
 - 1982 was the year in which interest in neural networks started to pick up again, when John Hopfield suggested creating a network which had bidirectional lines, similar to how neurons actually work. Furthermore, in 1982, Japan announced it was focusing on more advanced neural networks, which incentivised American funding into the area, and thus created more research in the area.
-

History of Machine Learning

- In 1997, the IBM computer Deep Blue, which was a chess-playing computer, beat the world chess champion. Since then, there have been many more advances in the field, such as in 1998, when research at AT&T Bell Laboratories on digit recognition resulted in good accuracy in detecting handwritten postcodes from the US Postal Service.
 - Since the start of the 21st century, many businesses have realised that machine learning will increase calculation potential. This is why they are researching more heavily in it, in order to stay ahead of the competition.
 - GoogleBrain (2012) - This was a deep neural network created by Google, which focused on pattern detection in images and videos.
 - AlexNet (2012) - AlexNet won the ImageNet competition by a large margin in 2012, which led to the use of GPUs and Convolutional Neural Networks in machine learning. They also used ReLU, which is an activation function that greatly improves efficiency of CNNs.
 - DeepFace (2014) - This is a Deep Neural Network created by Facebook, which they claimed can recognise people with the same precision as a human can.
 - DeepMind (2014) - This company was bought by Google, and can play basic video games to the same levels as humans. In 2016, it managed to beat a professional at the game Go, which is considered to be one of the world's most difficult board games.
 - OpenAI (2015) - This is a non-profit organisation created by Elon Musk and others, to create safe artificial intelligence that can benefit humanity.
 - Amazon Machine Learning Platform (2015) - This is part of Amazon Web Services, and used in services such as search recommendations and Alexa, to more experimental ones like Prime Air and Amazon Go.
 - ResNet (2015) - This was a major advancement in CNNs, and more information can be found on the Introduction to CNNs page.
 - U-net (2015) - This is an CNN architecture specialised in biomedical image segmentation.
-

And the GPU Emerged

- GPUs are extremely important in the world of machine learning. GPUs have around 200 times more processors per chip than CPUs. The flip side of this, however, is that whereas CPUs can perform any kind of computation, GPUs are tailored to only specific use cases, where operations (addition, multiplication, etc.) have to be performed on vectors, which are essentially lists of numbers. A CPU would perform each operation on each number in the vector synchronously, i.e., one by one. This is slow. A GPU would perform operations on each number in the vector in parallel i.e., at the same time.
 - - Vectors and matrices, which are grids of numbers (or lists of vectors) are essential to machine learning applications, and because of this, they are smaller, hence why more can be fit on one chip.
 - Nvidia are credited with making the world's first GPU, the GeForce 256 in 1999. At that time, launching the product was a risk as it was an entirely new kind of product. However, due to the use of vector calculations in video games, GPUs proliferated, as video games benefitted from a huge leap in performance.
 - It was years later, than mathematicians, scientists and engineers realised that GPUs could be used to improve the speed of computations used in their discipline, due to the use of vectors.
 - This led to the realisation that GPUs would make neural networks, a very old idea, leaps and bounds more practical. This led to GPU companies particularly Nvidia benefitting hugely from the "machine learning revolution". Nvidia's stock price has increased roughly 18-fold since 2012, the year in which the importance of GPUs in machine learning was demonstrated by AlexNet.
 - - GPUs are developed specifically for machine learning. E.g., the Tesla V100, announced in May 2017, uses Tensor Cores, which are used for matrix arithmetic in machine learning.
 - A tensor core can compute 64 fixed point operations per clock cycle, as it provides a 4x4x4 matrix processing array.
-

And the GPU Emerged

Google Tensor Processing Unit (TPU) - 2016

- TPUs power a lot of Google's main services, including Search, Street View, Photos and Translate. They allow the neural networks behind these services to run faster, and so are run at a more affordable cost.
- Similar to the Tensor cores, it optimises how the multiplications, additions and activation functions are applied to data in CNNs, making the process much faster. Unlike tensor cores, which are part of general-purpose GPUs, TPUs are chips designed solely for accelerating computations required for neural networks.
- This makes them even more faster than general-purpose GPUs when performing machine learning tasks, as GPUs have to handle other use cases, and so are less specialised.

Intel - Nervana Neural Processor - 2017

Intel was late in the game of GPUs and TPUs, but have come up with the Nervana Neural Processor. This is very similar to Google's TPU. Matrix multiplications and convolutions are the two core operations performed by the processor.

GPUs in Cloud Computing

- A recent trend is the renting of GPUs from big companies such as Google and Amazon, rather than buying them, as this can be a much cheaper alternative to purchasing lots of GPUs. It also allows many more GPUs to be used, as a high quantity can be rented, and therefore work can be done more efficiently. One such popular example for learners is Google Colab.
-

Why Machine Learning

Can easily replace traditional programming methods where complex and very large set of combinations of rules.

Complex Problems where no traditional solutions exist, e.g. Image Processing, Voice Processing, Autonomous Engines such as Self Driving Cars.

In situations with changing scenarios.

Getting insights about complex problems and large amounts of data.

Email Spam Filtering

Image Classification

Stock Market Predictions

Fraud Detection

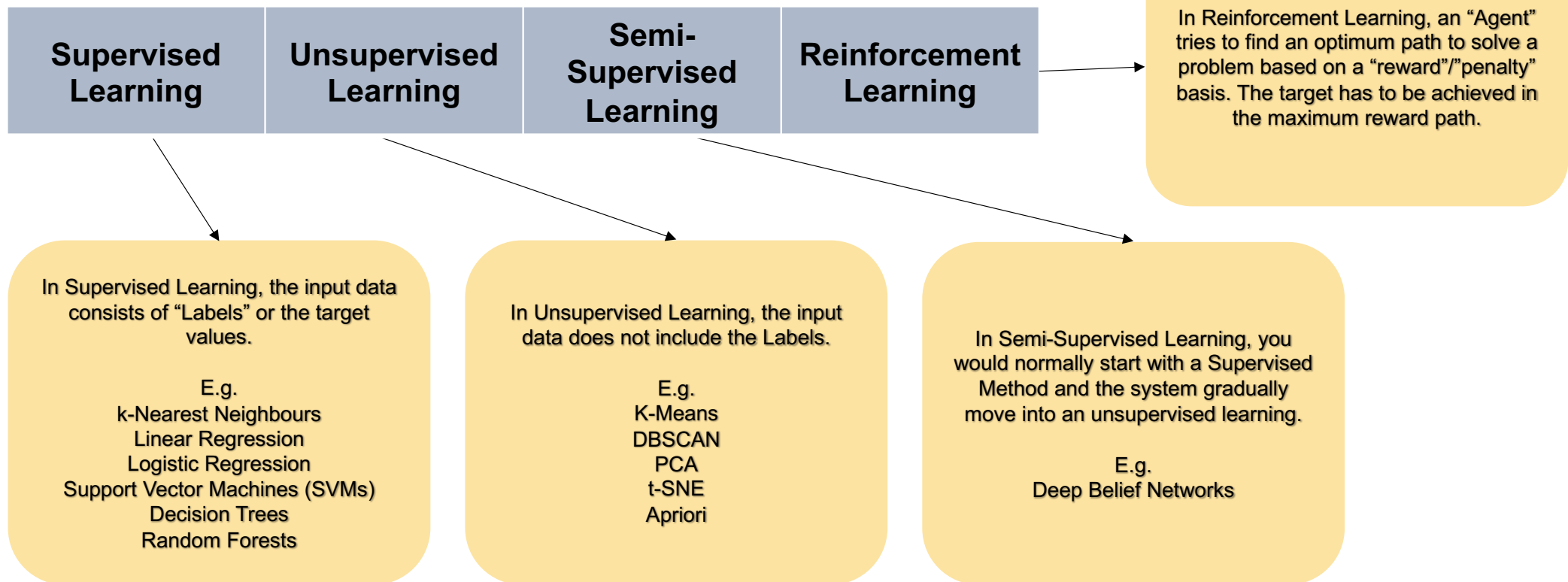
Types of Machine Learning

Machine Learning Algorithms can be classified in the following ways:

<u>Basis of Classifications of ML Algorithms</u>	<u>Classification Names</u>
Whether or not they are trained with Labelled Target values	Supervised Learning Unsupervised Learning Semi-Supervised Learning Reinforcement Learning
Whether or not they can learn incrementally on the fly	Online Learning Batch Learning
Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model	Instance-based Learning Model-based Learning

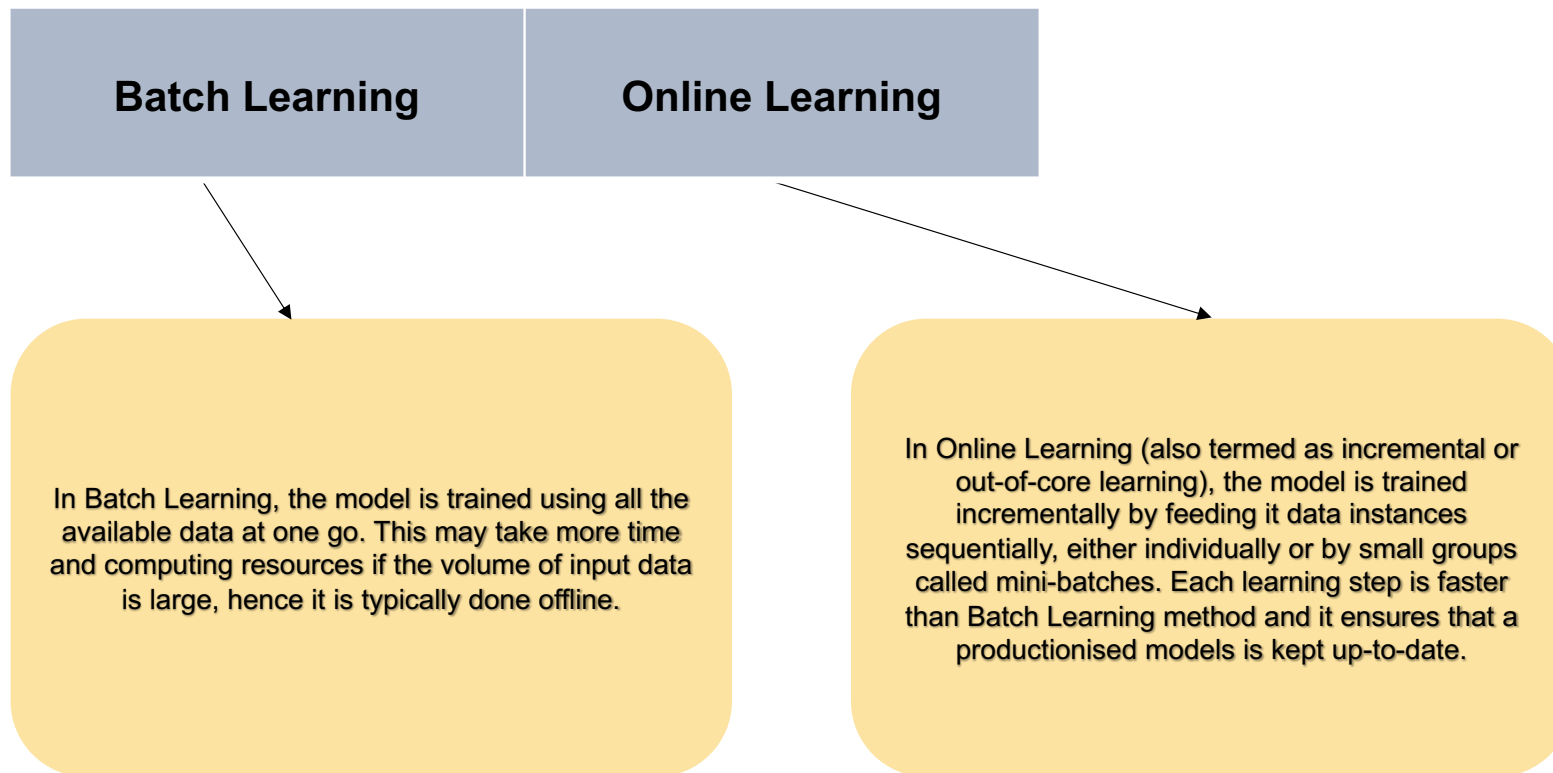
Types of Machine Learning

Classification based on whether they are labelled/unlabeled/reward based



Types of Machine Learning

Classification based on whether or not they can learn incrementally



Types of Machine Learning

Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model, much like scientists do

Instance based Learning

In instance-based learning (also called memory-based-learning), the system learns from the examples fed to it, then generalizes to new cases by comparing them to the learned examples (which are stored in the memory), using a similarity measure.

Model based Learning

In model-based learning, the system learns from input examples to build a model of these examples, then we use that model to make predictions.

Use cases of Machine Learning

Medical Imaging

Stock Price Prediction

Customer Segmentation

Product Recommendation

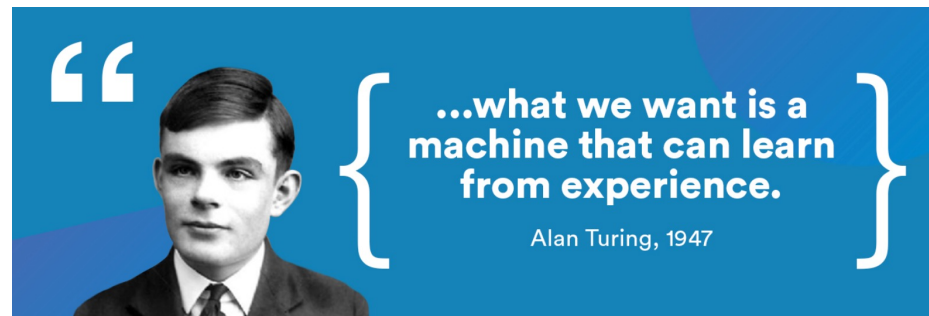
Sentiment Analysis

Fraud Detection

Customer Churn Prediction

Robotic Assistants

Autonomous Engines



Role of Data in Machine Learning

Data is in the heart of every Machine Learning Model. Learning Algorithms are applied on Data to learn patterns and create models to solve a problem, such as a Regression Problem or a Classification Problem.

The quality of the input data is hence key to the success or accuracy of the Machine Learning Model thus created.

Machine Learning professionals must take utmost care in choosing the right data and appropriate analysis and subsequent pre-processing to make the data suitable for ingestion to a specific Learning Algorithm to create the right model.

Models created by using rightly pre-processed data are subjected to validation/test to measure the accuracy/performance of the Model.

Types of Data

Numerical Data

Categorical Data

Time-Series Data

Text Data

Image Data

Challenges in Machine Learning

The two key components of Machine Learning

Algorithm

Data

Hence the challenges in setting up a successful ML system is:

- **Bad Algorithm**
 - **Bad Data**
-

Challenges in Machine Learning

Data Related Challenges

60-70% of the work involved in creating successful ML models is to do with data. However, availability of data suitable for creating such models is often an issue. The reason are varied and can be specific to organizations. Quite often it would point to organization's discipline in gathering and maintaining quality data within their systems and processes.

Data Availability

There are varied challenges related to Quality of data, such as missing values, outliers, lack of data, and so on.

There could be presence of outlier data that do not contribute to the generalization goal but have the potential to affect the model.

On the other hand, if we are to create a model for face recognition, we need a lot of images of the persons from various angles and sizes. Lack of these images will lead to inaccurate models.

Data Quality

The training data must be representative of the new (unseen) cases you want the model to generalize to. In case the data in terms of quantity and values are not representative of the goal of generalization, then we will be left with inaccurate models. In case of input sample set is too small, you will have sampling noise (i.e., nonrepresentative data as a result of chance). On the other hand, very large input (training) sample set can be non-representative if the sampling method is flawed. This is called sampling bias.

Non-representative Data

Challenges in Machine Learning

Data Related Challenges

One example of Imbalanced data is that in creating fraud detection models, often we have to deal with the fact that the quantity of fraud data is miniscule in ratio to the quantity of non-fraud data (thankfully fraud doesn't happen often). This situation leads to inaccurate models.

Imbalanced Data

Input Data set often comes with features that are irrelevant to the model goal. If we feed them to the model, the model will factor in their influence and the resultant model may not be a desired one given the business goal.

Unnecessary Features

Determining the right set of features is a part of data pre-processing, called Feature Engineering. Feature Engineering involves Feature Selection, Feature Extraction (deriving new features from existing ones, dimensionality reduction, etc.).

Noise in data refer to inaccurate/erroneous or missing values. The occurrences of noisy data in data set can significantly impact prediction of any meaningful information. Many empirical studies have shown that noise in data set dramatically led to decreased classification accuracy and poor prediction results.

Noisy Data

Challenges in Machine Learning

Algorithm Related Challenges

Machine Learning is all about finding patterns in data and be able to apply the patterns (model) on unseen (new) data to produce an outcome. Data can be various and noisy sometimes. And there are weak learning algorithms and strong and smart algorithms. Smarter algorithms have the ability to learn and memorize all the complex patterns of the entire input (training) data sets (e.g. in higher degree polynomial models). As a result, any variation found in the unseen data can lead to bad results (as this pattern may not have been found in training and the model overfitted on the training data). This is lack of generalizability due to overfitting. There are various constraining mechanism such as regularization that are applied on algorithms to prevent overfitting.

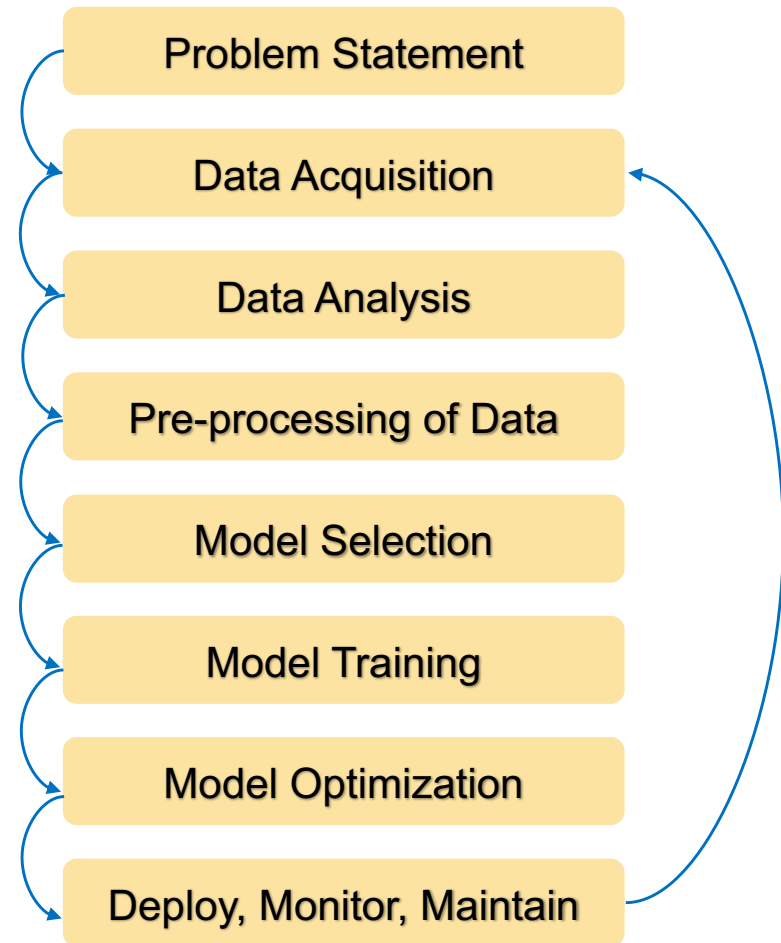
Overfitting

The reverse of overfitting is underfitting. When the algorithm is weak or heavily constrained to learn complex patterns, the resultant model will not have the ability to predict accurately. E.g. applying a linear model when the data is of higher degree will lead to underfitting.

Underfitting

Understanding the Machine Learning Project Lifecycle

Every Machine Learning initiative or project follow certain steps. There would be variations depending on the business goal and the type of Machine Learning algorithm used. However, there are some broad steps that will almost always be applicable. In the following few sections, we will look at these steps forming a complete Machine Learning project.



Framing the Problem or Business Goal

The start is always by asking the right set of questions:

- What is the problem in hand?
- Why do we want to do this?
- What objectives to achieve?
- Financial or other benefits to be achieved?
- Criteria/measurements for success of the solution?
- Is the right data available to solve the problem?
- Are there challenges in acquiring the data/quality of data?
- Can this be solved by traditional programming?
- Do we have target values/labels to train against or does it have to be unsupervised?



Data Acquisition Approach

Analysis Approach

Pre-processing Approach

Model Selection

Measurement Metrics Selection

Optimisation Approach

Sample Problem Statements

Insurance Company: “Our customer churn has increased almost 10% from last year. We would like to predict which customers are likely to leave us so that we can take actions to retain them. We should be able to prevent 70% of our likely churn customers to stay with us”.

eCommerce Company: Our cross-selling process is manual in the merchandizing system. We need a mechanism to push such products the customers are likely to buy automatically. We are aiming at a 25% increase in cross-sales volume.”

Hospital: We would like to achieve a minimum 80% coverage of all Heart Condition image evaluation process from the current manual scanning that our doctors undertake. We should achieve an optimally high Precision, Recall and Specificity.

Pipeline

A machine learning pipeline is a way to codify and automate the workflow it takes to produce a machine learning model. Machine learning pipelines consist of multiple sequential steps that do everything from data extraction and pre-processing to model training and deployment.

As a best practice for Machine Learning teams, all the logical components of Data processing and model building are compartmentalized and encapsulated into pipelines.

Pipelines enable the organization to maintain and scale its ML projects without disturbing the high-level architecture. It applies on whether you are maintaining multiple models in production or supporting a single model that needs to be updated frequently, an end-to-end machine learning pipeline is critical.

Examples of different components that may form a ML Pipeline:

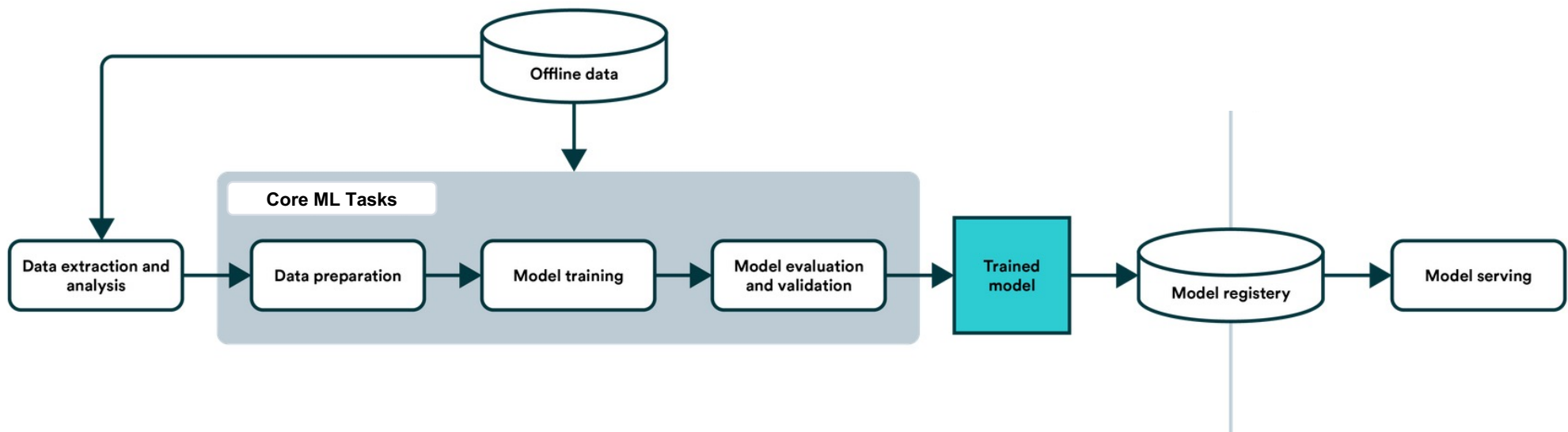
- Data Validation
- Data Clean-up
- Model Training
- Model Evaluation
- Model Validation
- Re-training Trigger

Repeatability

Maintainability

Reusability

A Typical Pipeline



The idea of a **Pipeline** is to codify all the individual logical components in the above workflow to deploy in production as a Pipeline for better maintainability and scalability.

Performance Metrics

A key element in the ML scheme of things is the Performance Metrics. There are a large number of Performance Metrics that are available. These depend upon the kind of Learning Algorithm in question and also the kind of business problem we are solving.

In the next few sections, we talk about the various performance metrics that are commonly used in Machine Learning.

Performance Measures for Regression Problems

Performance Measures for Classification Problems

Regression Problems

Regression Predictive Modelling

Regression predictive modelling is the task of approximating a mapping function (f) from input variables (X) to a continuous output variable (y).

A continuous output variable is a real-value, such as an integer or floating point value. These are often quantities, such as amounts and sizes.

For example, a house may be predicted to sell for a specific dollar value, perhaps in the range of \$100,000 to \$200,000.

- A regression problem requires the prediction of a quantity.
- A regression can have real valued or discrete input variables.
- A problem with multiple input variables is often called a multivariate regression problem.
- A regression problem where input variables are ordered by time is called a time series forecasting problem.

Because a regression predictive model predicts a quantity, the skill of the model must be reported as an error in those predictions.

A typical Linear regression model may look like the following:

$$y = a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3 + b$$

$$y = 3.67x_1 + 7.9x_2 + 8.1x_3$$

Where 'y' is the House Price to be predicted (Target Feature)

And x1, x2, x3 are various Input Features (e.g., Square foot, Number of BRs, Number of Balconies – these are features on which 'House Price' will typically vary.)

The problem of the Linear Regression model is to find out the optimum co-efficient or weight values (here, 3.67, 7.9 and 8.1).

Regression Metrics Worked out..

x1	x2	x3	y-act	y-pred	Difference
SqrFt Area	No of BRs	No of Balconies	House Price	House Price	House Price
1650	3	2	2,23,000	2,25,600	-2,600
1320	2	1	1,76,000	1,82,000	-6,000
1830	3	2	2,53,000	2,57,700	-4,700
1620	3	2	2,29,000	2,31,000	-2,000
2680	5	3	4,52,000	4,51,200	800

$$y = a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3 + b$$

$$y = 27x_1 + 121x_2 + 235x_3 + 766$$

<<< Linear Regression Mapping Function

<<< Learning Algorithm finds the optimum weight values

And x1, x2, x3 are various Input Features (e.g., Square foot, Number of BRs, Number of Balconies – these are features on which 'House Price' will typically vary.)

Performance Measures for Regression Problems

Common Regression Metrics

- Mean Absolute Error (MAE)
 - Residual Sum of Squares (RSS)
 - Mean Squared Error (MSE)
 - Total Sum of Squares (TSS)
 - Root Mean Squared Error (RMSE)
 - Co-efficient of Determination or R^2
-

Mean Squared Error (MSE) & RMSE

MSE or Mean Squared Error is an important Loss Function for regression algorithms. “*Least Squares*” refers to minimizing the mean squared error between predicted values and expected values.

MSE is calculated as the mean or average of the squared differences between predicted and expected target values in a dataset.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Where y_i is the i 'th expected value in the dataset and \hat{y}_i is the i 'th predicted value. The difference between these two values is squared, which has the effect of removing the sign, resulting in a positive error value.

The squaring also has the effect of inflating or magnifying large errors. That is, the larger the difference between the predicted and expected values, the larger the resulting squared positive error. This has the effect of “*punishing*” models more for larger errors when MSE is used as a loss function.

The Scikit Learn Library has the `mean_squared_error` function that automatically calculates MSE from arrays containing the expected and predicted values.

```
from sklearn.metrics import mean_squared_error
:
:
errors = mean_squared_error(expected, predicted)
```

Root Mean Squared Error (RMSE)

RMSE is the Square Root of the MSE value calculated from the sets of actual and predicted values.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

By square rooting the MSE value, RMSE essentially negates the ‘amplification’ of the error magnitude.

RMSE is used widely in regression problems.

Mean Absolute Error (MAE)

The MAE score is calculated as the average of the absolute error values. In MAE or Mean Absolute Error, the scale of the error score match the scale of the target value that is being predicted.

Unlike the RMSE, the changes in MAE are linear and therefore intuitive.

MSE punishes larger errors more, inflating the mean error score. The MAE does not give more or less weight to different types of errors and instead the scores increase linearly with increases in error.

Absolute or *abs()* is a mathematical function that simply makes a number positive. Therefore, the difference between an expected and predicted value may be positive or negative and is forced to be positive when calculating the MAE.

The MAE can be calculated as follows:

$$\text{MAE} = \frac{\sum_{i=1}^n |y_i - x_i|}{n}$$

Where y_i is the i 'th expected value in the dataset, x_i is the i 'th predicted value and *abs()* is the absolute function.

```
err = abs((expected[i] - predicted[i]))
```

Other Metrics

Residual Sum of Squares (RSS)	RSS or Residual Sum of Squares is the sum of all the squares of residuals (error or difference between the predicted and actual values).
Total Sum of Squares (TSS)	TSS or Total Sum of Squares is defined as the sum of all squared differences between the observations and their overall mean.
Co-efficient of Determination – R^2	Coefficient of Determination , denoted R^2 or r^2 and pronounced "R squared", is the proportion of the variation in the dependent variable that is predictable from the independent variable(s).

$$RSS = \sum_{i=1}^n (y_i - f(x_i))^2$$

$$TSS = \sum_{i=1}^n (y_i - \bar{y})^2$$

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Notes:

- Typically, the aim of a Regression Algorithm is to find out the optimum values of a_1 , a_2 , a_3 , in a regression equation like $y = a_1.x_1 + a_2.x_2 + a_3.x_3$.
 - The optimum values of a_1 , a_2 , a_3 are found out by minimizing the chosen Regression Metrics value (one of the ones that we studied above).
-

Regression Metrics Worked out..

LR Metrics

House Price - Actual	House Price - Predicted	Differences	Absolute Differences	<< Squares of the Differences	Differences with Mean of Actual	<< Squares of the Differences
\$ 2,23,000	\$ 2,36,000	\$ (13,000)	\$ 13,000	\$ 16,90,00,000	\$ 30,600	\$ 93,63,60,000
\$ 1,76,000	\$ 1,69,000	\$ 7,000	\$ 7,000	\$ 4,90,00,000	\$ 97,600	\$ 9,52,57,60,000
\$ 2,53,000	\$ 2,46,000	\$ 7,000	\$ 7,000	\$ 4,90,00,000	\$ 20,600	\$ 42,43,60,000
\$ 2,29,000	\$ 2,36,000	\$ (7,000)	\$ 7,000	\$ 4,90,00,000	\$ 30,600	\$ 93,63,60,000
\$ 4,52,000	\$ 4,58,000	\$ (6,000)	\$ 6,000	\$ 3,60,00,000	\$ (1,91,400)	\$ 36,63,39,60,000

\$ 2,66,600 << Mean of Actual Values

Mean Absolute Error (MAE)	\$ 8,000
Mean Squared Error (MSE)	\$ 7,04,00,000
Residual Sum of Squares (RSS)	\$ 35,20,00,000
Total Sum of Squares (TSS)	\$ 48,45,68,00,000
Root Mean Squared Error	\$ 18,762

- **'Predicted'** are values are found by using the model during the training/test process.
- The difference between the **'Actual'** and **'Predicted'** is the **'Error'**.
- In Regression problems having continuous numeric value as **'Target'**, **'Error'** is also measured in continuous numeric values.

Classification Problems

Classification predictive modelling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y).

The output variables are often called labels or categories. The mapping function predicts the class or category for a given observation.

For example, an email of text can be classified as belonging to one of two classes: “spam” and “ham”.

- A classification problem requires that examples be classified and labelled into one of two or more classes.
- A classification can have real-valued or discrete input variables.
- A problem with two classes is often called a two-class or binary classification problem.
- A problem with more than two classes is often called a multi-class classification problem.
- A problem where an example is assigned multiple classes is called a multi-label classification problem.

It is common for classification models to predict a continuous value as the probability of a given example belonging to each output class. The probabilities can be interpreted as the likelihood or confidence of a given example belonging to each class. A predicted probability can be converted into a class value by selecting the class label that has the highest probability.

E.g., a specific email of text may be assigned the probabilities of 0.1 as being “spam” and 0.9 as being “not spam”. We can convert these probabilities to a class label by selecting the “not spam” label as it has the highest predicted likelihood.

Classification Problems

x1	x2	x3	x4	x5	x6	x7	x8	y	y-pred
Age	Sex	Resting BP	Cholesterol	Fasting Blood Sugar	Resting ECG	Max Heart Rate	Thalium Stress	Disease?	Disease?
63	1	145	233	1	0	150	1	1	1
37	1	130	250	0	1	187	2	1	0
41	0	130	204	0	0	172	2	0	0
56	1	120	236	0	1	178	2	1	1
57	0	120	354	0	1	163	2	0	0

Above is an example of a 'Binary' Classification problem (data)

'Error' for such is classification problem is in terms of how many labels the model is able to predict correctly.

Classification Problem Metrics

The basis of Classification Metrics hinge on the **Confusion Matrix**.

In the following example, we will go through a Binary Classification **Confusion Matrix**.
We assume that there are two labels – Positive and Negative.

Actual class \ Predicted class	P	N
P	TP	FN
N	FP	TN

The confusion matrix captures the count of the following numbers:

- **True Positives (TP)** – Count of Positive Labels that the model predicted correctly
- **True Negatives (TN)** – Count of Negative Labels that the model predicted correctly
- **False Positive (FP)** – Count of Positive Labels that the model predicted as Negative (wrongly)
- **False Negative (FN)** – Count of Negative Labels that the model predicted as Positive (wrongly)

Accuracy:

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$$

Classification Problem Metrics

There are several other Metrics related to Classification problems.

Sensitivity, Recall or True Positive Rate

$$\text{TPR} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}} = 1 - \text{FNR}$$

Specificity, Selectivity, True Negative Rate

$$\text{TNR} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{TN} + \text{FP}} = 1 - \text{FPR}$$

Precision or Positive Predictive Value

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}} = 1 - \text{FDR}$$

F1-Score

$$\text{F}_1 = 2 \times \frac{\text{PPV} \times \text{TPR}}{\text{PPV} + \text{TPR}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

Optimizing Metrics for better Performance

Accuracy in classification problems is the number of correct predictions made by the model over all kinds predictions made.

Typically, higher accuracy would mean better performance by the Classification Model. However, that is not always true.

E.g., In a Heart Condition detection example with 100 people, only 5 people has Heart Condition . Let's say our model is very bad and predicts every case as No Heart Condition . In doing so, it has classified those 95 non-Heart Condition patients correctly and 5 Heart Condition patients as Non-Heart Condition. Now even though the model is terrible at predicting Heart Condition, The accuracy of such a bad model is also 95% which apparently looks great.

Precision is a measure that tells us what proportion of patients that we diagnosed as having Heart Condition, actually had Heart Condition. The predicted positives (People predicted as Heart Condition are TP and FP) and the people actually having a Heart Condition are TP.

*E.g., In an example with 100 people, only 5 people have Heart Condition. Let's say our model is very bad and predicts every case as **Heart Condition**. Since we are predicting everyone as having Heart Condition, our denominator (True positives and False Positives) is 100 and the numerator, person having Heart Condition and the model predicting his case as Heart Condition is 5. So in this example, we can say that **Precision** of such model is 5%.*

Recall is a measure that tells us what proportion of patients that actually had Heart Condition was diagnosed by the algorithm as having Heart Condition. The actual positives (People having Heart Condition are TP and FN) and the people diagnosed by the model having a Heart Condition are TP. (Note: FN is included because the Person actually had a Heart Condition even though the model predicted otherwise).

*Ex: In our Heart Condition example with 100 people, 5 people actually have Heart Condition. Let's say that the model predicts every case as Heart Condition. So our denominator (True positives and False Negatives) is 5 and the numerator, person having Heart Condition and the model predicting his case as Heart Condition is also 5 (Since we predicted 5 Heart Condition cases correctly). So in this example, we can say that the **Recall** of such model is 100%. And Precision of such a model (As we saw above) is 5%.*

IMP: So basically if we want to focus more on minimising False Negatives, we would want our Recall to be as close to 100% as possible without precision being too bad and if we want to focus on minimising False positives, then our focus should be to make Precision as close to 100% as possible.

Optimizing Metrics for better Performance

Specificity is a measure that tells us what proportion of patients that did NOT have Heart Condition, were predicted by the model as non-Heart Condition. The actual negatives (People actually NOT having Heart Condition are FP and TN) and the people diagnosed by us not having Heart Condition are TN. (Note: FP is included because the Person did NOT actually have Heart Condition even though the model predicted otherwise).

Specificity is the exact opposite of recall.

*E.g., In our Heart Condition example with 100 people, 5 people actually have Heart Condition. Let's say that the model predicts every case as Heart Condition. So our denominator (False positives and True Negatives) is 95 and the numerator, person not having Heart Condition and the model predicting his case as no Heart Condition is 0 (Since we predicted every case as Heart Condition). So in this example, we can say that the **Specificity** of such a model is 0%.*

F1-Score is a single consolidated score that represents both Precision(P) and Recall(R).

F1 Score = $2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall})$

Generally, if any of precision and recall is small, the F1 Score kind of raises a flag and is more closer to the smaller number than the bigger one, giving the model an appropriate score rather than just an arithmetic mean. This is because the formula is a Harmonic mean, a kind of an average when x and y are equal. But when x and y are different, then it's closer to the smaller number as compared to the larger number.

Bias and Variance

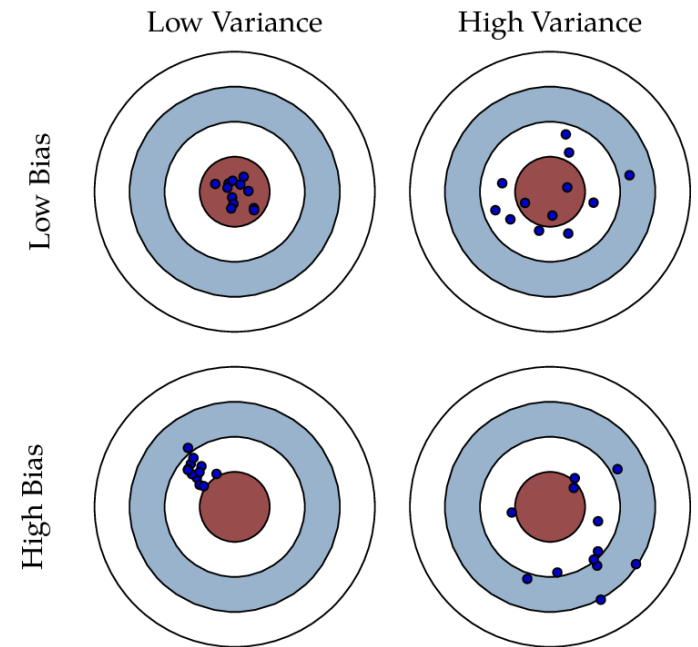
There are two types of Generalization Errors that can come out of Machine Learning Models:

Bias

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. **Model with high bias pays very little attention to the training data and oversimplifies the model.** It always leads to high error on training and test data.

Variance

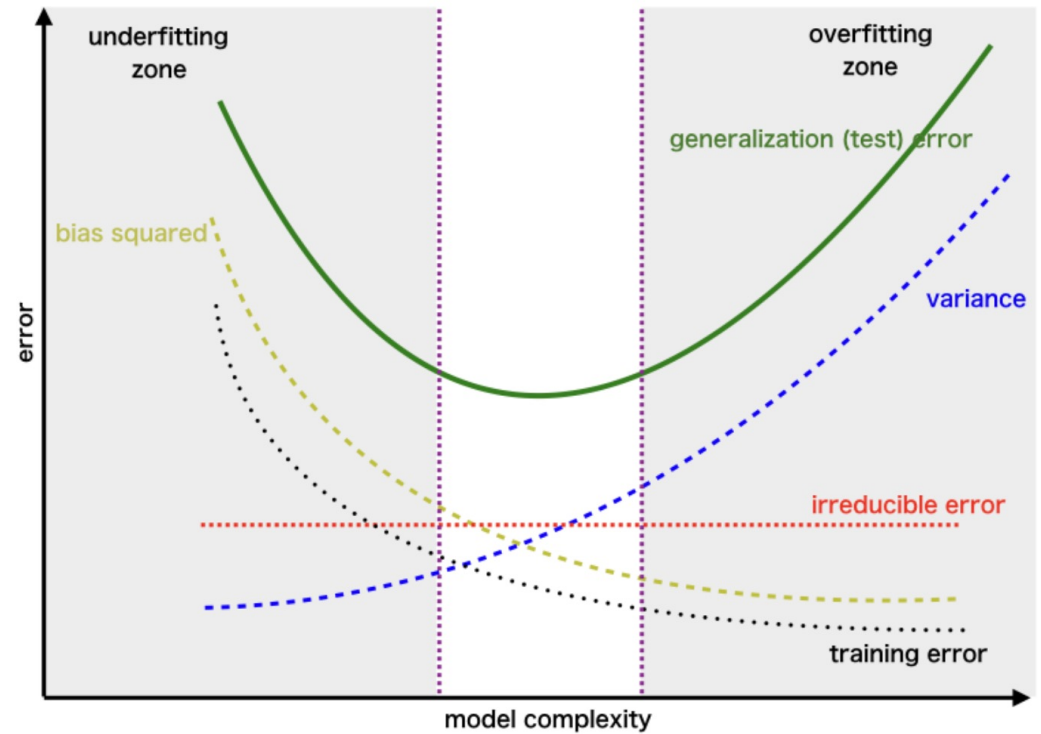
Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. **Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before.** As a result, such models perform very well on training data but has high error rates on test data.



Bias-Variance Trade-off

Irreducible error

This part is due to the noisiness of the data itself. The only way to reduce this part of the error is to clean up the data (e.g., fix the data sources, such as broken sensors, or detect and remove outliers).



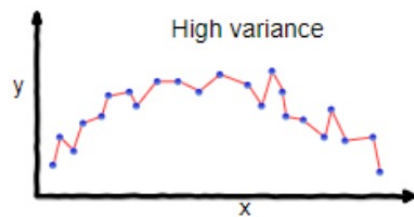
$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Bias-Variance Trade-off

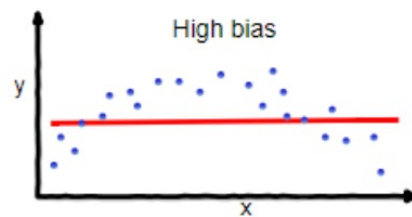
If our model is too simple and has very few parameters then it may have **high bias and low variance**.

On the other hand if our model has large number of parameters then it's going to have **high variance and low bias**. So we need to find the right/good balance without overfitting and underfitting the data.

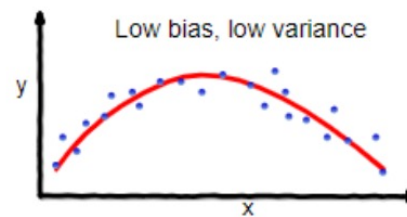
This trade-off in complexity is why there is a trade-off between bias and variance. An algorithm can't be more complex and less complex at the same time.



overfitting



underfitting



Good balance

Linear Algebra



Introduction to Linear Algebra

Linear algebra is widely used in the field of **Data Science, Machine Learning and Deep Learning** and its **essential for students of this subjects to attain at least some foundation level knowledge**. Good understanding of linear algebra really enhances the understanding of many **Machine Learning Algorithms** and to really understand **Deep Learning algorithms**, **Linear Algebra** is essential.

This article introduces the most important basic linear algebra concepts, and shows two relevant data science applications of linear algebra.

We will also draw the Code equivalents of handling Matrix and Vectors in Python, in the form of Numpy Arrays.

Topics Covered

Matrices and Vectors

Matrix Operations

Matrix Inverse

Orthogonal Matrix

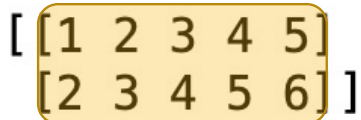
Applications in ML

Matrix and Vectors

A **Matrix** is an array of numbers, symbols or expressions, made up of rows and columns. A matrix is characterized by the amount of rows, m , and the amount of columns, n , it has. In general, a matrix of order ' $m \times n$ ' (read: "m by n") has m rows and n columns.

In the below example, we display an example 2 x 5 matrix.

```
# Create a Numpy Array from Tuple
arr = np.array([[1,2,3,4,5], [2,3,4,5,6]])
print(arr)
```

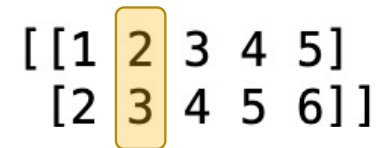


[[1 2 3 4 5]
[2 3 4 5 6]]

↑
A 2 x 5 Matrix

A matrix with only a single column is called a **Vector**.

For example, every column of our matrix is a vector. Let us take the second column of the matrix as the vector v :



[[1 2 3 4 5]
[2 3 4 5 6]]

↑
A Vector ' v '

Matrix Summation

If A and B are $m \times n$ matrices, then the **sum** $A+B$ is the $m \times n$ matrix whose columns are the sums of the corresponding columns in A and B. The sum $A+B$ is defined only when A and B are the same size.

```
A = np.array([[1,2,3,4,5], [2,3,4,5,6]])  
print(A)
```

```
[[1 2 3 4 5]  
 [2 3 4 5 6]]
```

```
B = np.array([[3,2,6,2,5], [1,3,5,7,8]])  
print(B)
```

```
[[3 2 6 2 5]  
 [1 3 5 7 8]]
```

```
C = A + B  
print(C)
```

```
[[ 4  4  9  6 10]  
 [ 3  6  9 12 14]]
```

Scaler Multiplication

If r is a scalar, then the scalar multiple of the matrix A is rA , which is the matrix whose columns are r times the corresponding columns in A .

```
A = np.array([[1,2,3,4,5], [2,3,4,5,6]])  
print(A)
```

```
[[1 2 3 4 5]  
 [2 3 4 5 6]]
```

```
X = 5*A  
print(X)
```

```
[[ 5 10 15 20 25]  
 [10 15 20 25 30]]
```

Matrix-Vector Multiplication

If the matrix A is of size $m \times n$ (thus, it has n columns), and u is a vector of size n , then the product of A and u , denoted by Au , is the **linear combination** of the columns of A using the corresponding entries in u as weights.

```
A = np.array([[1,2,3,4,5], [2,3,4,5,6]])  
print(A)
```

```
[[1 2 3 4 5]  
 [2 3 4 5 6]]
```

```
v = [2,4,3,6,4]  
Av = A*v  
print(Av)
```

```
[[ 2  8  9 24 20]  
 [ 4 12 12 30 24]]
```

Matrix Multiplication

If A is an $m \times n$ matrix and $B = [\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_p]$ is an $n \times p$ matrix where \mathbf{b}_i is the i -th column of the matrix B, then the matrix product AB is the $m \times p$ matrix whose columns are $A\mathbf{b}_1, A\mathbf{b}_2, \dots, A\mathbf{b}_p$.

The number of columns in A must match the number of rows in B in order to perform matrix multiplication.

A1	A2
A3	A4
A5	A6
A7	A8



B1	B2	B3
B4	B5	B6



A1*B1 + A2*B4	A1*B2 + A2*B5	A1*B3 + A2*B6
A3*B1 + A4*B4	A3*B2 + A4*B5	A3*B3 + A4*B6
A5*B1 + A6*B4	A5*B2 + A6*B5	A5*B3 + A6*B6
A7*B1 + A8*B4	A7*B2 + A8*B5	A7*B3 + A8*B6

```
# 2x3 and 3x2 dimensional arrays Multiplication
A = np.array([[1,4,7],[2,5,8]])
B = np.array([[1,4],[2,5],[3,6]])
C = np.dot(A,B)
C
```

```
[30, 66]
[36, 81]
```


Matrix Transpose

Suppose we have a matrix A of size $m \times n$, then the **transpose** of A (denoted by A^T) is the $n \times m$ matrix whose columns are formed from the corresponding rows of A .

```
A = np.array([[1,2,3], [4,5,6], [3,2,5]])
print(A)
A_T = np.transpose(A)
A_T
```

```
[[1 2 3]
 [4 5 6]
 [3 2 5]]
```

```
array([[1, 4, 3],
       [2, 5, 2],
       [3, 6, 5]])
```

Matrix Transpose Properties

1. $(A+B)^T = A^T + B^T$
 2. $(AB)^T = B^T A^T$
 3. $(kA)^T = kA^T$
 4. $(A^T)^T = A$
-

Matrix Inverse

An $n \times n$ matrix A is said to be **invertible** if there is an $n \times n$ matrix C such that:

Where I is the $n \times n$ **identity matrix**. An identity matrix is a square matrix with 1's on the diagonal and 0's elsewhere. Below, the 5×5 identity matrix is shown:

$$CA = I \text{ and } AC = I$$

$$I_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Going back to the invertibility principle above, we call the matrix C an **inverse** of A . In fact, C is uniquely determined by A , because if B were another inverse of A , then $B = BI = B(AC) = (BA)C = IC = C$. This unique inverse is denoted by A^{-1} , so that:

$$A^{-1}A = I \text{ and } AA^{-1} = I$$

```
A = np.array([[1,2,3], [4,5,6], [3,2,5]])
print(A)
A_inv = np.linalg.inv(A)
A_inv
```

```
[[1 2 3]
 [4 5 6]
 [3 2 5]]
```

```
array([[ -1.08333333,  0.33333333,  0.25      ],
       [ 0.16666667,  0.33333333, -0.5      ],
       [ 0.58333333, -0.33333333,  0.25      ]])
```

Orthogonal Matrix

An orthogonal matrix is a square matrix whose columns and rows are orthogonal unit vectors. That is, an **orthogonal matrix** is an invertible matrix, let us call it Q , for which:

$$Q^T Q = Q Q^T = I$$

This leads to the equivalent characterization: a matrix Q is orthogonal if its transpose is equal to its inverse:

$$Q^T = Q^{-1}$$

```
# Create a numpy Array from Tuple
arr = np.array([[1,2,3,4,5], [2,3,4,5,6]])
print(arr)
print(arr.shape)
```