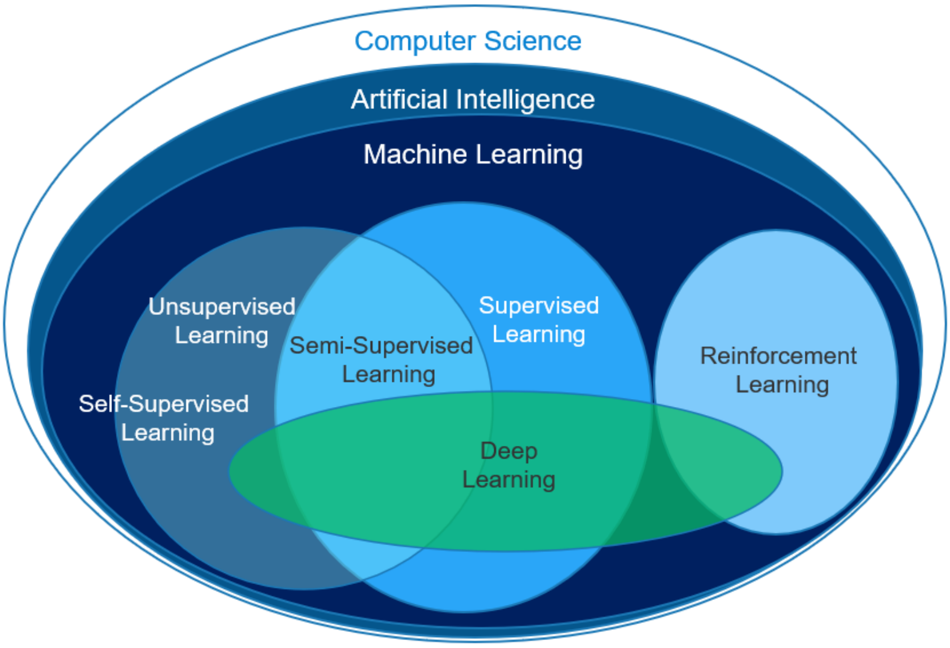
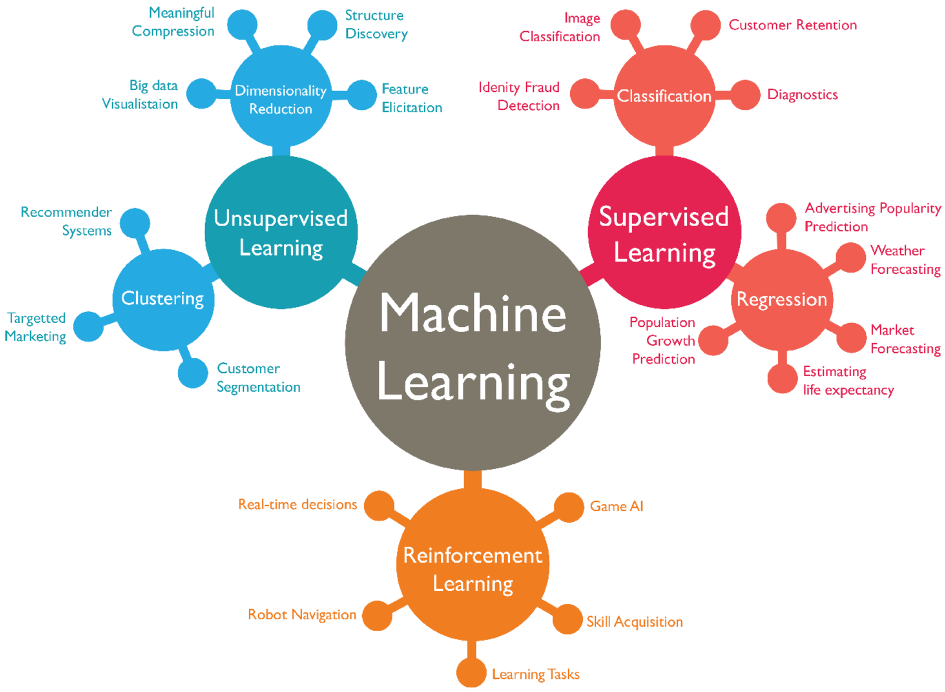
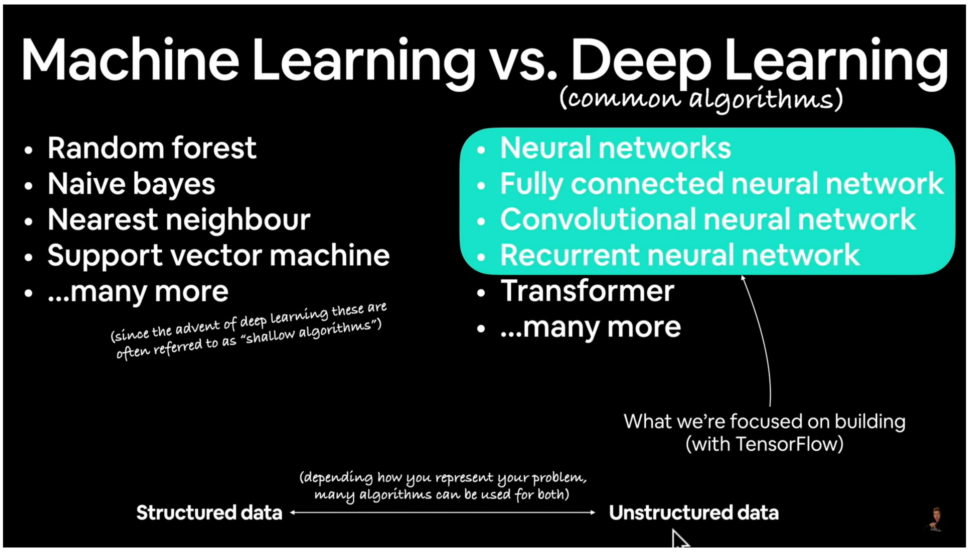
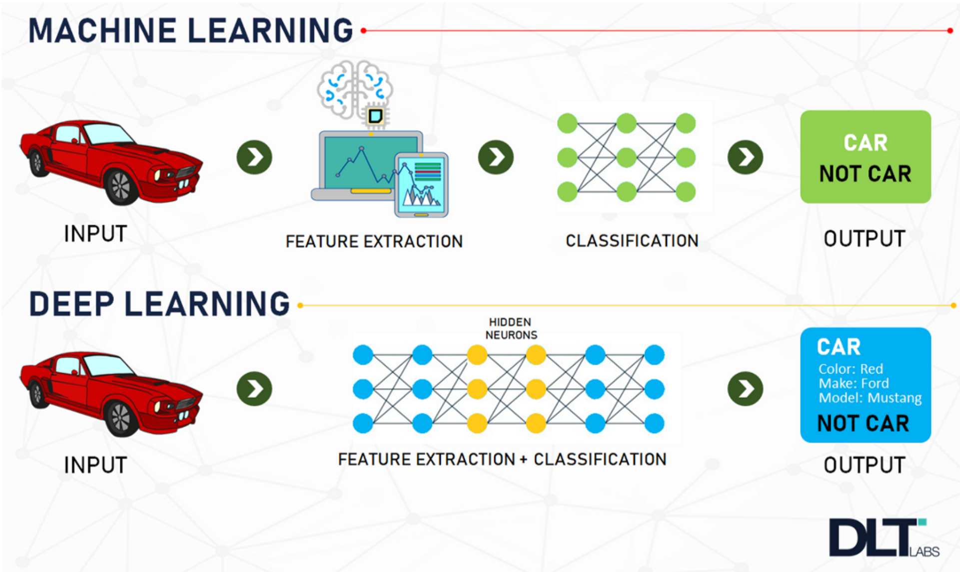
Machine Learning Overview



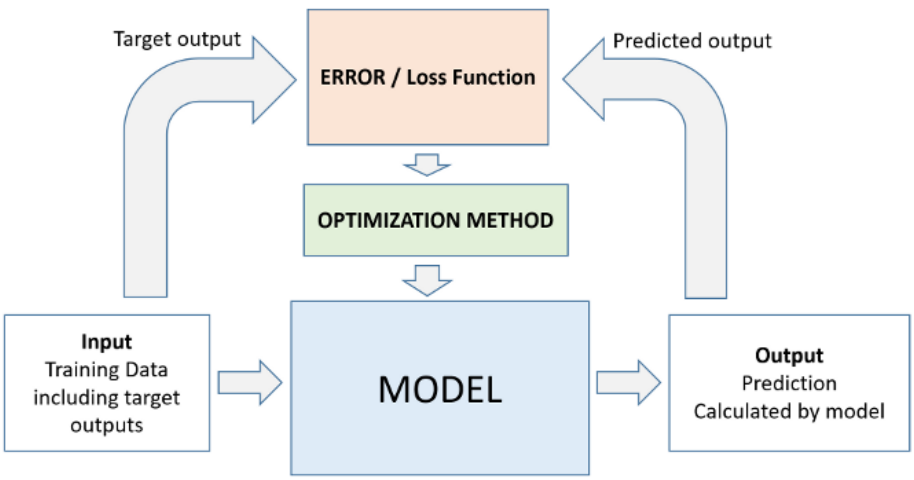


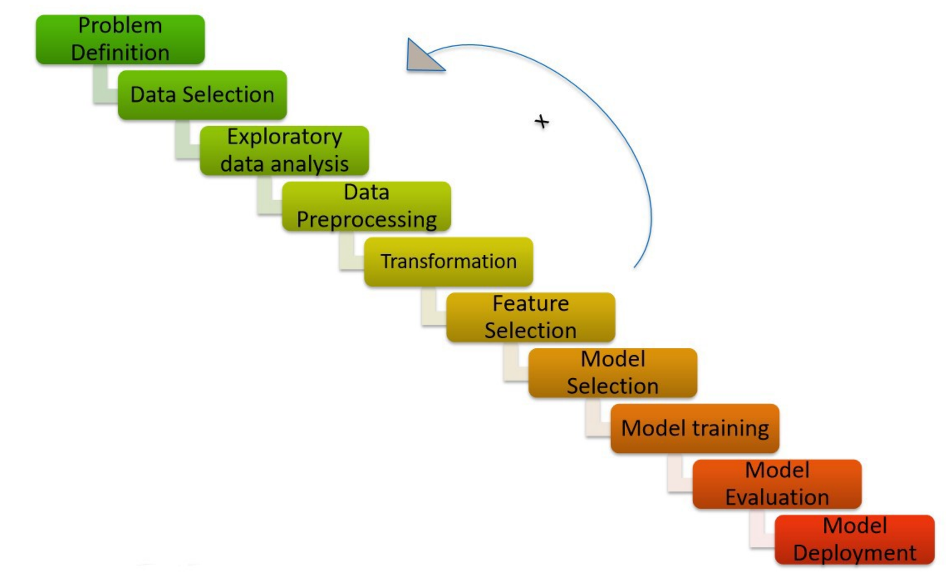
Maching Learning vs Deep Learning

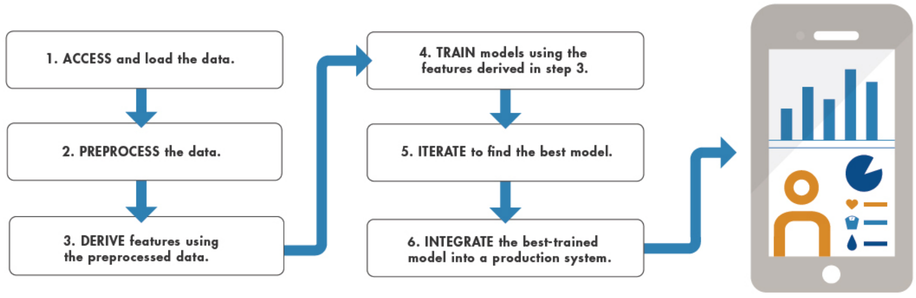




Machine Learning Workflow







Workflow 1. Access and load Data

> `import opendatasets as od`
> ->*od.download(dataset_url)*
>
> `from urllib.request import urlretrieve`
> ->*urlretrieve(url, "path/file") # rewrite the file (becareful)*
>
> `from zipfile import ZipFile`
> -> with ZipFile('filename.zip') as f:
> f.extractall(path='filename')
>
> `import pandas as pd`
> ->*pd.read_csv("path/csvfile")*

Workflow 2. Preprocess the Data -> Design Thinking

> ## 2.1 Exploratory Analysis
>
> `import numpy as np`
> -> *np.arange()*

-> *np.mean()*

`import pandas as pd`

-> *df.head(),*

-> df.sample()*

-> *df.isna().sum()*

-> *df.unique()*

-> *df.nunique()*

-> *df.value_counts()*

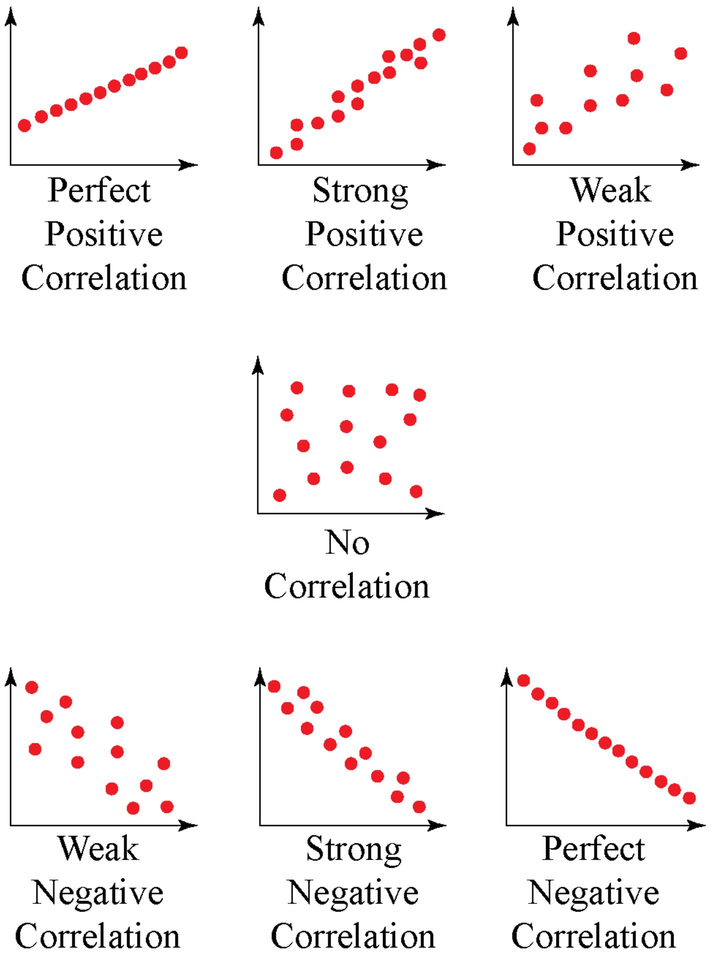
-> *df.sum()*

-> *df.shape*

-> *df.info()*

-> *df.describe()*

-> *df.corr()* (Source) >(Correlation Explaination)



2.2 Visualization

`import matplotlib.pyplot as plt`

`import seaborn as sn`

`import plotly.express as px`

2.3 Imputing Missing Numeric Data

> - `pandas -> pd.DataFrame -> df`
>   -> *df.to_datetime()*
>   -> *df.fillna()*
>   -> *df.dropna(subset=[])*
>
> - `from sklearn.impute import SimpleImputer`
>
> | | col1 | col2 | col3 | col4 | col5 |
> |---|---|---|---|---|---|
> | **0** | 2 | 5.0 | 3.0 | 6 | NaN |
> | **1** | 9 | NaN | 9.0 | 0 | 7.0 |
> | **2** | 19 | 17.0 | NaN | 9 | NaN |
>
> mean() →
>
> | | col1 | col2 | col3 | col4 | col5 |
> |---|---|---|---|---|---|
> | **0** | 2.0 | 5.0 | 3.0 | 6.0 | 7.0 |
> | **1** | 9.0 | 11.0 | 9.0 | 0.0 | 7.0 |
> | **2** | 19.0 | 17.0 | 6.0 | 9.0 | 7.0 |

Workflow 3. Drive Features Using the Processed Data

> - `Numerical features` : StandardScaler or MinMaxScaler
> - `Categorical features` OneHotEncoder
> - **`Using Power BI to Prepare the Data`**

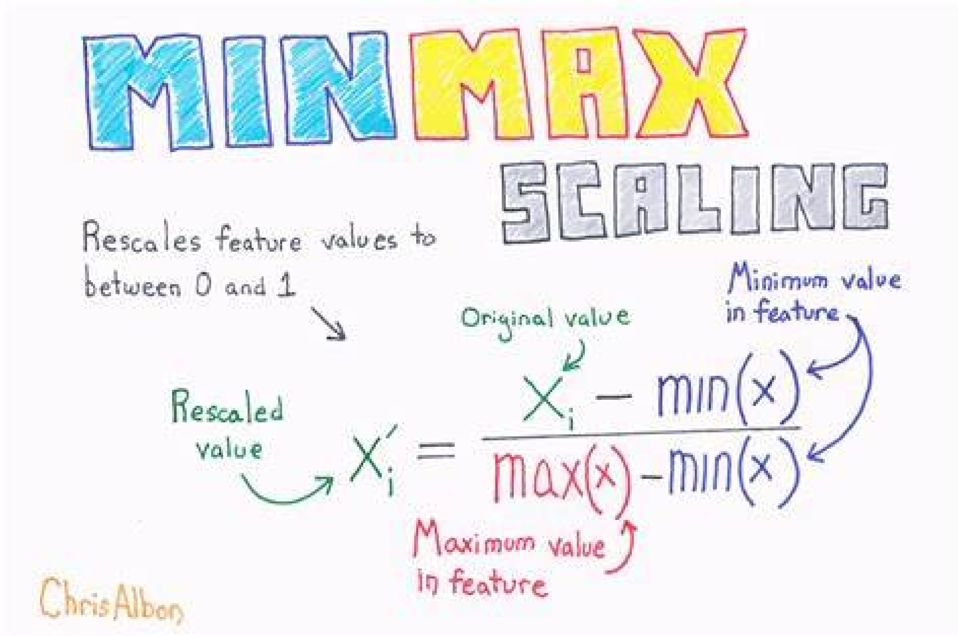
> ## Numerical features
>
> - StandardScaler
>
> **`from sklearn.preprocessing import StandardScaler`**
> -> *df=StandardScaler().fit_transform(df)*

> Standardization:
>
> $$z = \frac{x-\mu}{\sigma}$$
>
> with mean:
>
> $$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$
>
> and standard deviation:
>
> $$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}$$

> - MinMaxScaler

`from sklearn.preprocessing import MinMaxScaler`

-> *df=MinMaxScaler().fit_transform(df)*

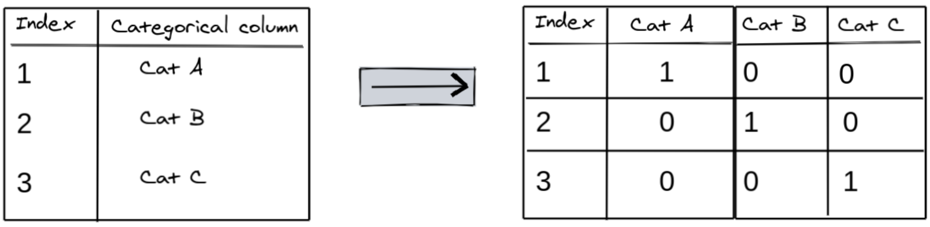


Categorical Features

`df.map({})`

`from sklearn.preprocessing import OneHotEncoder`

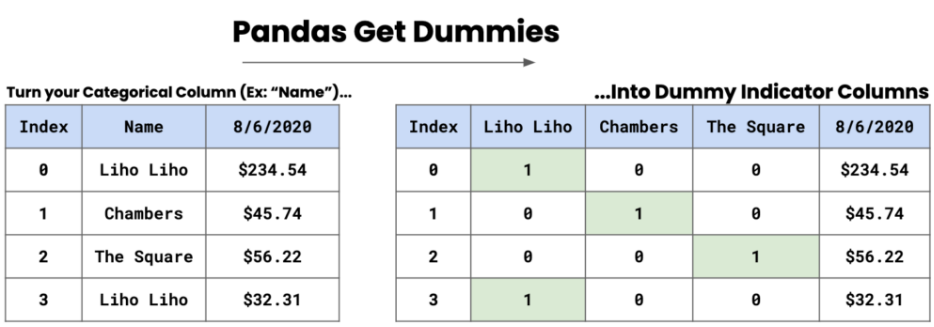
One hot encoding involves adding a new binary (0/1) column for each unique category of a categorical column.



Index	Categorical column
1	Cat A
2	Cat B
3	Cat C

Index	Cat A	Cat B	Cat C
1	1	0	0
2	0	1	0
3	0	0	1

`pd.get_dummies(df['columnName'])`



Pandas Get Dummies

Turn your Categorical Column (Ex: "Name")...

Index	Name	8/6/2020
0	Liho Liho	$234.54
1	Chambers	$45.74
2	The Square	$56.22
3	Liho Liho	$32.31

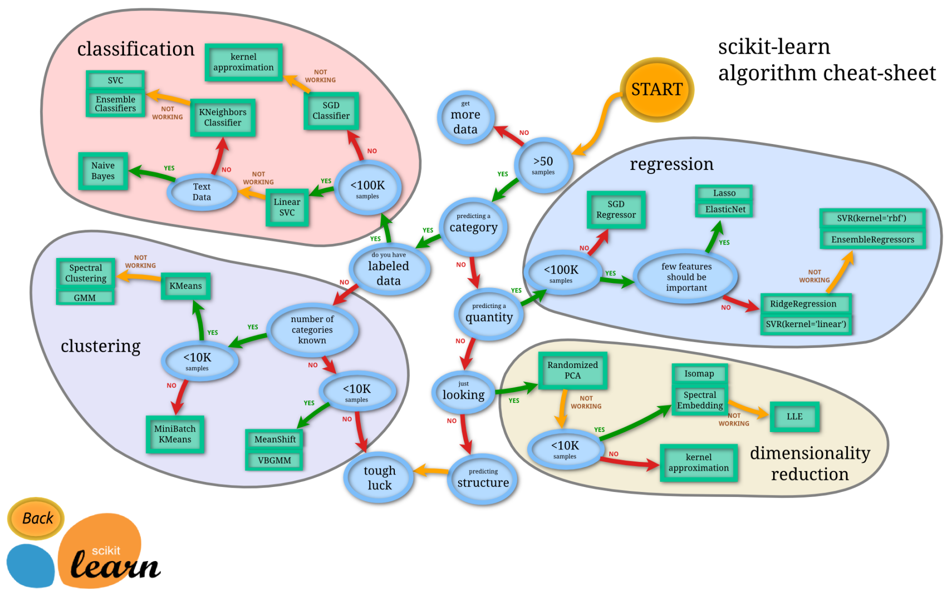
...Into Dummy Indicator Columns

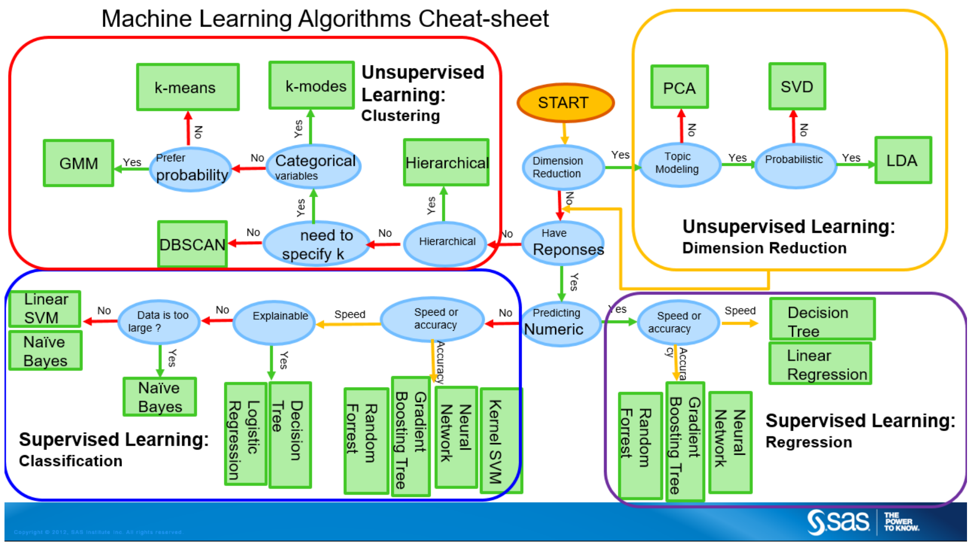
Index	Liho Liho	Chambers	The Square	8/6/2020
0	1	0	0	$234.54
1	0	1	0	$45.74
2	0	0	1	$56.22
3	1	0	0	$32.31

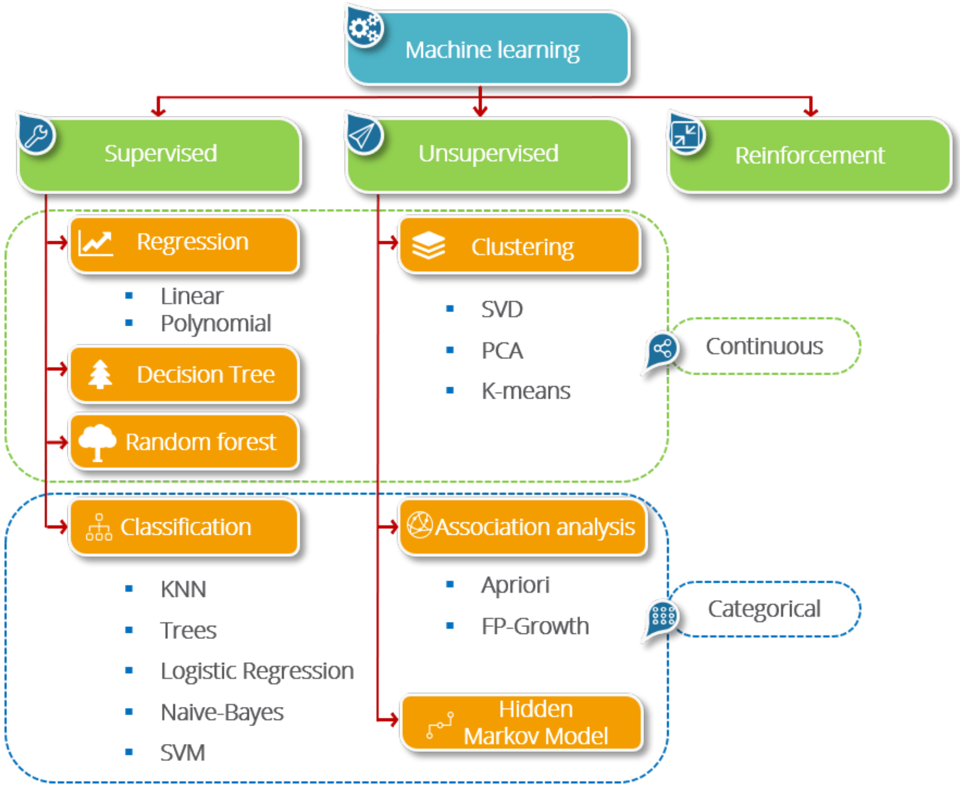
Workflow 4. Model Selection and Training

4.1 Model Selection

Scikit-learn offers the following cheatsheet to decide which model to pick for a given problem

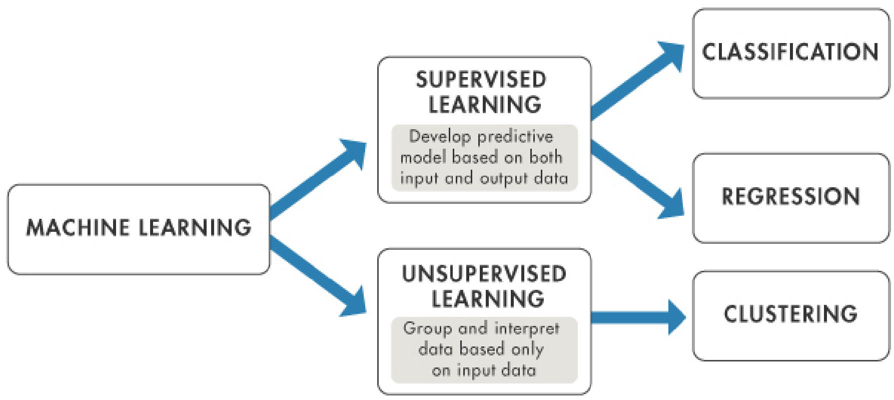


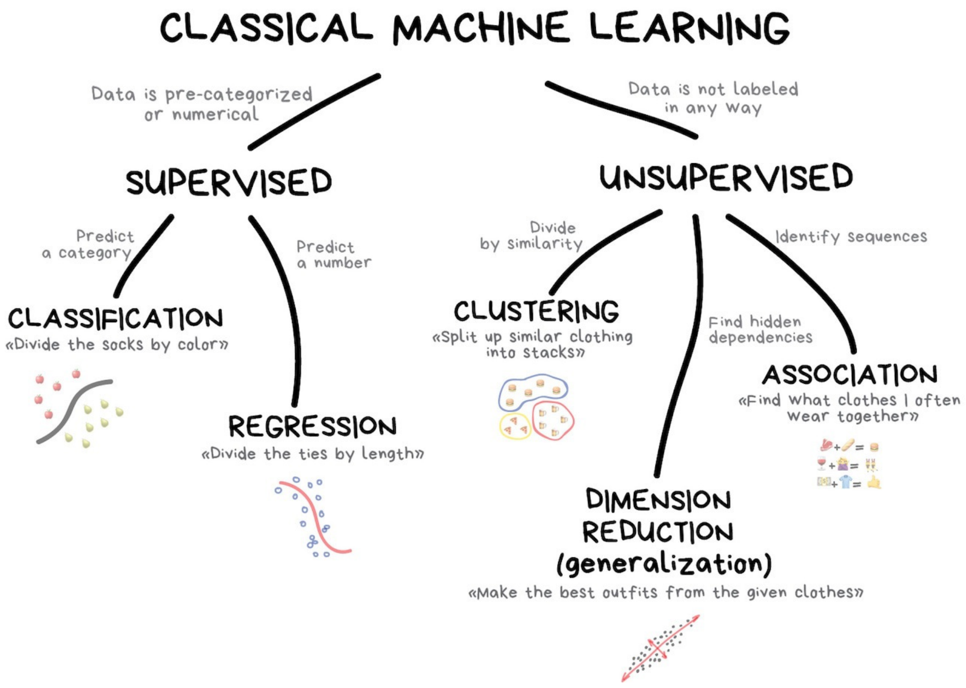


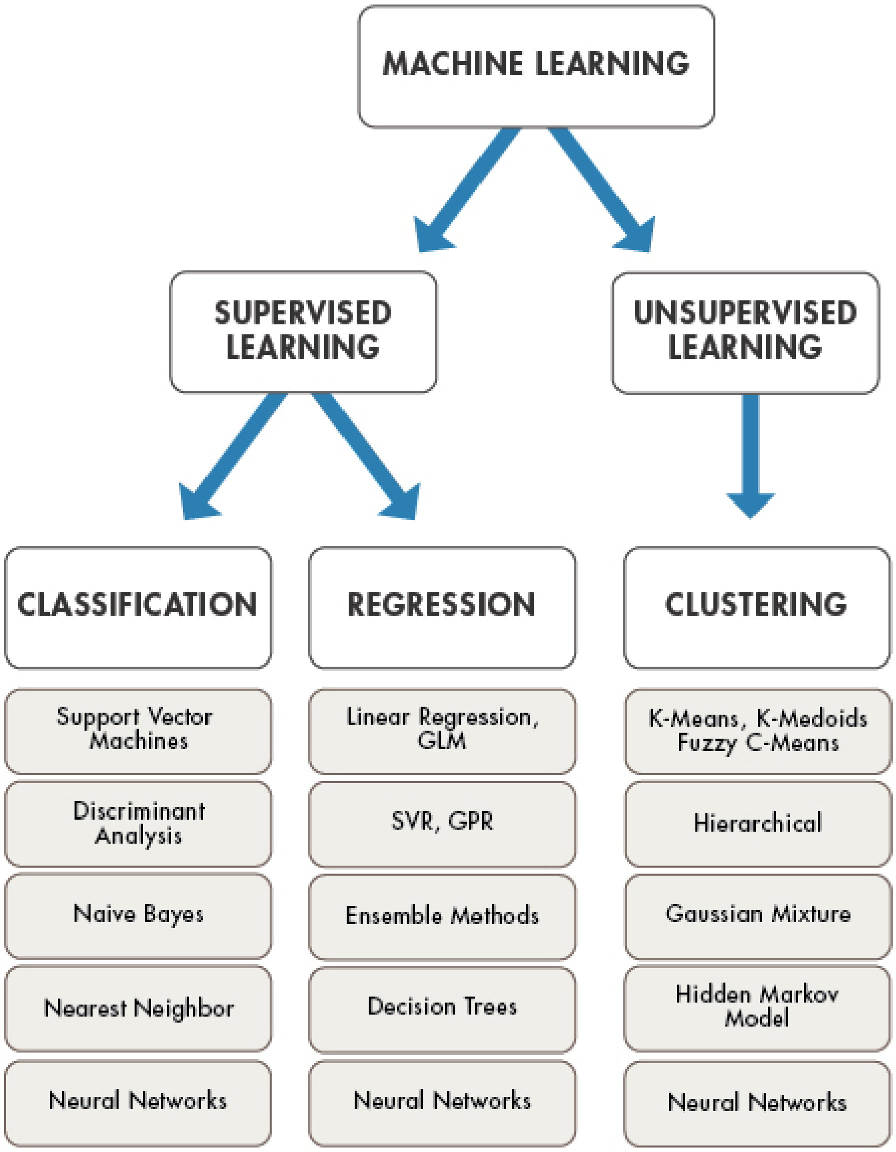


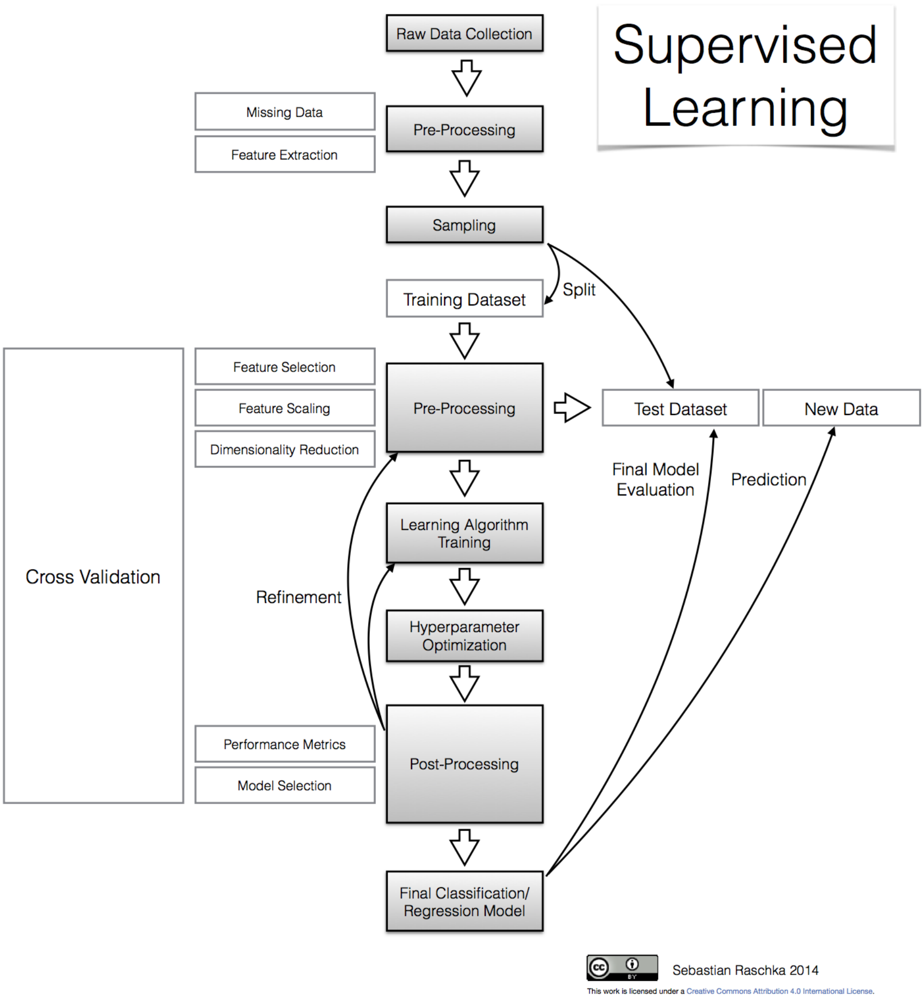
Supervised learning vs Unsupervised learning

Machine learning uses two types of techniques:(image source) -> Supervised learning, which trains a model on known input and output data so that it can predict future outputs. -> Unsupervised learning, which finds hidden patterns or intrinsic structures in input data.




CLASSICAL MACHINE LEARNING
Data is pre-categorized or numerical
Data is not labeled in any way
SUPERVISED
UNSUPERVISED
Predict a category
Predict a number
Divide by similarity
Identify sequences
CLASSIFICATION
«Divide the socks by color»
REGRESSION
«Divide the ties by length»
CLUSTERING
«Split up similar clothing into stacks»
Find hidden dependencies
ASSOCIATION
«Find what clothes I often wear together»
DIMENSION REDUCTION (generalization)
«Make the best outfits from the given clothes»


MACHINE LEARNING
SUPERVISED LEARNING
UNSUPERVISED LEARNING
CLASSIFICATION
REGRESSION
CLUSTERING
Support Vector Machines
Discriminant Analysis
Naive Bayes
Nearest Neighbor
Neural Networks
Linear Regression, GLM
SVR, GPR
Ensemble Methods
Decision Trees
Neural Networks
K-Means, K-Medoids Fuzzy C-Means
Hierarchical
Gaussian Mixture
Hidden Markov Model
Neural Networks



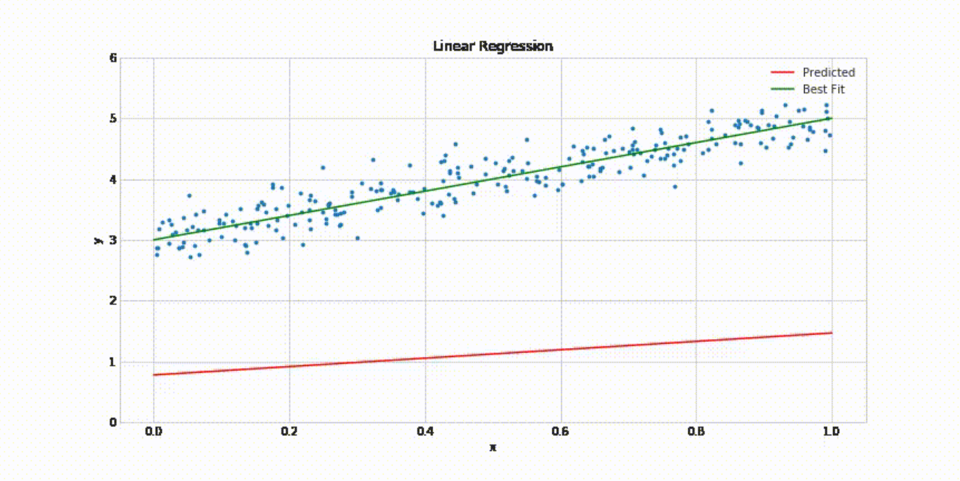
> ## `Supervised Learning` -Regression Problems

> `from sklearn.linear_model import LinearRegression`
> `from sklearn.linear_model import SGDRegressor`
> `from sklearn.linear_model import Ridge`

-> `*model = LinearRegression().fit(inputs, targets)*`
-> `*model = SGDRegressor().fit(inputs, targets)*`
-> `*model = Ridge().fit(inputs, targets)*`
-> `*model.coef_, model.intercept_*`

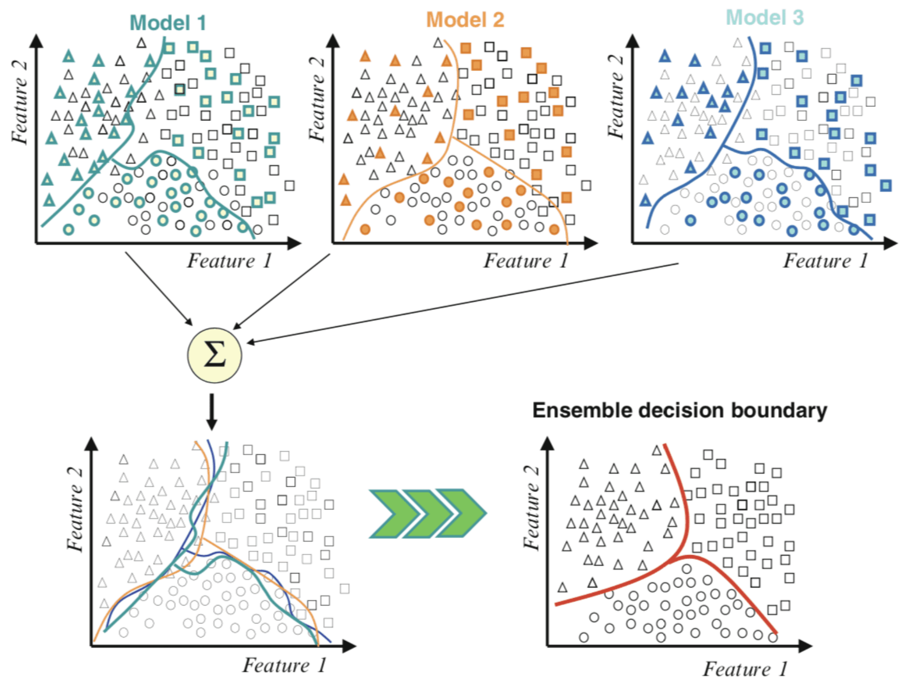
> -> $y = w \times x + b$
>
> - The numbers w and b are called the ***parameters*** or ***weights*** of the model.

Here's a visualization of how gradient descent works:



Random Forest

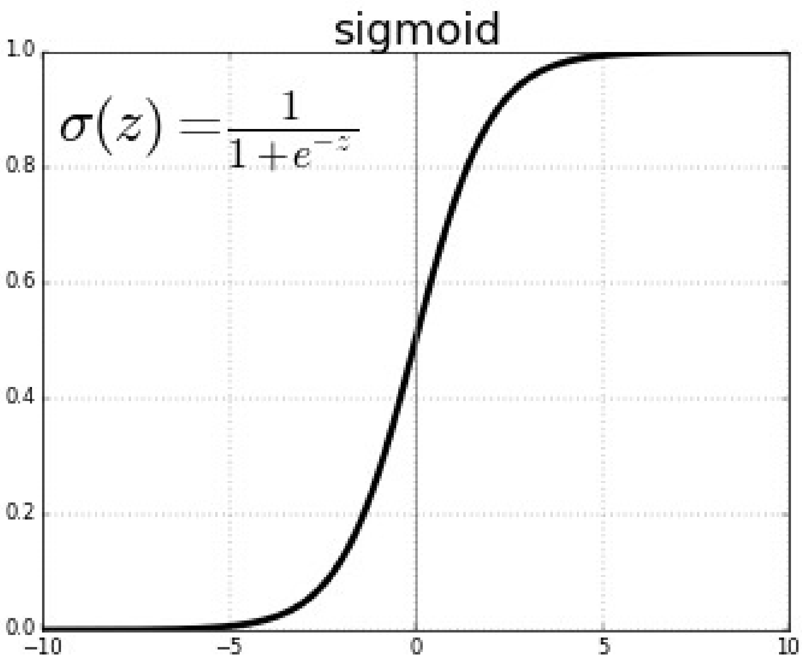
`from sklearn.ensemble import RandomForestClassifier`
<img src="https://miro.medium.com/max/5752/1*5dq_1hnqkboZTcKFfwbO9A.png" width="800>

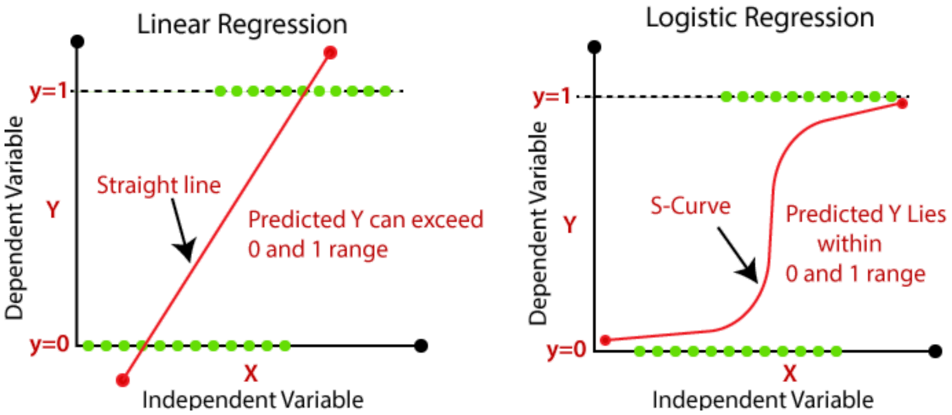


`Supervised Learning` - Classification Problems

Logistic Regression

`from sklearn.linear_model import LogisticRegression`





Descision Tree Classifier

`from sklearn.tree import DecisionTreeClassifier`

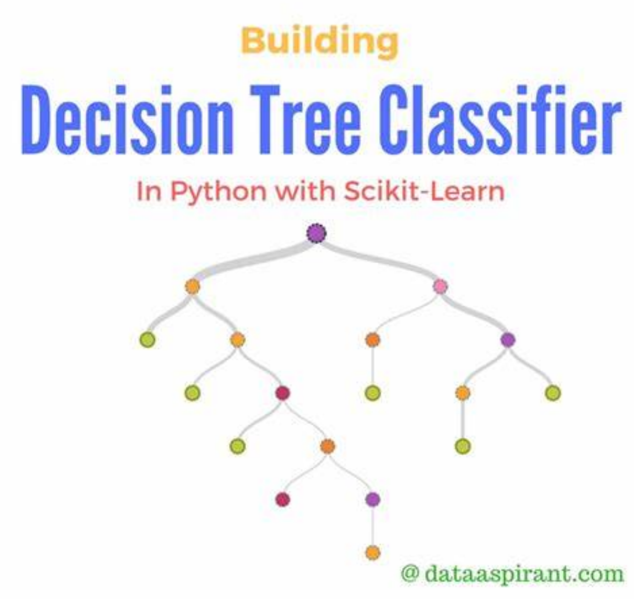
-> *`Feature Important`*

-> *`np.argmax(model.feature_importances_)`*

-> *`df_fi = pd.DataFrame({'Feature': inputs.columns, 'Importance': model.feature_importances_}).sort_values('Importance', ascending=False)`*

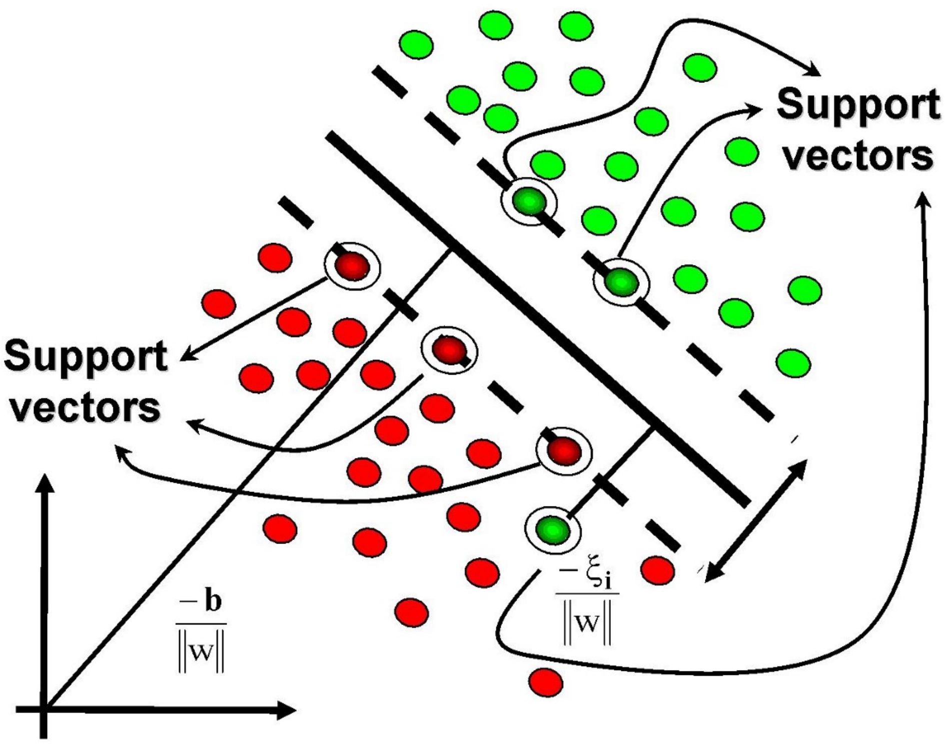
`from sklearn.tree import plot_tree`

`from sklearn.tree import export_text`



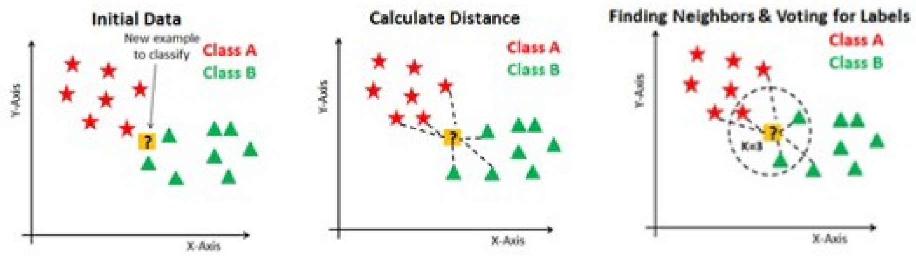
Support Vector Machine

`from sklearn.svm import SVC`

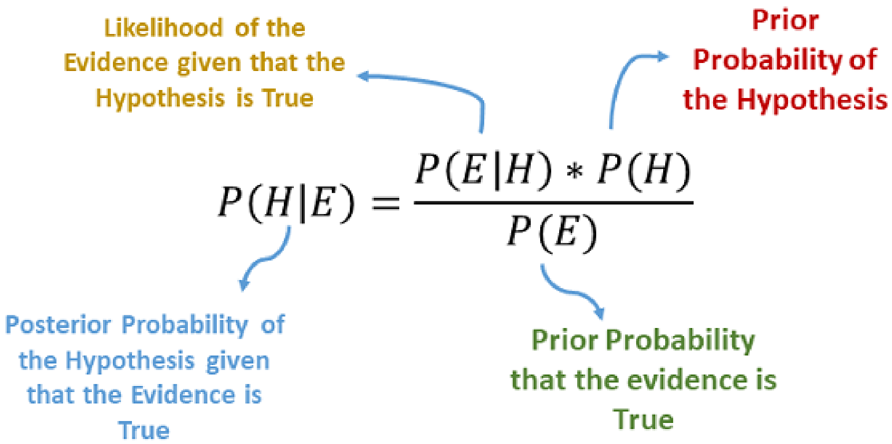


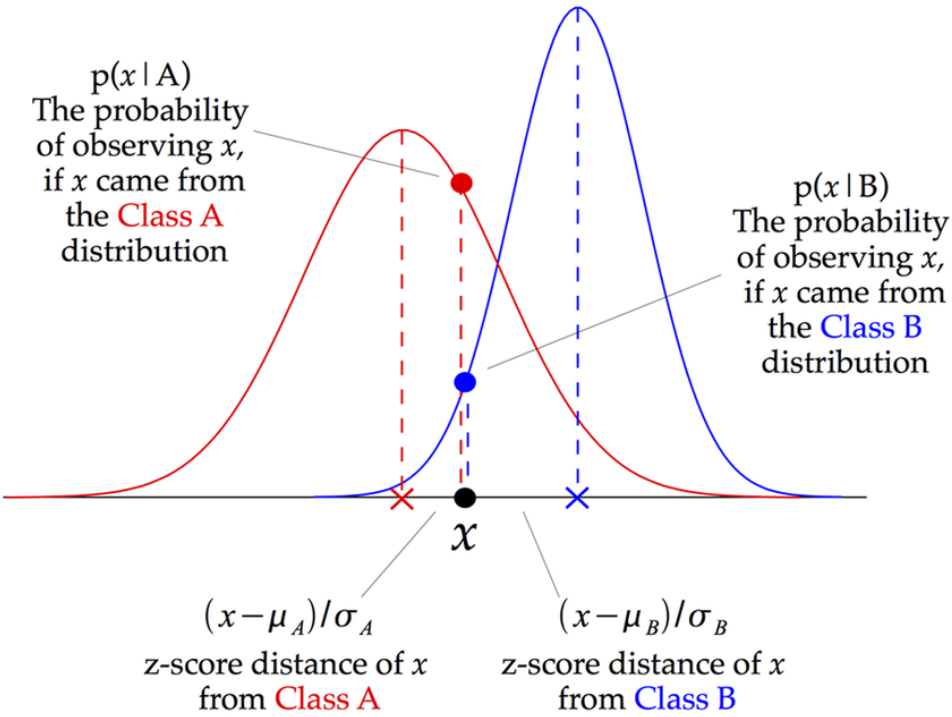
KNN K nearest neighbors Classification

`from sklearn.neighbors import KNeighborsClassifier` -> knn = KNeighborsClassifier(n_neighbors=10)

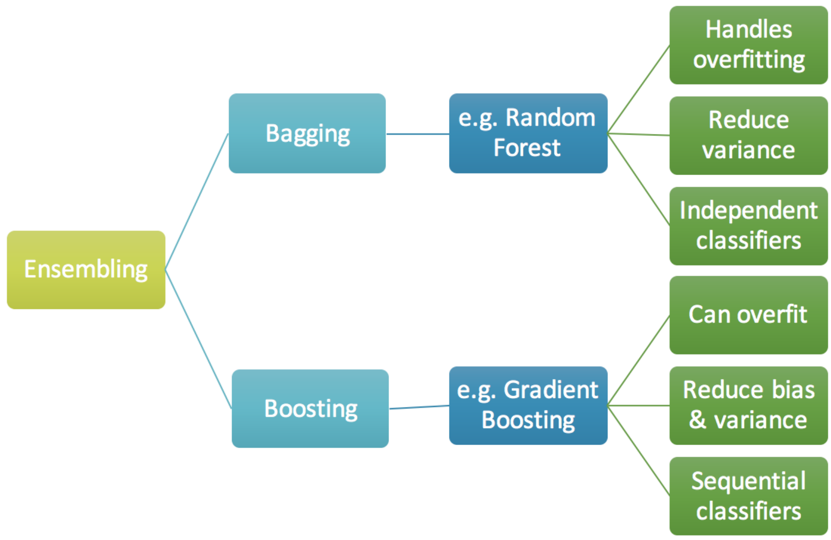


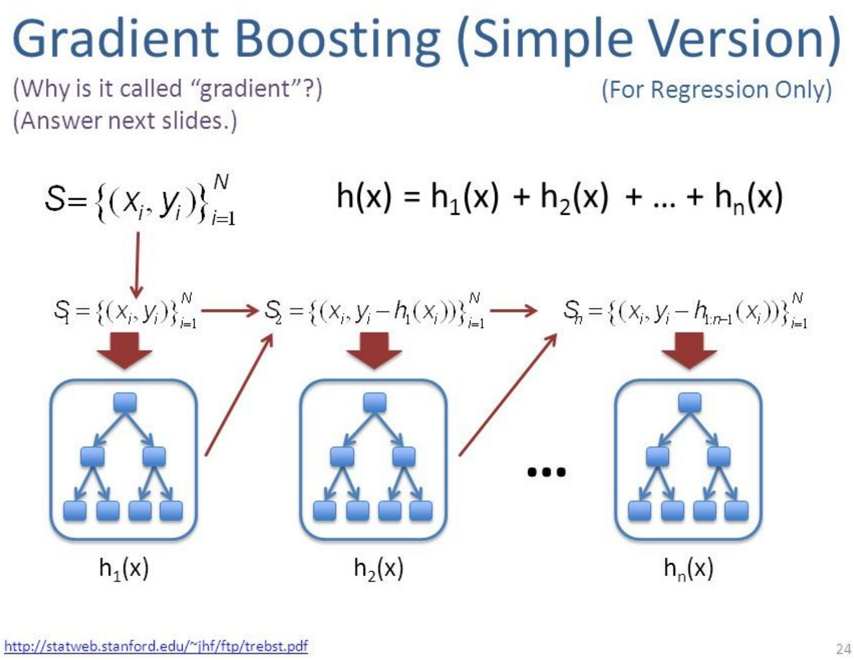
Naive Bayes Classifier

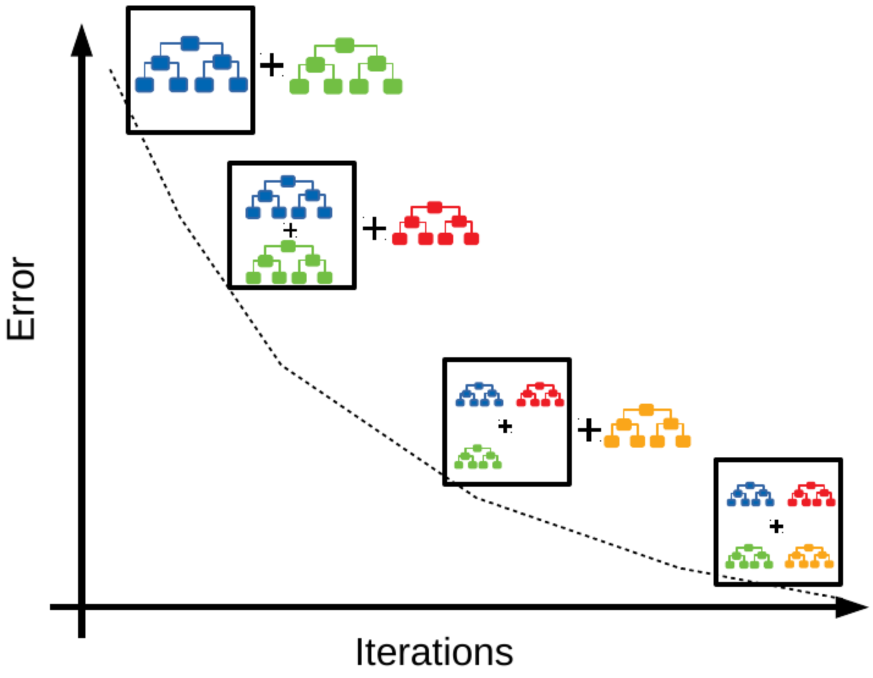




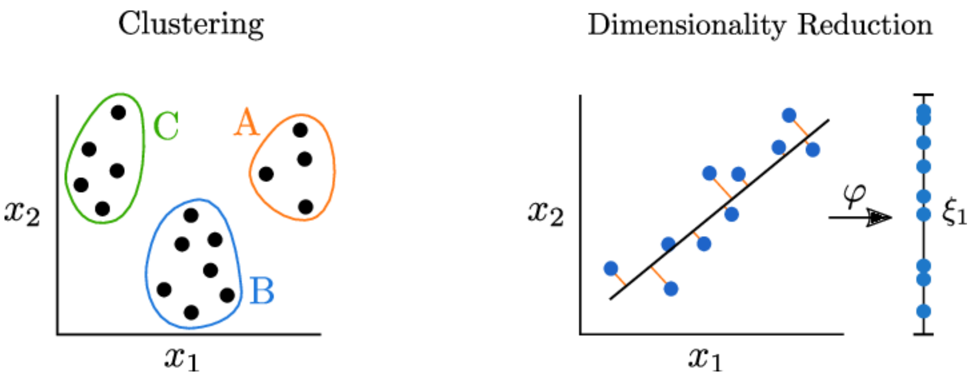
Gradient Boosting Classifier


Ensembling
Bagging
e.g. Random Forest
Handles overfitting
Reduce variance
Independent classifiers
Boosting
e.g. Gradient Boosting
Can overfit
Reduce bias & variance
Sequential classifiers


Gradient Boosting (Simple Version)
(Why is it called "gradient"?)
(Answer next slides.)
(For Regression Only)
$S=\{(x_i, y_i)\}_{i=1}^N$
$h(x) = h_1(x) + h_2(x) + ... + h_n(x)$
$S_1=\{(x_i, y_i)\}_{i=1}^N$
$S_2=\{(x_i, y_i - h_1(x_i))\}_{i=1}^N$
$S_n=\{(x_i, y_i - h_{1:n-1}(x_i))\}_{i=1}^N$
$h_1(x)$
$h_2(x)$
...
$h_n(x)$
http://statweb.stanford.edu/~jhf/ftp/trebst.pdf
24

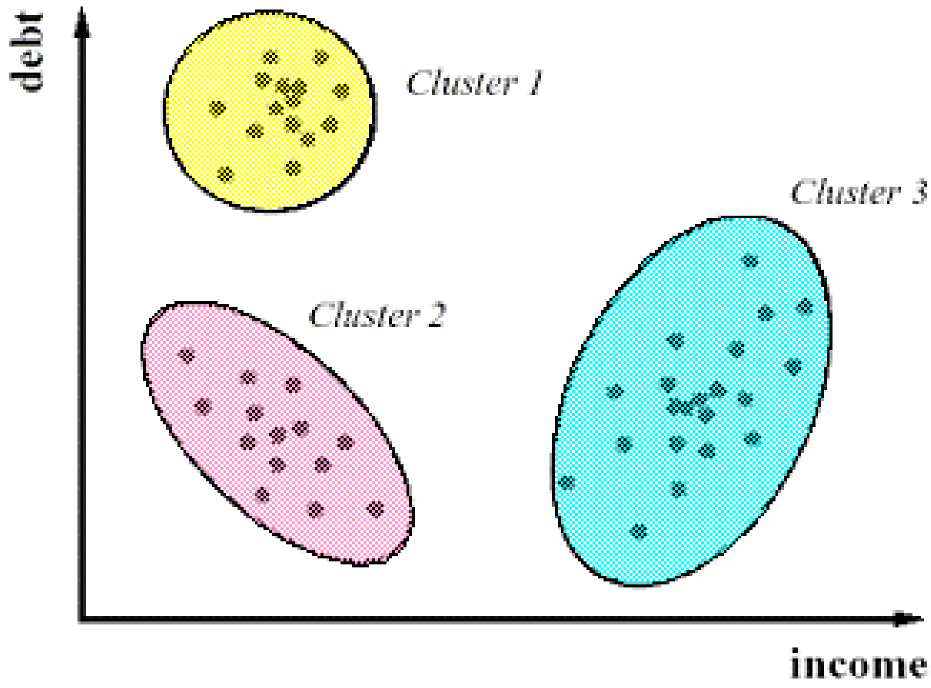


Unsupervised Learning



Clustering

Clustering is the process of grouping objects from a dataset such that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (Wikipedia).
Here is a full list of unsupervised learning algorithms available in Scikit-learn: https://scikit-learn.org/stable/unsupervised_learning.html
Here is a visual representation of clustering:

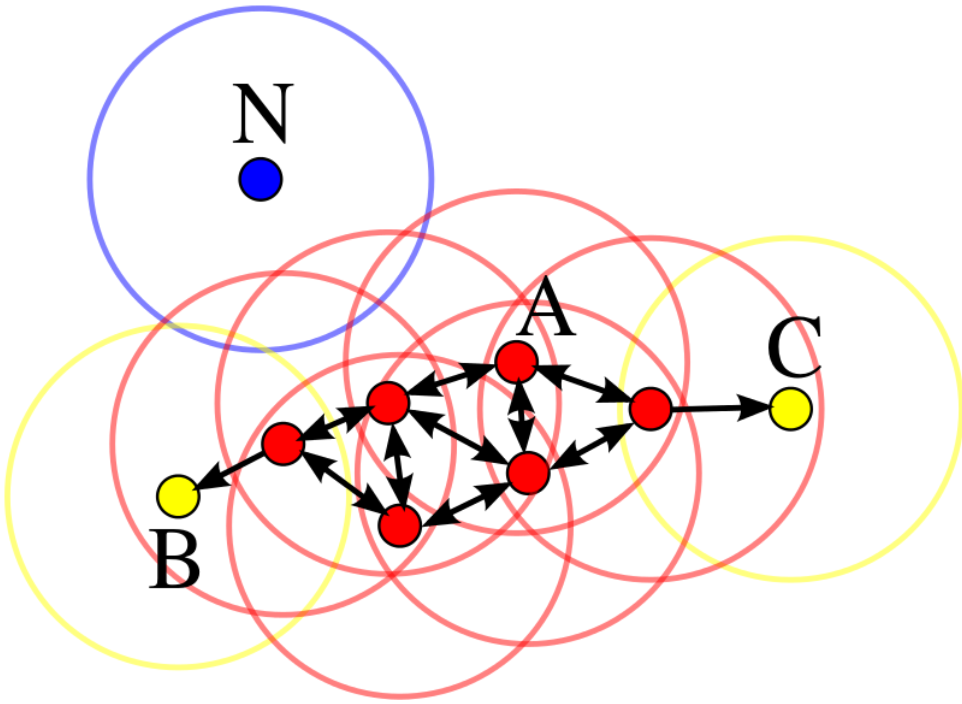


DBSCAN

`from sklearn.cluster import DBSCAN`

-> *model = DBSCAN(eps=0.5, min_samples=4).fit(inputs)*

-> *model.labels_*



K Means Clustering

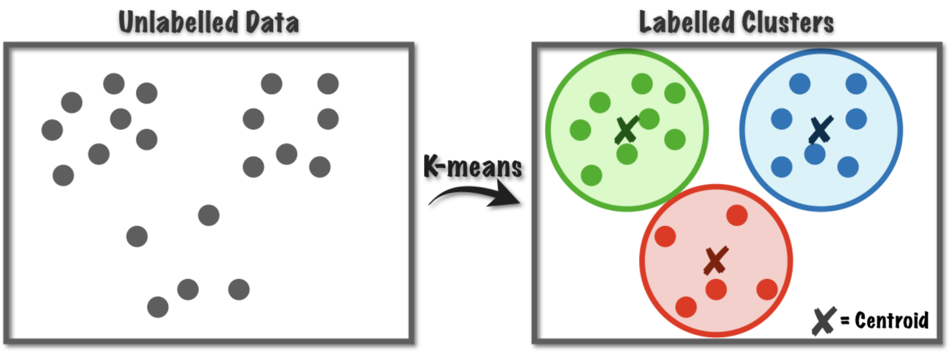
`from sklearn.cluster import KMeans`

-> *model = KMeans(n_clusters=3, random_state=42).fit(inputs)*

-> *model.cluster_centers_*

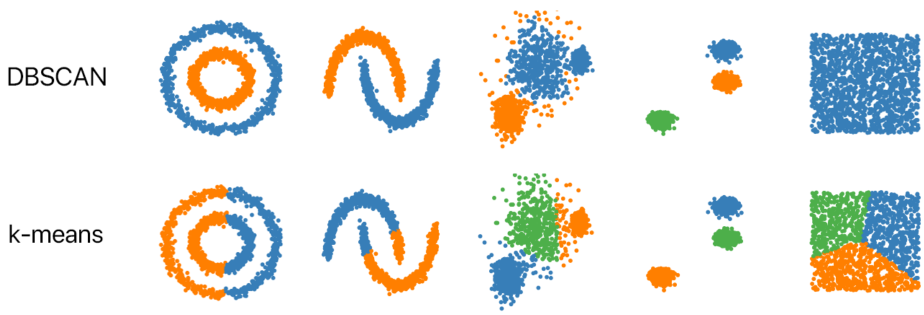
-> *model.inertia_*

The K-means algorithm attempts to classify objects into a pre-determined number of clusters by finding optimal central points (called centroids) for each cluster. Each object is classifed as belonging the cluster represented by the closest centroid.



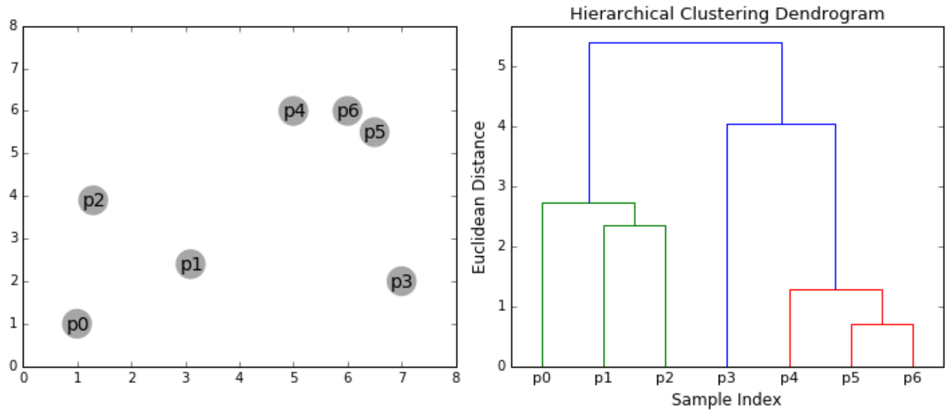
Here's how the K-means algorithm works: 1. Pick K random objects as the initial cluster centers. 2. Classify each object into the cluster whose center is closest to the point. 3. For each cluster of classified objects, compute the centroid (mean). 4. Now reclassify each object using the centroids as cluster centers. 5. Calculate the total variance of the clusters (this is the measure of goodness). 6. Repeat steps 1 to 6 a few more times and pick the cluster centers with the lowest total variance.

Here's how the results of DBSCAN and K Means differ:



Hierarchical Clustering

Hierarchical clustering, as the name suggests, creates a hierarchy or a tree of clusters.

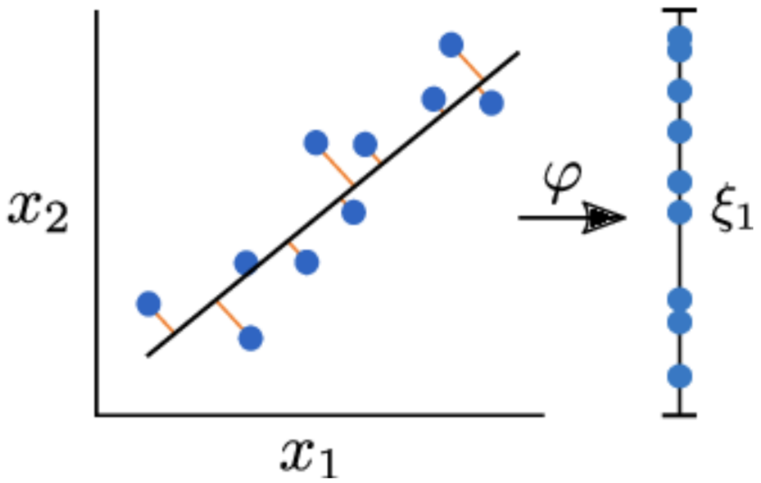


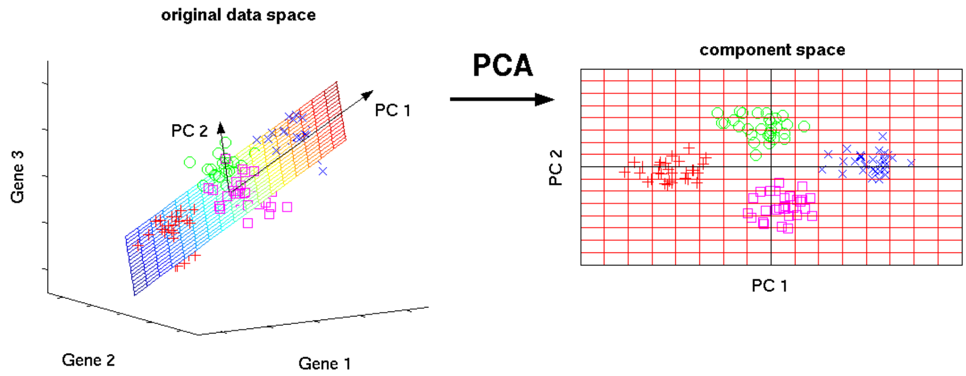
While there are several approaches to hierarchical clustering, the most common approach works as follows: 1. Mark each point in the dataset as a cluster. 2. Pick the two closest cluster centers without a parent and combine them into a new cluster. 3. The new cluster is the parent cluster of the two clusters, and its center is the mean of all the points in the cluster. 4. Repeat steps 2 and 3 till there's just one cluster left.

Principal Component Analysis (PCA)

`from sklearn.decomposition import PCA`
->*model = PCA(n_components=2).fit(X)*
->*model.components_*

Principal component is a dimensionality reduction technique that uses linear projections of data to reduce their dimensions, while attempting to maximize the variance of data in the projection.

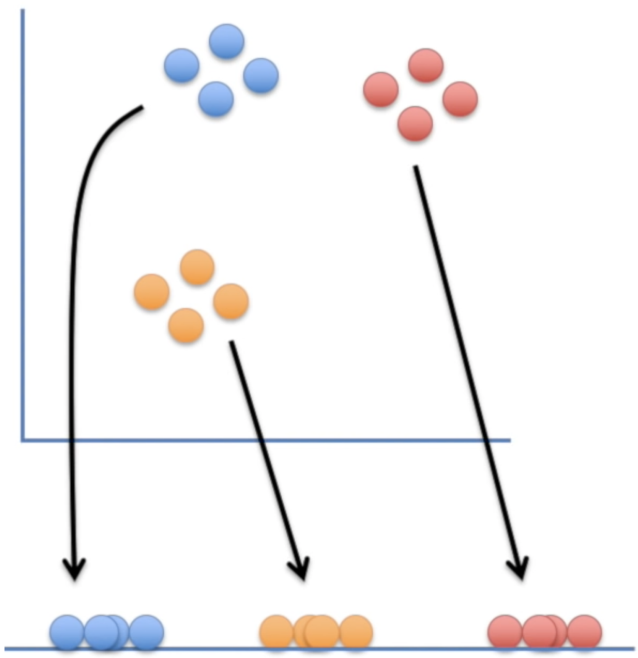




t-Distributed Stochastic Neighbor Embedding (t-SNE)

`from sklearn.manifold import TSNE`
inputs_transformed = TSNE(n_components=2).fit_transform(inputs)

Manifold learning is an` approach to non-linear dimensionality reduction. A commonly-used technique is t-Distributed Stochastic Neighbor Embedding or t-SNE.

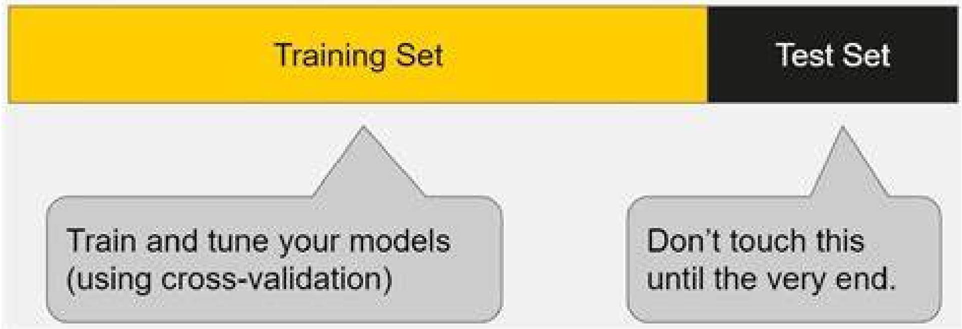


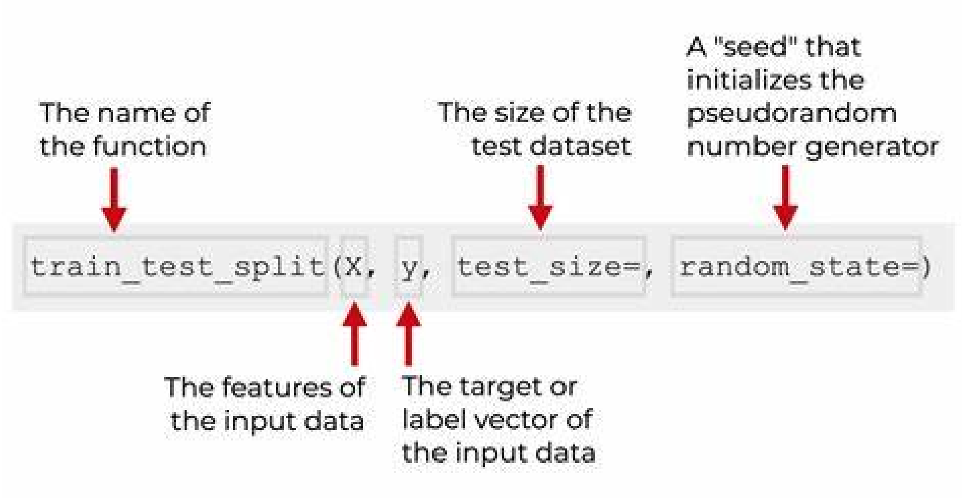
Collaborative filtering with FastAI

Collaborative filtering is perhaps the most common technique used by recommender systems. *Collaborative filtering is a method of making predictions about the interests of a user by collecting preferences from many users. The underlying assumption is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.* - Wikipedia

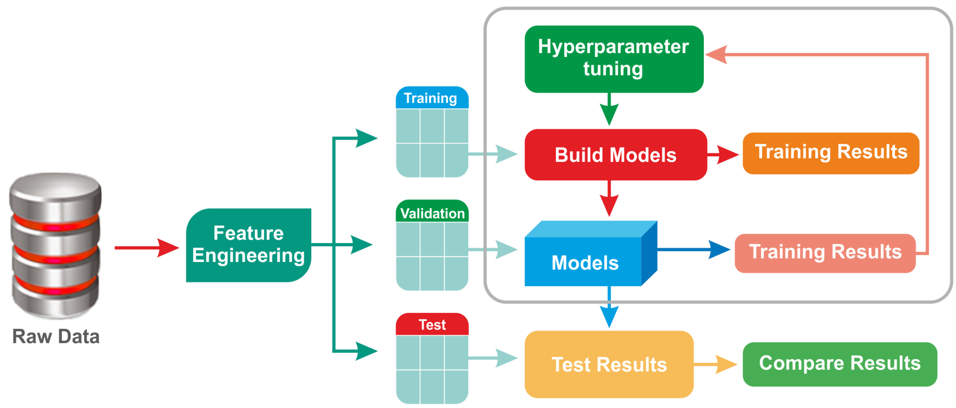
4.2 Training

> `from sklearn.model_selection import train_test_split`
> -> *X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=42)*





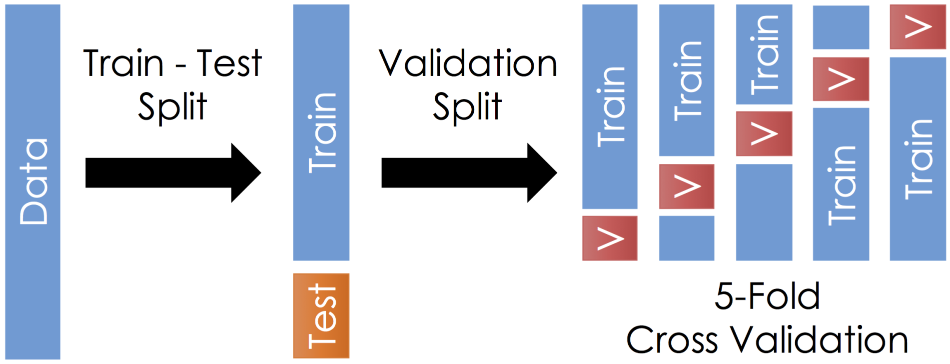
Workflow 5. Evaluating Model



> ## Evaluating Model - K Fold Cross Validation

> `from sklearn.model_selection import KFold`

```
-> *for train_index, val_index in KFold().split(X):*
->     *X_train, X_val = X.iloc[train_index], X.iloc[val_index]*
->     *y_train, y_val = y.iloc[train_index], y.iloc[val_index]*
```



Evaluating Model - Score

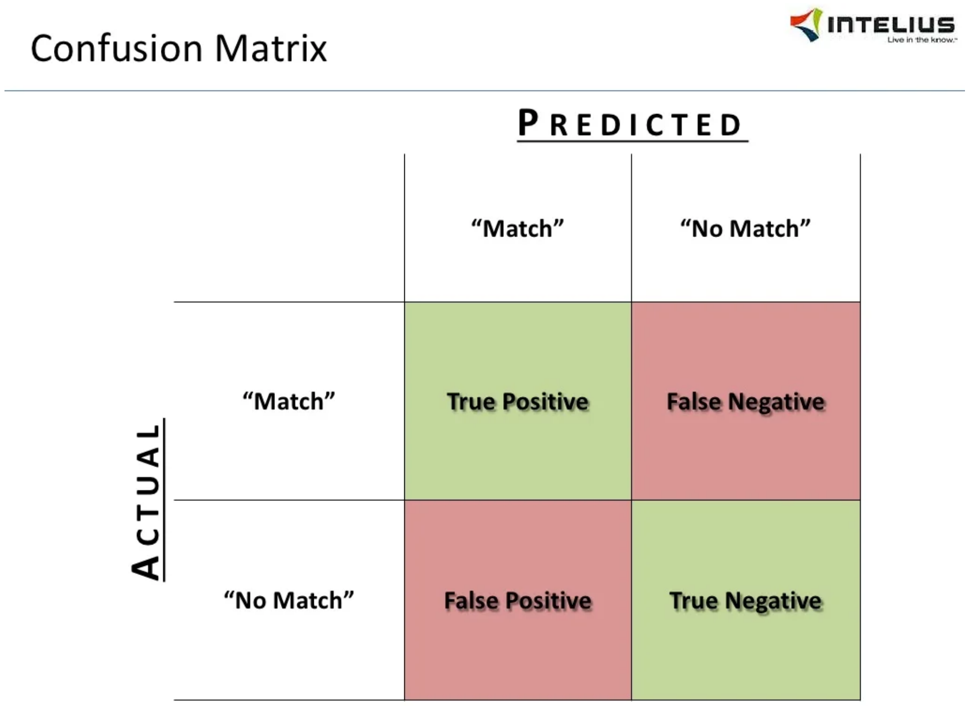
-> *model.score(inputs, targets)*

`from sklearn.metrics import accuracy_score`

-> *accuracy_score(targets, predictions)*

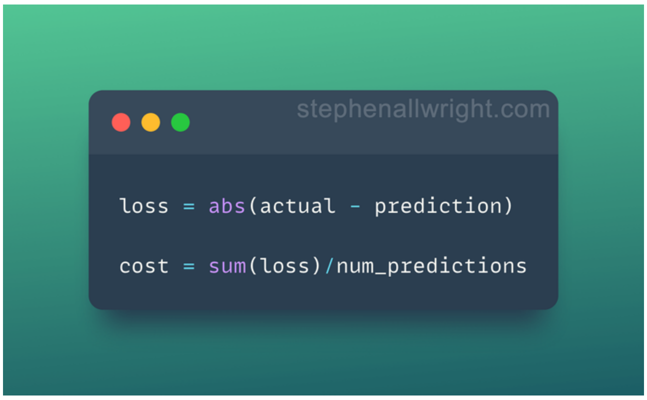
Evaluating Model - confusion_matrix - Sensitivity and Specificity

`from sklearn.metrics import confusion_matrix`

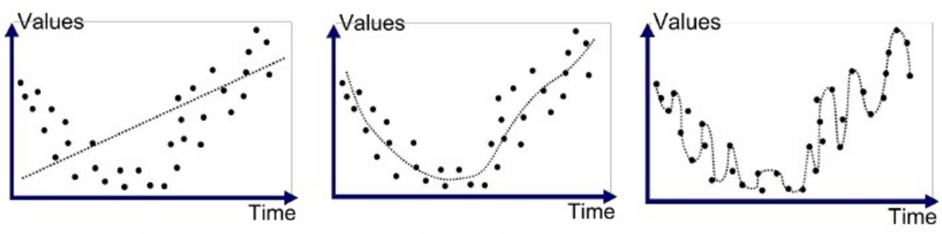


		Predicted	
		Negative (**N**) **-**	Positive (**P**) **+**
Actual	Negative **-**	True Negatives (**TN**)	False Positives (**FP**) **Type I error**
	Positive **+**	False Negatives (**FN**) **Type II error**	True Positives (**TP**)

Evaluating Model - Loss/Cost Function



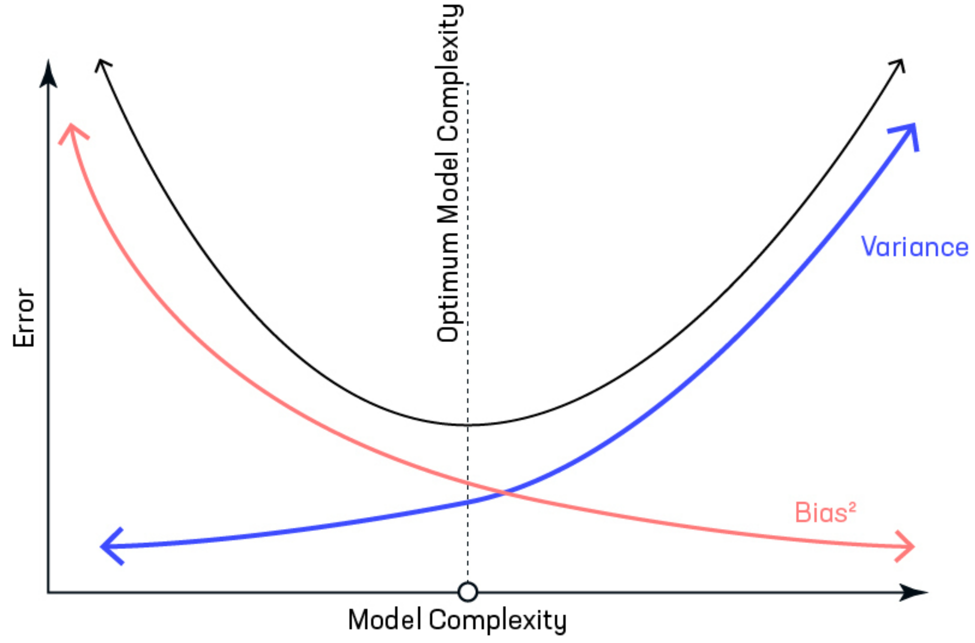
Overfitting vs. Underfitting



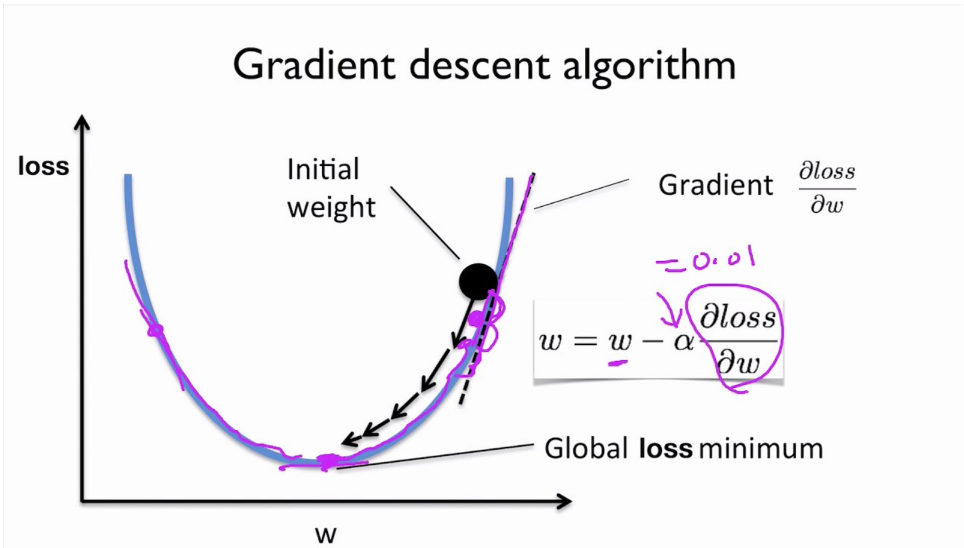
Underfitted Good Fit/Robust Overfitted

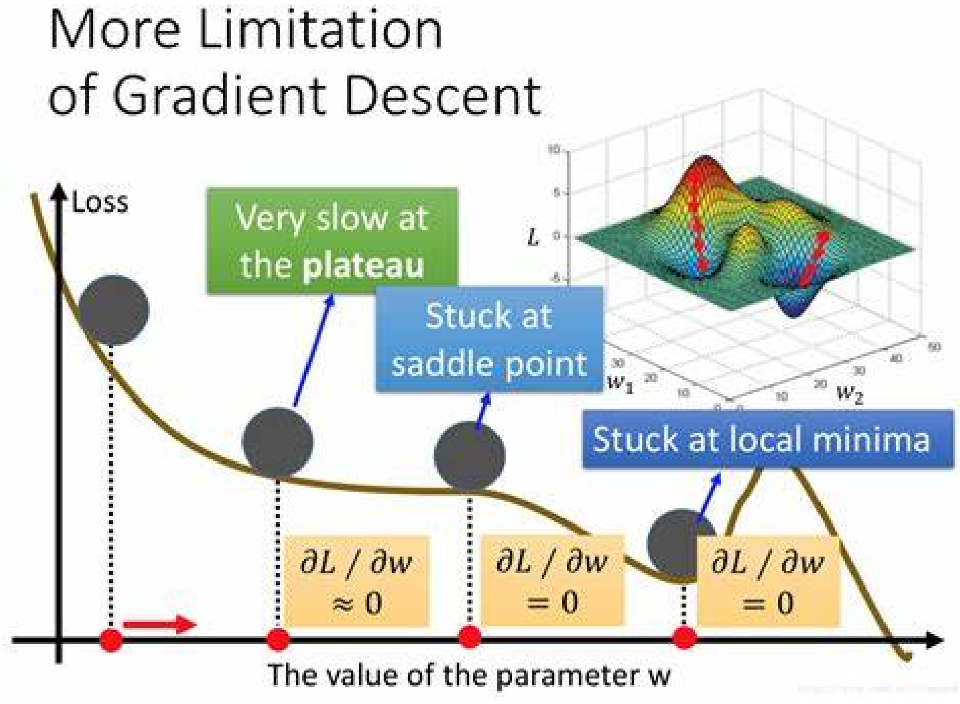
<img src="https://i.imgur.com/EJCrSZw.png" width=480

Bias vs. Variance



Gradient Descent





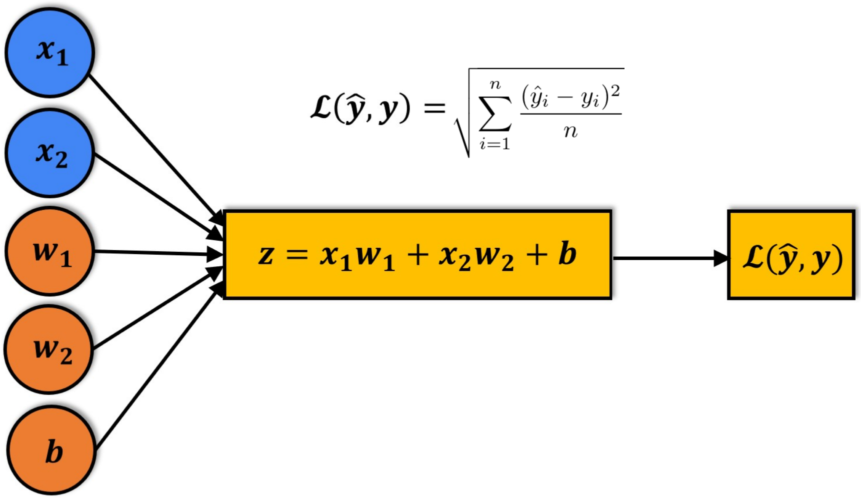
Regression Problems

`RMSE`

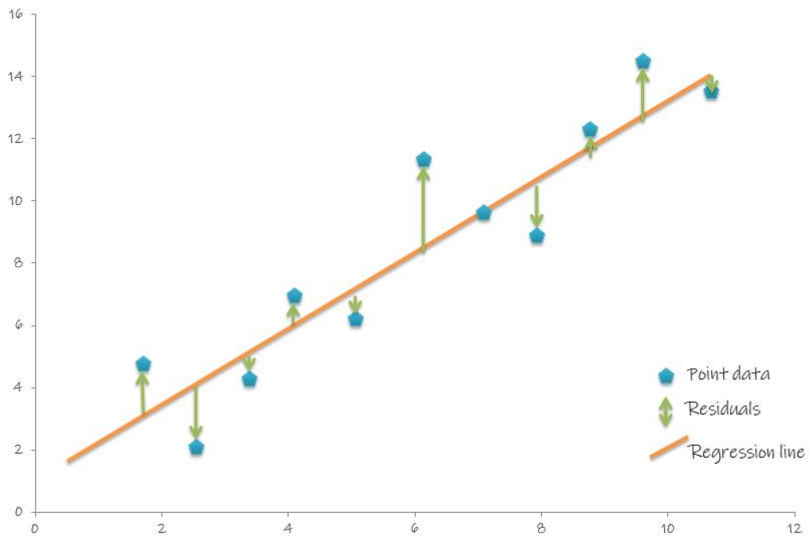
-> `np.sqrt(np.mean(np.square(targets - predictions)))`

-> `from sklearn.metrics import mean_squared_error`

-> `mean_squared_error(targets, predictions, squared=False)`

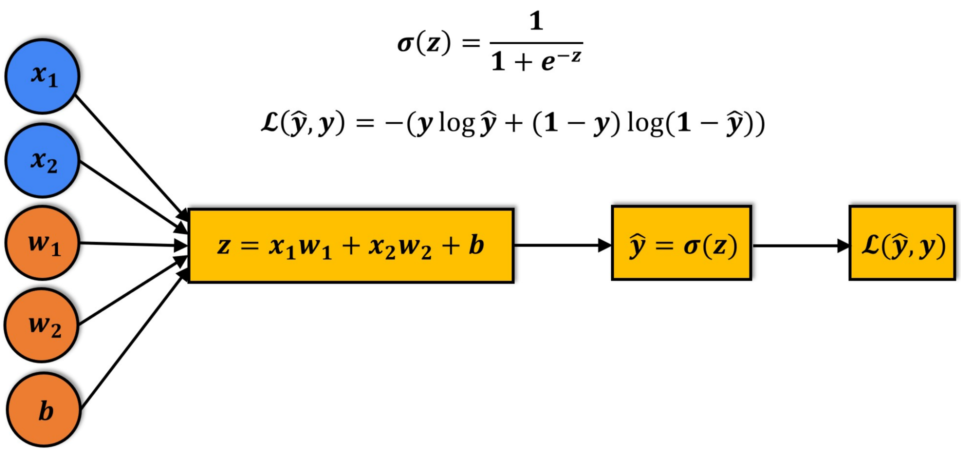


Geometrically, the residuals can be visualized as follows:



Classification Problems

`cross entropy loss function`



L1and L2 Regularization

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

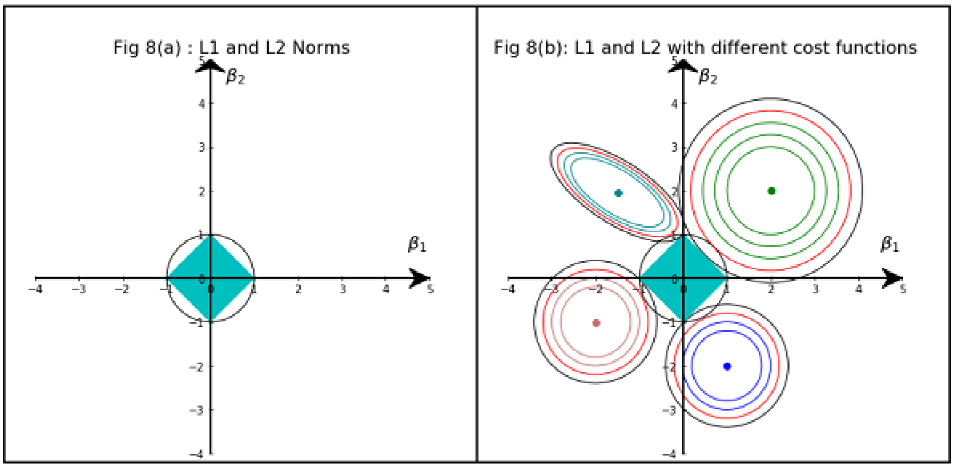
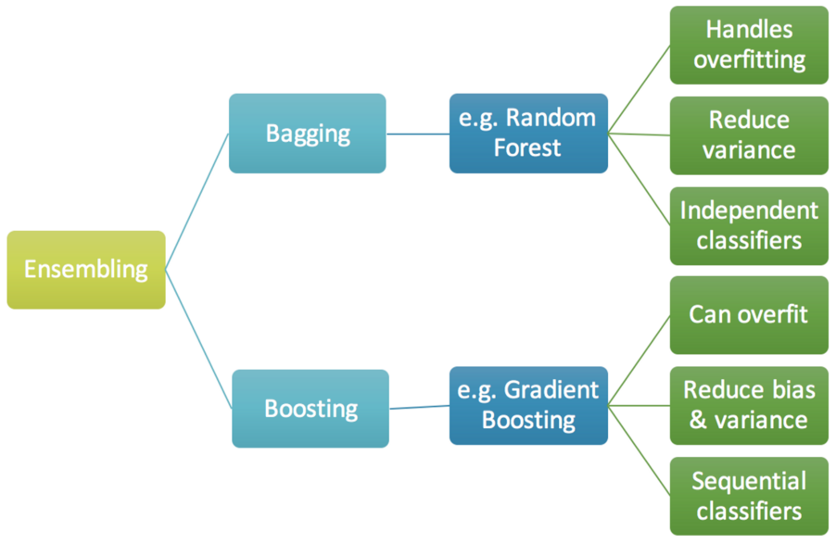


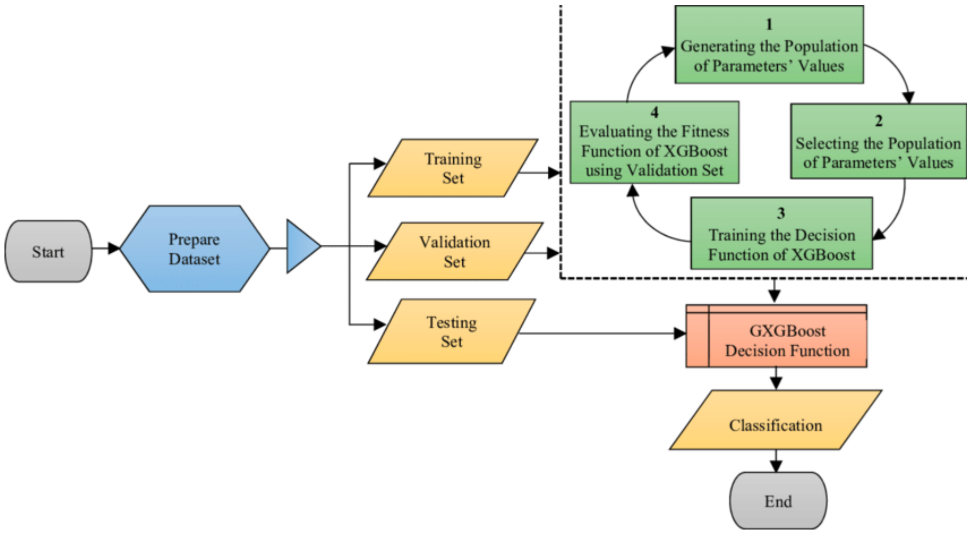
Fig 8(a) : L1 and L2 Norms
Fig 8(b): L1 and L2 with different cost functions

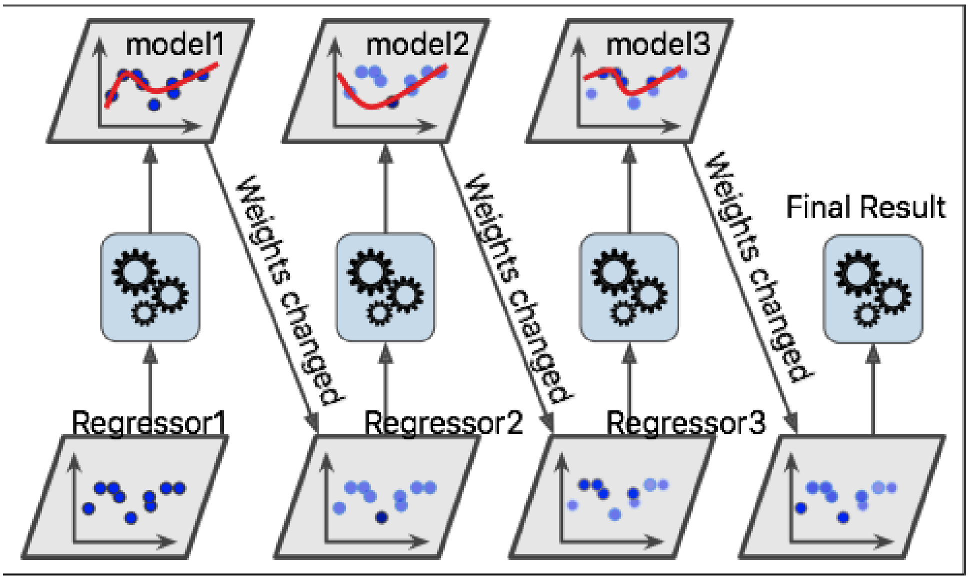
Gradient Boosting

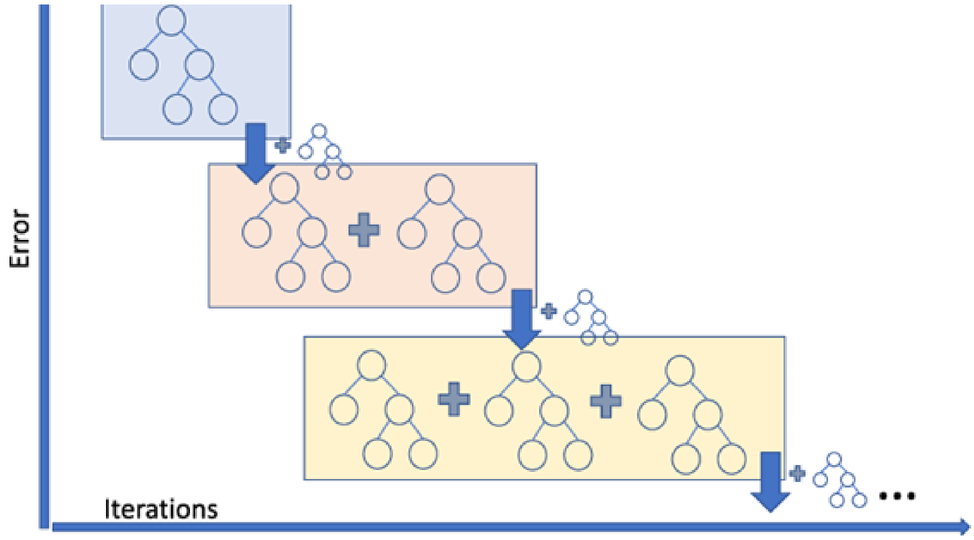
The term "gradient" refers to the fact that each decision tree is trained with the purpose of reducing the loss from the previous iteration (similar to gradient descent). The term "boosting" refers the general technique of training new models to improve the results of an existing model.

- Video Tutorials on StatQuest


Ensembling
Bagging
e.g. Random Forest
Handles overfitting
Reduce variance
Independent classifiers
Boosting
e.g. Gradient Boosting
Can overfit
Reduce bias & variance
Sequential classifiers


Start
Prepare Dataset
Training Set
Validation Set
Testing Set
1 Generating the Population of Parameters' Values
2 Selecting the Population of Parameters' Values
3 Training the Decision Function of XGBoost
4 Evaluating the Fitness Function of XGBoost using Validation Set
GXGBoost Decision Function
Classification
End


model1
model2
model3
Weights changed
Weights changed
Weights changed
Final Result
Regressor1
Regressor2
Regressor3



Workflow 6. Save Model Using Joblib And Pickle

> `import pickle`
> -> *pickle.dump(model, "path/file")*
> -> *pickle.load("path/file")*
>
> `import joblib`
> -> *joblib.dump(model, "path/file")*
> -> *joblib.load("path/file")*

Approach	PROS	CONS
Pickle	Quick Implementation Easy Readability	Python version issues. No stored results or data Only File Object
Joblib	Quick Implementation Easy Readability Accepts Objects and String filenames Different compression options.	Python version issues.
Proprietary	More Flexibility No Python version issues	Harder Readability Complexity

---

-- Memo End --