

ML Notes:-

Contents

| | |
|---|----|
| Definition & Applications | 2 |
| 3 Types of Machine Learning | 2 |
| Broad Data Specialisations | 2 |
| Stability In Machine Learning Models | 3 |
| Reducible Errors in Machine Learning Models | 3 |
| Machine Learning Workflow | 4 |
| Dependencies Involved in Machine Learning | 5 |
| Python Codes for Exploratory Data Analysis..... | 5 |
| Steps in Pre-processing | 6 |
| Tools To Evaluate Model Performance | 6 |
| Supervised Learning | 7 |
| Linear Regression Model..... | 7 |
| Logistic Regression Model | 9 |
| Gaussian Naive Bayes Model | 11 |
| Decision Trees Model | 13 |
| K- Nearest Neighbour Model | 14 |
| Cross Validation | 16 |
| Hyperparameter Tuning | 18 |
| Ensemble Techniques | 19 |
| Unsupervised Learning - Clustering | 21 |
| K Means Clustering | 21 |
| Hierarchical Clustering | 23 |
| DBSCAN Clustering | 26 |
| Bonus: Feature Extraction (Pre-processing step) | 28 |

Definition & Applications

The field of study that gives computers the ability to learn from data without being explicitly programmed.

Basically: The human provides the machine with input data to train it based on an algorithm. It learns from this data. The machine is then given new and unseen data, on which it makes predictions. The human then evaluates the accuracy of this model.

Applications:

- ⑨ Self-driving cars,
 - ⑨ Forecasting company revenue based on performance metrics,
 - ⑨ customer segmentation,
 - ⑨ analysing employee attrition (based on years in the company, salary level, age, etc),
 - ⑨ AI chatbots
-

3 Types of Machine Learning

Supervised Learning: Here, the training data is labelled: means the Target column is present (which helps us make a final decision)*

Unsupervised Learning: Training data is unlabelled- the system tries to learn without the target column

Reinforcement Learning: the agent (machine) learns from the environment and its experience of rewards and penalties: it learns to avoid penalties and finally we get a policy model.

*Deep Learning: Deep learning uses supervised learning in situations such as image classification or object detection, as the network is used to predict a label or a number (the input and the output are both known). As the labels of the images are known, the network is used to reduce the error rate, so it is "supervised".

Broad Data Specialisations

Here are the most common job titles in the field, along with the main tasks performed by each of them:

1. **Data Engineer:** Data collection and data cleaning
 2. **Data Analyst:** Data cleaning and exploratory data analysis
 3. **Machine Learning Engineer:** Model building and model deployment
 4. **Data Scientist:** Performs all the above tasks
-

Stability In Machine Learning Models

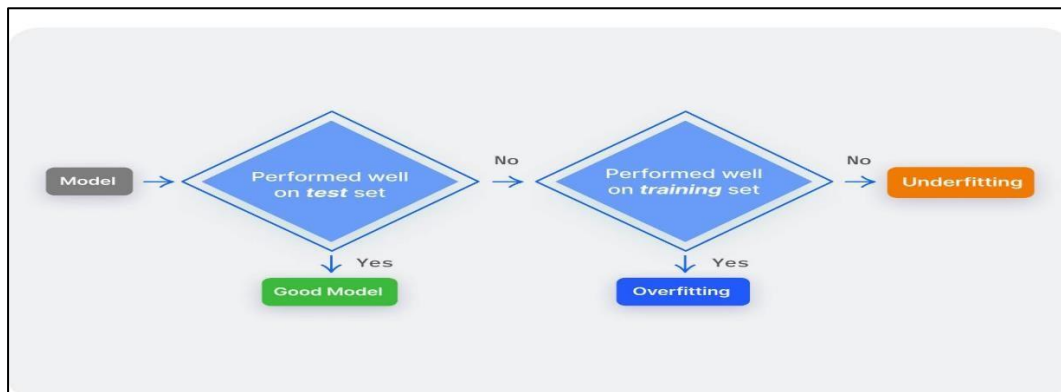
Stability is the most important feature of any ml model. This stability is gained by reducing 2 major errors in the model: Bias and variance. An ideal ml model should have low bias and low variance.

Bias: Inability of a model to capture the true relationship between target and predictors. Bias could happen due to class imbalance.

For example, if we have a gender column where Male class makes up 99% of records, and female class makes up only 1% of records - most predictions would be biased to result in male- hence the predictions are unreliable. If 50% is male and 50% female (balanced target feature): then target values have equal likelihood to be predicted for our test set.

Variance: Refers to the variation in the train and test performance. If train score is higher than test score, then this occurs because of high variance (usually test score is not higher than train score). Ideally, train and test scores should be the same - means our model generalises over unseen data relatively well. When the model makes predictions on X_{test} , it is predicting the target as well as it did for seen data. Basically, low variance means that the data can generalise well (predict well) on unseen data – which is our goal.

Reducible Errors in Machine Learning Models



1. **Overfitting:** If the training set fits the data very well (high R score) but testing set does not fit the data that well => due to high variance. Leads to Good training performance but poor performance for unseen data. It captures every single aspect of our data, including the noise (outliers). Because it fit too specifically to our training data, it will not perform well while predicting unseen data

Real life Example of overfitting: Ordering food from Domino's that reaches you 30 minutes late, and assuming Dominos always delivers late.

How To Fix Overfitting:

- (A) Cross validation
 - (B) Increase training dataset
 - (C) Reduce the noise in training data (remove outliers)
 - (D) Regularisation: Constraining the model to simplify the model: reduce number of independent features: use hyperparameter tuning (eg: gridsearch)
 - (E) Perform ensemble methods like bagging, boosting, stacking.
2. **Underfitting:** Model is not able to learn enough from the training data. Model is too simple to learn the true relationship in the data. Happens because of high bias. This results in low performance of training set.

How To Fix Underfitting:

- (A) Select a more powerful and complex model
- (B) Increase the number of independent features
- (C) Reduce the constraints on the model (reduce the hyperparameter regularisation)

Machine Learning Workflow

1. Import the dependencies
2. Load the dataset
3. Perform basic exploratory data analysis
4. Perform pre-processing: convert raw data into clean data
5. Split the data into train and test

6. Create the algorithm model object
7. Fit the algorithm to the training dataset (x_train, y_train)
8. Check the performance of the training set (r2 or accuracy)
 - ⑨ if overfitting (high variance) – perform cross validation (splitting techniques) to prevent overfitting
 - ⑨ if data is biased, use boosting or stacking to improve model performance
9. Generate predictions on test data (create y_pred from x_test) 10. Evaluate the model predictions

Dependencies Involved in Machine Learning

Here are the major python libraries that will be used during the entire machine learning workflow: a. Pandas

- b. From pandas_profiling import ProfileReport
- c. Numpy
- d. Matplotlib
- e. Seaborn (heatmap)
- f. Sklearn.preprocessing: LabelEncoder, OneHotEncoder, OrdinalEncoder, StandardScaler, MinMaxScaler
(*Normalization- minmax scaler*- typically means rescales the values into a range of [0,1]. *Standardization – standard scaler (z score)* - typically means rescales data to have a mean of 0 and a standard deviation of 1 (unit variance).
- g. Sklearn.model_selection: train_test_split, KFold, StratifiedKFold, RepeatedKFold, cross_val_score, GridSearchCV, RandomizedSearchCV, StratifiedRepeatedKFold
- h. Sklearn.metrics: confusion_matrix, accuracy_score, recall_score, f1_score, precision_score, roc_auc_score
- i. from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
- j. from sklearn.naive_bayes import GaussianNB
- k. from sklearn.linear_model import LinearRegression, LogisticRegression
- l. from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
- m. from sklearn.ensemble import AdaBoostClassifier, AdaBoostRegressor, RandomForestClassifier, RandomForestRegressor, StackingClassifier, StackingRegressor

Python Codes for Exploratory Data Analysis

- **df.info()**
- **df.shape[0]** and **df.shape[1]** : number of records (0 gives rows) and number of features(1 gives columns)
- **df.describe()** : statistical summary
- **df.dtypes**
- **ProfileReport(df)**: automated exploratory data analysis
- **Visualisations for important features** (can do after analysing heatmap)

Steps in Pre-processing

† **MISSING VALUE TREATMENT**

- Checking null values
- Replacing null values with mean/median(int features) or mode (categorical features)

† **FEATURE ENGINEERING** –The science and art of extracting more information from existing features of a dataset. Important because machines work more efficiently with numerical data

- Data encoding: labelencoder/pd.get_dummies/OneHotEncoder/OrdinalEncoder
- Add these encoded columns to df, and drop the categorical columns
- Includes categorical encoding, feature scaling, missing value treatment, outlier removal.

† **FEATURE SCALING** – So that no feature outweighs another in terms of importance regarding target variable. Improves the model performance.

- Use normalisation: MinMaxScaler OR standardisation: StandardScaler
- Both **StandardScaler** and **MinMaxScaler** are very sensitive to the presence of outliers.

NOTE: All the tree-based algorithms — **CART, Random Forests, Gradient Boosted Decision Trees** - **do not require scaling.**

† **FEATURE REDUCTION:** Process of reducing the number of features under consideration by obtaining a set of important features.

- **Reduction is of 2 types:**
 - **Feature Extraction** (using LDA or PCA)(This is the method for creating a new and smaller set of features that captures most of the useful info from the raw data).
 - **Feature Selection** (finding relevant features using **correlation**- heatmap).

† **SPLIT DATA INTO X AND Y:** X represents all the independent features. Y represents the dependent (target) feature.

Tools To Evaluate Model Performance

| ML REGRESSORS | ML CLASSIFIERS |
|---------------|------------------|
| MAE | Confusion Matrix |

| | |
|----------------------|----------------------|
| MSE | Accuracy |
| RMSE | Precision |
| R SQUARED | Recall (Sensitivity) |
| Regplot/ Scatterplot | ROC-AUC Curve |
| | F1 Score |

Supervised Learning

| ML REGRESSORS | ML CLASSIFIERS |
|--------------------------|---------------------------|
| Linear Regression | Logistic Regression |
| | Gaussian Naïve Bayes |
| Decision Tree Regressor | Decision Tree Classifier |
| Random Forest Regressor | Random Forest Classifier |
| Support Vector Regressor | Support Vector Classifier |
| KNN Regressor | KNN Classifier |

Linear Regression Model

Definition: This model creates a best fit line using the mathematical equation of $y = mX + c + e$ where m is coefficient of determination (slope), c is y intercept and e is the error(residual). The model aims to minimise the errors in prediction – tries to minimise the distance between the predicted target scores and the actual test target scores. (y_{pred} vs y_{test})

- This model has a **numerical (continuous) target feature** (eg: price of a car)
- This model aims for high R-squared scores and minimal errors
-

Assumptions:

1. **Linearity:** The relationship between X and the mean of Y is linear.
2. **Homoscedasticity:** Residuals have constant variance at every level of x . This is known as homoscedasticity. When this is not the case, the residuals are said to suffer from heteroscedasticity. Heteroscedasticity is present in a regression analysis, the results of the analysis become hard to trust. **Specifically, heteroscedasticity increases the variance of the regression coefficient estimates, but the regression model doesn't pick up on this.** This makes it much more likely for a regression model to declare that a term in the model is statistically significant, when in fact it is not.

3. **Independence:** Observations are independent of each other.
4. **Normality:** The errors of the model are normally distributed.

| Advantages | Disadvantages |
|---|--|
| Short learning time | Model prone to overfitting for high dimensional data |
| Works with numeric attributes | We have to perform feature selection manually |
| If number of columns is large, the predictions are powerful | Sensitive to outliers |

Regressor Evaluation:

(i) Coefficient of determination (R^2)

R-squared measures how well the model explain the variation in the data. Basically, how well the model performs. (Good fit or not).

Ranges from -infinity to 1

$$R^2 = 1 - \text{RSS/TSS or}$$

$$R^2 = 1 - (\text{actual} - \text{predicted})^2 / (\text{Squared sum of deviation from the mean})$$

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \equiv 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

$$\downarrow$$

$$R^2 = \frac{SS_{\text{reg}}}{SS_{\text{tot}}}$$

(ii) Mean Squared Errors (MSE), Root Mean Squared Errors (RMSE), MAE (Mean Absolute Error), Mean Absolute Percentage Errors (MAPE)

- A. RMSE and MSE works on the principle of averaging the errors
- B. MAE calculation is based on the median of the error.
- C. RMSE is more sensitive to outliers whereas MAE is more robust to outliers.
- D. MAE refers to Mean Absolute Error, which is

$$\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

This gives less weight to outliers, which is **not sensitive to outliers**.

- **MAPE** refers to Mean Absolute Percentage Error, which is

$$\frac{100}{n} \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{y_i}$$

Similar to MAE, but normalized by true observation. Downside is when true obs is zero, this metric will be problematic

- **MSE** refers to Mean Squared Error, which is

$$\frac{1}{n} \sum_i^n (y_i - \hat{y}_i)^2$$

MSE is like a combination measurement of bias and variance of your prediction, i.e., **MSE = Bias² + Variance**, which is also most popular one – IT IS ALSO THE COST FUNCTION OF LINEAR REGRESSION

- **RMSE** refers to Root MSE, usually take a root of MSE would **bring the unit back to actual unit, easy to interpret your model accuracy**.

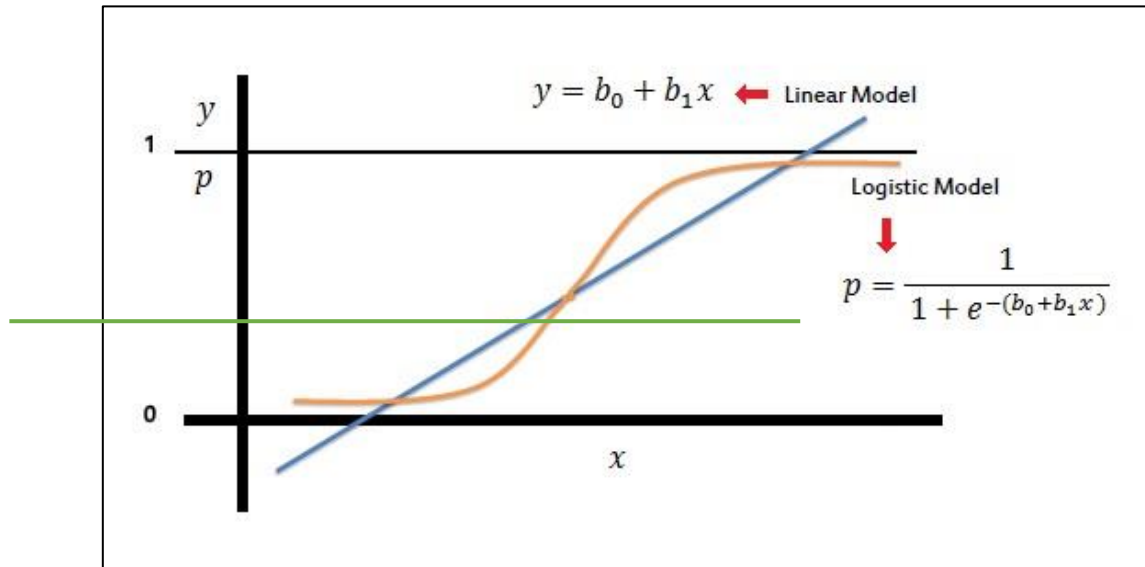
Snippet of Algorithm Code:

```
Model = LinearRegression()  
Model.fit(X_train, y_train)  
Y_pred = Model.predict(X_test)  
Model.score(y_test, Y_pred)
```

Logistic Regression Model

Definition: This Classification model is used when the target variable is binary in nature (eg: Yes or No). **The hypothesis of the logistic function is to limit the cost function (sigmoid function) between 0 and 1.**

Model: The sigmoid curve



Here, the Orange line denotes the decision boundary: if the y value is above this threshold value, then it is classified as 1. If y value is below threshold value then it is classified as 0.

The logistic regression model is obtained when we equate the log odds ratio and the logit(x) function and then simplify.

Odds = $p(\text{event occurring}) / p(\text{event not occurring})$

$$\Rightarrow P/1-P$$

Odds ratio = $\text{Odds}(1) / \text{Odds}(2)$

Taking the log of this ratio gives us **log odds ratio**.

The **logit(x)** function is similar to the linear regression model ($mX+c$) – the name logistic comes from this function.

Assumptions:

1. The dependent variable should be **dichotomous/binary** in nature (e.g., presence vs. absent).
2. There should be **no outliers** in the data, which can be assessed by converting the continuous predictors to standardized scores, and removing values below -3.29 or greater than 3.29.
3. There should be **no high correlations (multicollinearity) among the predictors. (as it reduces our models efficiency)**. This can be assessed by a correlation matrix among the predictors. Tabachnick and Fidell (2013) suggest that as long correlation coefficients among independent variables are less than 0.90 the assumption is met.

Advantages:

- Logistic regression is less inclined to over-fitting but it can overfit in high dimensional datasets. One may consider Regularization (L1 and L2) techniques to avoid over-fitting in these scenarios.
- Logistic regression is easier to implement, interpret, and very efficient to train.

Disadvantages:

- The major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables.
- Non-linear problems cannot be solved with logistic regression because it has a linear decision surface.
- Linearly separable data is rarely found in real life.
- It is tough to obtain complex relationships using logistic regression.
- More powerful and compact algorithms such as Neural Networks can easily outperform this algorithm.

Model Evaluation:

1. **Confusion Matrix:** a table that is used to define the performance of a classification algorithm. It compares the actual target values (y_{test}) with the predictions (y_{pred})

TP: Positive cases that are correctly predicted as positive

TN: Negative cases that are correctly predicted as negative

FP (Type I error): Negative cases that are wrongly predicted as positive.

FN (Type II error): Positive cases that are wrongly predicted as negative.

| | | Actual Values | |
|------------------|--------------|---------------|--------------|
| | | Positive (1) | Negative (0) |
| Predicted Values | Positive (1) | TP | FP |
| | Negative (0) | FN | TN |

2. **Accuracy:** $\frac{TP+TN}{TP+TN+FP+FN}$

Out of **all the cases**, how many are correctly predicted (as positive or negative)

3. **Precision:** $\frac{TP}{TP+FP}$

Out of **all the positive predictions**, how many were correctly predicted to be positive?

4. **Recall (Sensitivity/True Positive Rate):** $\frac{TP}{TP+FN}$

Out of **all the actual positive cases**, how many are correctly predicted to be positive?

Note: Precision vs Recall trade-off

-Case when we need high recall, and precision doesn't matter as much: Shop security camera to detect thieves. It would be ok to detect non-thieves also (low precision). But in case of actual thieves our sensitivity should be high and able to detect max. thieves. -Case when we need high precision, and recall doesn't matter as much: Child protection web surfing to prevent children from viewing inappropriate websites.

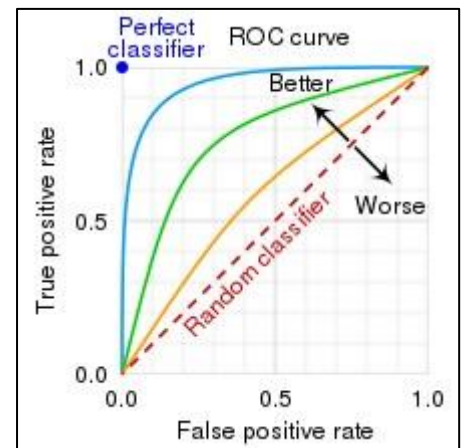
5. **F1 Score:** The harmonic mean between precision and recall. We prefer this over accuracy score when there **is class imbalance in the target column**. (males=99%, females=1% of total data). It finds out both sides of false positives=> higher means more false positive predictions.

6. Roc – Auc Curve (Receiver Operating Characteristic – Area Under the Curve):

The ROC curve is a method of evaluating a **Binary Classification** model where we plot the TPR against the FPR. The higher the AUC, the more **accurately the model can distinguish between positive and negative classes**. Ideally, we want the curve to reach the top left portion of the graph. However, any model above the random classifier threshold line (usually 0.5), is considered better than one below this line.

$$\text{TPR} = \text{TP} / \text{Actual positives} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{FPR} = \text{FP} / \text{Actual negatives} = \text{FP} / (\text{FP} + \text{TN})$$



Gaussian Naive Bayes Model

Definition: This is a supervised learning model used to solve classification problems. It is based on Bayes theorem – way to figure out conditional probability. Gives you actual probability of an event, given the information about the tests. This is a **probabilistic and generative model**.

$$\text{NB} = \text{prior} * \text{posterior} / \text{likelihood} = P(B|A)$$

$$P(B|A) = P(B) * P(A|B) / P(A)$$

Assumptions:

- Assumes that features are normally distributed (gaussian)
- Assumes that **the features are independent of each other (same as linear regression)**

Advantages:

1. Doesn't require feature scaling
2. Popular for text-based problems
3. Easy and fast
4. Used for both multiclass and binary classification
5. Better for **multiclass** than other models

Disadvantages:

1. Assumes that all features are **uncorrelated**, so it can't learn the relationship between features.

Model Evaluation:

1. Confusion matrix
2. Accuracy score

(Visualisation using meshgrid)

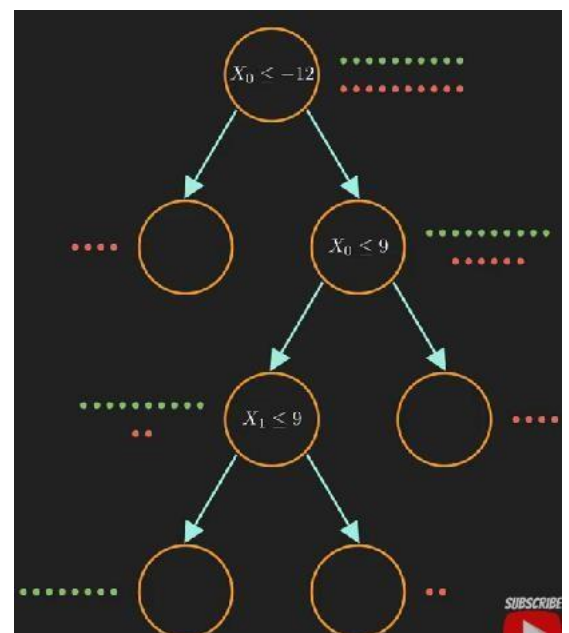
Snippet of Algorithm Code:

```
from sklearn.naive_bayes import GaussianNB #
Fitting Naive Bayes to the Training set classifier =
GaussianNB() classifier.fit(x_train, y_train)
Evaluate the Model
# Predicting the Test set results y_pred =
classifier.predict(x_test) # Making the
Confusion Matrix cm =
confusion_matrix(y_test, y_pred); cm
accuracy_score(y_test, y_pred)
```

Decision Trees Model

Definition: Is a discriminative model that is a flowchart and structure in which **the internal nodes** represent a test on an attribute, each **branch** represents the outcome of the test and each **leaf node** (decision node) represents a class label.

- It recursively splits a dataset until we are left with a pure leaf node (node with only one type of class of the target feature). At each node, there are certain conditions for features that further split into 2 child nodes (if condition is True and if condition is False. This continues till we arrive at a leaf node (child that is not a parent).
- Our model needs to learn which features to take, and the corresponding correct threshold values to optimally split the data.
- It uses **Top-down induction** learning by selecting the best attribute at each node.
- The quality of an attribute feature can be measured (EG: using statistical concepts like Gini Index)



- The **CART** (Classification and Regression Tree) algorithm is a type of classification algorithm that is required to build a decision tree on the basis of Gini's impurity index.

Decision Tree Is A Greedy Algorithm: It selects the current best split that maximises information gain. It doesn't go back and change previous splits, so any progressive split is dependent on the previous ones. This makes our algorithm very fast!

How does a decision tree decide which split is better?

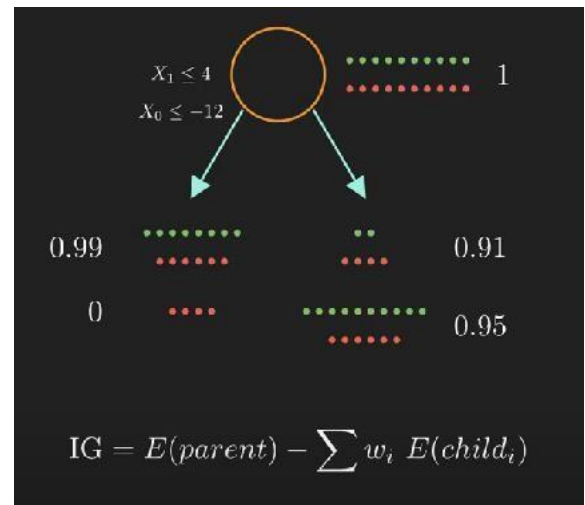
It chooses that split that maximises Information Gain (IG).

The **goal** is to reduce the impurity: Impurity is based on the proportion of values present from each class at a particular node. eg: if we have 50 males and 50 females at the root node, this means a 50:50 chance of selecting a male/female: The impurity is the highest. Impurity is calculated using Entropy:

Entropy = $\text{SUMMATION} - (P_i) \cdot \log(P_i)$

Where P_i = probability of class i

Basically, it compares the entropy of the parent node, to the average of the child nodes splits (T/F) and chooses that split which gives us **the maximum information gain (or decrease in entropy)**



How It Works:

Our model compares every possible split and takes the one that maximises information gain.

Advantages:

1. Easy to interpret and visualise
2. Can easily capture non-linear patterns: useful when a line cant
3. Requires fewer data pre-processing from the user (**No need to do feature scaling**)
4. Can be used for feature engineering (Like predicting missing values)

Disadvantages:

1. Sensitive to noisy data (can lead to overfitting)
2. Results in different decision trees if small variation in data (this can be reduced by bagging and boosting though)
3. Biased with an imbalanced dataset – recommended to balance out the dataset before creating the decision tree.

K- Nearest Neighbour Model

Definition: KNN is a **supervised** ML algorithm based on feature **similarities** and is mostly used for classification (but also regression). It classifies a data point based on how its neighbours are classified OR the distance from its nearest neighbours. KNN takes the training data, and then classifies new unseen data based on a similarity measure. **Q. What is K?**

A. K is a hyperparameter that refers to the number of nearest neighbours to include in the voting process. Choosing the right value of k is a process called parameter tuning and is VERY important for better accuracy.

Q. How to choose the value of k?

A. The initial value of k is determined by the square root of (number of records/rows). We prefer selecting an odd k value to avoid a situation where we get a 50:50 chance between two classes.

We use KNN when:

1. Data is labelled
2. Data is noise free (class is cleaned)
3. Dataset is small and not so complicated

Assumption:

Similar datapoints lie closer in spatial coordinates

How does it work?


- **Euclidian Distance** – squared sum of distance

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- **Manhattan Distance** – Sum of absolute distances

$$\text{Distance}(p, q) = \text{Distance} = |\text{SUM}(|p_i - q_i|)|$$

| Weight(x2) | Height(y2) | Class | Euclidean Distance |
|------------|------------|-------------|--------------------|
| 51 | 167 | Underweight | 6.7 |
| 62 | 182 | Normal | 13 |
| 69 | 176 | Normal | 13.4 |
| 64 | 173 | Normal | 7.6 |
| 65 | 172 | Normal | 8.2 |
| 56 | 174 | Underweight | 4.1 |
| 58 | 169 | Normal | 1.4 |
| 57 | 173 | Normal | 3 |
| 55 | 170 | Normal | 2 |



| | | |
|-------|--------|---|
| 57 kg | 170 cm | ? |
|-------|--------|---|

Basic Algorithm code:

From sklearn.neighbors import **KNeighborsClassifier** OR **KNeighborsRegressor**

-standardise data

-split the dataset using train_test_split

-import math

X = math.sqrt(len(y_test)) : to get starting k value, minus 1 to get odd if its even

-classifier = KNeighborsClassifier((n_neighbors= x, metric = 'euclidean')

Model Evaluation:

F1 score, accuracy score, confusion matrix

Hyperparameter Tuning: Error Rate

For example

k_vals = [5, 15, 25, 35, 45, 55, 65, 75, 85, 95]

```
error_rate=[]
for i in k_vals:
    clf1 = neighbors.KNeighborsClassifier(n_neighbors=i)
    clf.fit(X_train,y_train)
    pred_i = clf.predict(X_test)
    error_rate.append(np.mean(pred_i !=y_test))
    print(f'\033[1;3mFor i = {i}\033[0m')
    print('Accuracy score:', accuracy_score(pred_i, y_test))
    print('Error rate is:', np.mean(pred_i !=y_test))
    print('----')
#all error rates and accuracy scores are same
```

```
For i = 5
Accuracy score: 0.8007068223724647
Error rate is: 0.19929317762753535
----
For i = 15
Accuracy score: 0.8007068223724647
Error rate is: 0.19929317762753535
----
For i = 25
Accuracy score: 0.8007068223724647
Error rate is: 0.19929317762753535
----
For i = 35
Accuracy score: 0.8007068223724647
Error rate is: 0.19929317762753535
----
For i = 45
Accuracy score: 0.8007068223724647
Error rate is: 0.19929317762753535
----
For i = 55
Accuracy score: 0.8007068223724647
Error rate is: 0.19929317762753535
----
For i = 65
Accuracy score: 0.8007068223724647
Error rate is: 0.19929317762753535
----
For i = 75
Accuracy score: 0.8007068223724647
Error rate is: 0.19929317762753535
----
For i = 85
Accuracy score: 0.8007068223724647
Error rate is: 0.19929317762753535
----
For i = 95
Accuracy score: 0.8007068223724647
Error rate is: 0.19929317762753535
----
```

Use the resulting error rate and k value lists to make a line plot: **choose the k with the lowest error rate**

Cross Validation

It is important to know that our model accuracy remains consistent throughout. For example: If our model performs well on one particular train-test split, would it perform equally well on a different kind of split? – Back to the most important question in ML: is our model stable?

Two main reasons for instability are **overfitting (high variance)** and **underfitting (low train score, bad fit)**

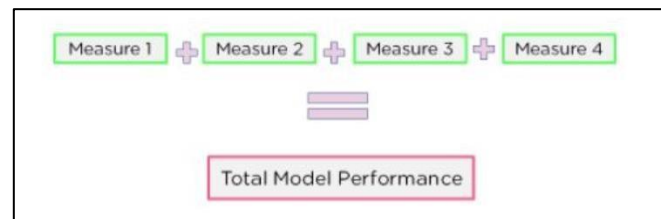
Definition: Cross Validation is a technique used to train and evaluate our model on one portion of data, then re-portioning our dataset and evaluating it on new portions. So instead of splitting data into 2 parts of train and test, we split it into many parts – some to train and some to test.

Steps:

Step 1: Split the data into train and test sets and evaluate the model's performance

Step 2: Split the data into new train and test sets and re-evaluate the model's performance. Repeat until the model has been trained and evaluated on the entire dataset

Step 3: To get the actual performance metric the average of all measures is taken



Purpose of Cross Validation:

- Exposes our model to minority classes present in the data (this might reduce bias)
- Tells us how the model will generalise to unseen data
- Helps to select variables to include in our model (in X and y)
- Used to detect overfitting or bias

Type of Cross Validation:

1. **Hold Out Method** – splitting the data into 2 parts: training and testing. The proportion depends on the size of the dataset, but 80:20 is used in most cases.
2. **Leave One Out** – Not used because it is too computationally expensive. Also subject to high variance. Here, the number of folds equals the number of instances in the data set. The learning algorithm of our choice, is applied once for each instance: all other instances are used as a training set, and the selected instance makes up the one-item test set (Consider a dataset with N points. N-1 will be the training set and 1 point will be the testing set). Used only when you have a small dataset or when an accurate estimate of model performance is more important than the computational cost of the method.

3. **K Folds** - Here we split the data into k folds (or k splits) and use k-1 parts in training, and the remaining single portion for testing. This process is repeated till each portion has been a test set. Each step also validates the test set. In the end, the score is provided as an average of all test scores. This makes the model more stable. Used in **Regression problems** only, NOT classification problems. Uses **random sampling**.

CONS

- : cost of computation can be high (if k=1000, model runs 1000 times)
- : All classes are mixed (class imbalance can make this method less effective)

4. **Stratified K Fold** – useful when there are minority classes present in our data. Here, data is split in a similar way to K Folds, but it is split in such a way that each split represents all the classes proportionally, from the population. It uses **stratified sampling** rather than random sampling. It is used **for classification problems** and helps **avoid class imbalance and thus, lowers bias**.

Higher the k, lower the variance and higher the accuracy: until a threshold is reached beyond which accuracy score stabilises.

5. **Repeated K Fold:**

Repeated k-fold cross-validation provides a way to improve the estimated performance of a machine learning model. This involves simply repeating the cross-validation procedure multiple times and reporting the mean result across all folds from all runs.

6. **Repeated Stratified K Fold** – `cv = RepeatedStratifiedKFold(n_splits=7, n_repeats=3, random_state=2)` Similar to repeated k fold, but it is used for classification problems

Hyperparameter Tuning

Definition: Hyperparameter tuning (or hyperparameter optimization) is the process of determining the right combination of hyperparameters that maximizes the model performance.

Why? To reach to the somewhat highest performance of a model, you need to try different hyperparameters

When? whenever you find an "appropriate" model for your task or made a architecture of a model (e.g. in artificial neural networks) then you need to tune hyperparameters to make sure that the model could make good enough predictions. It is when you **could not improve** the performance of the model **by changing the model architecture**.

There are different approaches for tuning of hyperparameters such as **grid search** and **random search** that you could choose based on your preferences. The point is that the "appropriate" model may differ from one application to another.

Parameters v Hyperparameters:

| PARAMETERS | HYPERPARAMETERS |
|---|--|
| They are required for making predictions | They are required for estimating the model parameters |
| They are estimated by optimization algorithms(Gradient Descent, Adam, Adagrad) | They are estimated by hyperparameter tuning |
| They are not set manually | They are set manually |
| The final parameters found after training will decide how the model will perform on unseen data | The choice of hyperparameters decide how efficient the training is. In gradient descent the learning rate decide how efficient and accurate the optimization process is in estimating the parameters |

How is it done?

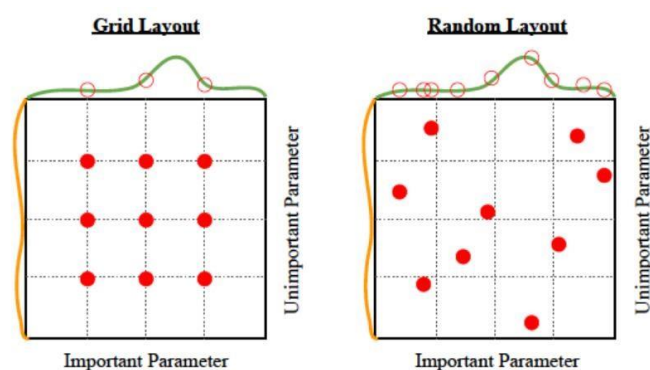
It works by running multiple trials in a single training process. Each trial is a complete execution of your training application with values for your chosen hyperparameters, set within the limits you specify. This process once finished will give you the set of hyperparameter values that are best suited for the model to give optimal results.

○ GridsearchCV

In the grid search method, we create a grid of possible values for hyperparameters. Each iteration tries a combination of hyperparameters in a specific order. It fits the model on each and every combination of hyperparameters possible and records the model performance. Finally, it returns the best model with the best hyperparameters.

○ RandomizedSeachCV

In the random search method, we create a grid of possible values for hyperparameters. Each iteration tries a random combination of hyperparameters from this grid, records the performance, and lastly returns the combination of hyperparameters that provided the best performance.



Ensemble Techniques

Lowers Bias, lowers Variance, and Increases Accuracy

Ensemble learning is about combining some base models in order to obtain an ensemble model with better performances/properties.

Technique that creates multiple models – homogenous or heterogenous- and combines them for the best result. Usually, it produces **more accurate results** than a single model would. Majority voting is used for classifiers and central tendencies for regressors.

Hard Voting: concerned with the class that wins the majority vote

Soft Voting: concerned with the highest average probability

The main **purpose** of using an ensemble model is **to form a strong learner from a group of weak learners**.

1. Bagging using Random Forest: Use when we have high variance

Bootstrapping is a statistical method of creating samples using the **replacement method**.

Bagging, also known as Bootstrap aggregating, is an ensemble learning technique that helps reduce variance and increase accuracy of machine learning algorithms.

Bagging is a homogeneous weak learners' model that learns from each other independently in parallel and combines them for determining the model average. Uses **deep decision trees**.

Snippet code for creating Random Forest Classifier:

| |
|---|
| from sklearn.ensemble import RandomForestClassifier |
| # define the model |
| model = RandomForestClassifier() |
| # evaluate the model |
| cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1) |
| n_scores = cross_val_score(model, X, binary_encoded_y, scoring='f1', cv=cv, n_jobs=-1, error_score='raise') |
| # report performance |
| print('F1-Score: %.3f (%.3f)' % (mean(n_scores), std(n_scores))) |

2. Boosting using Adaboost: Boosting is also a homogeneous weak learners' model but works differently from Bagging. In this model, learners **learn sequentially** and adaptively to improve model predictions of a learning algorithm. Each model is dependent on another.

The term 'Boosting' in a layman language, refers to algorithms that convert a weak learner to a stronger one. It decreases the bias error and builds strong predictive models.

3. **Stacking:** Stacking is an ensemble learning technique that uses predictions from multiple models (for example decision tree, knn or svm) to build a new model. This model is used for making predictions on the test set.

Revision: Use of techniques

Cross validation: used at the time of splitting data: before fitting the model

Ensemble techniques: used at the time of model creation (fitting the model)

Feature selection: Used to select independent features at the time of data preprocessing

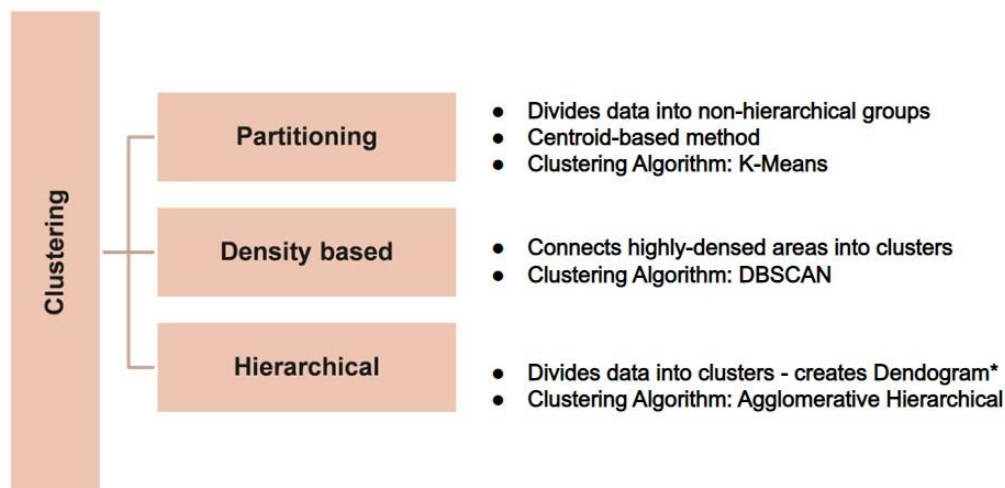
Hyperparameter tuning: Done after model evaluation/selection? to improve our models performance to the best possible performance.

So first we do cross validation to reduce variance and bias, then ensemble techniques to boost accuracy

Unsupervised Learning - Clustering

(Data is unlabelled – we have input features X, but no labels y)

Types of Clustering Techniques



K Means Clustering

Group similar objects into clusters.

- **Objective:** learn what normal data looks like, then identify similar instances and assign them to clusters.

- **Applications:** customer segmentation, recommender systems, search engines, image segmentation, data analysis.
- K-Means clustering looks for datapoints centred around a particular point called the **Centroid** ○ **Scaling** is important for KMeans: One particular column with large values will have more impact than other smaller ones
- **Important calculations in kmeans:** centroids, distance metric, number of clusters ○ **Any extreme values** will have an impact on the performance of k means

How does it work: Randomly selects centroids, then it checks the distance of all data points from each of the centroids. The intra distance (within the cluster should be close) should be less than the inter distance of clusters (between clusters, distance should be large)

Assumption: Assumes the spherical shape of clusters

Snippet of Code to create K Means Model:

From sklearn.cluster import KMeans

K=5 cluster_model =

KMeans(n_clusters=k)

Y_pred = cluster_model.fit_predict(X)

Note: We have to specify the number of clusters k that the algorithm must find.

- KMeans algorithm then creates a copy of the labels of each instance that it was trained on (basically the label of the cluster numbers eg:0,1,2,3,4). You can access it using:

```
cluster_model.labels_
```

- We can also access the 5 centroids that the algorithm found:

```
cluster_model.cluster_centers_
```

- Measure the distance of each point to every centroid:

```
cluster_model.transform(X)
```

Advantages and Disadvantages of K Means:

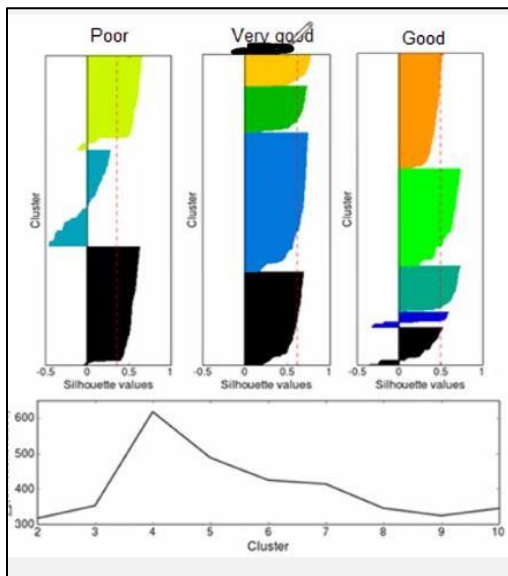
| Pros | Cons |
|---|--|
| Very fast and scales well | Sensitive to outliers |
| New points can be assigned to a cluster without training a new model | Number of clusters must be known in advance |
| Cluster assignments can be used as additional features for other models | Limited cluster shapes: All clusters tend to be spherical with the same radius |
| General purpose | Result depends on the random start points → Runs it multiple times and chooses the best one |
| Easy to understand | Tends to produce clusters of the same size |

Additional Cons:

- Cannot handle uneven cluster sizes: all clusters are approximately the same size and shape.
- **Clustering outliers:** the centroids can be dragged by outliers or these outliers might even get their own cluster.

Important Metrics in K Means Clustering:

- **Inertia:** The mean squared error of the distance between each data point, and its cluster centroid. **Code:** `model.inertia_`
- **Elbow Score (Measures the errors):** Can finalise the optimal number of clusters **at the elbow of the curve.**
- **Silhouette Score** = $(b - a) / \max(a, b)$: Indicates goodness of model.
 - Where b = inter cluster distance, and a = intra cluster distance
 - Tells us about the quality of the cluster model (PERFORMANCE METRIC)
 - (Just like r2 and accuracy in regression and classification problems)
 - Ranges from -1 to 1
 - Score Of 1: Means clusters are well apart from each other and clearly distinguished.
 - Score Of 0: Means distance between clusters is not significant,
 - Score Of -1: Means clusters are assigned in the wrong way



- Used to calculate the goodness of a clustering technique
- Also used to find the optimum value for k during K-Means clustering. Its value ranges between -1 and +1.
- A negative silhouette value indicates that a point would fit better into another cluster
- The greater the silhouette value, the better the clustering

Snippet of Algorithm code:

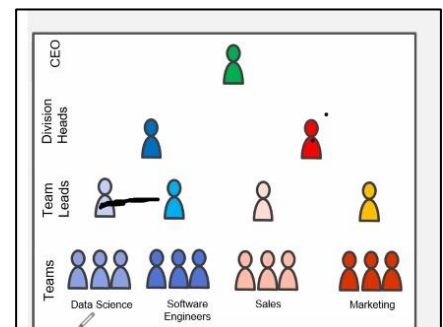
From sklearn.metrics import silhouette_score

Silhouette_score(df, model.labels_)

Hierarchical Clustering

Definition: method of unsupervised clustering that aims to build a hierarchy of clusters.

Here, how would you begin clustering – if the data is ordinal?

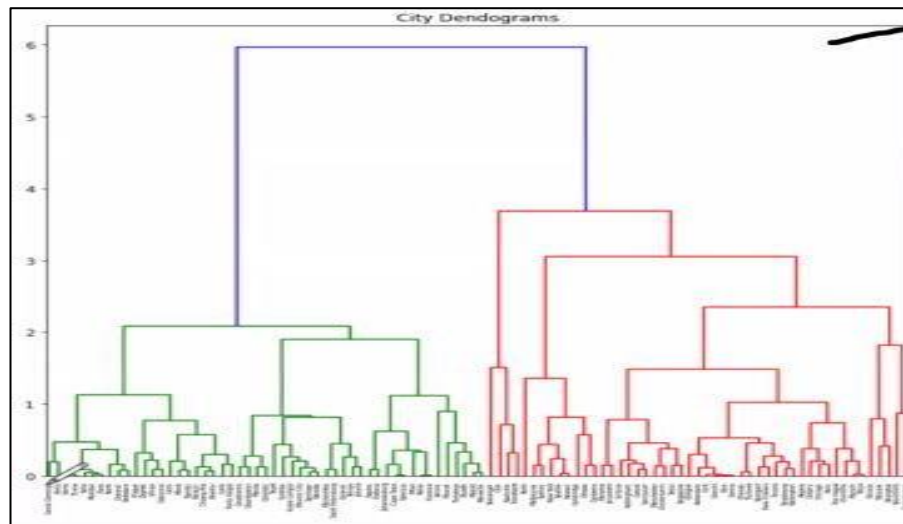


Main drawback of KMeans that Hierarchical clustering solves: K-Means tries to create clusters of ~same size. But hierarchical clustering does not do this. It is also computationally faster than KMeans.

Dendrogram: inverted tree-like diagram that is a graphical representation of the hierarchy of clusters. It that explains the relationship between all the data points in the system.

The sole concept of hierarchical clustering lies in just the construction and analysis of a dendrogram.

Dendrogram Diagram



How It Works:

1. Machine takes each data point as a separate cluster (8 datapoints => 8 clusters). Each at the lowest layer in this graph
2. Identify the two clusters that are closest to each other (using Euclidean distance)
3. Merge the two most similar clusters
4. This process repeats until all clusters are merged and form only one single cluster (top layer of the graph)

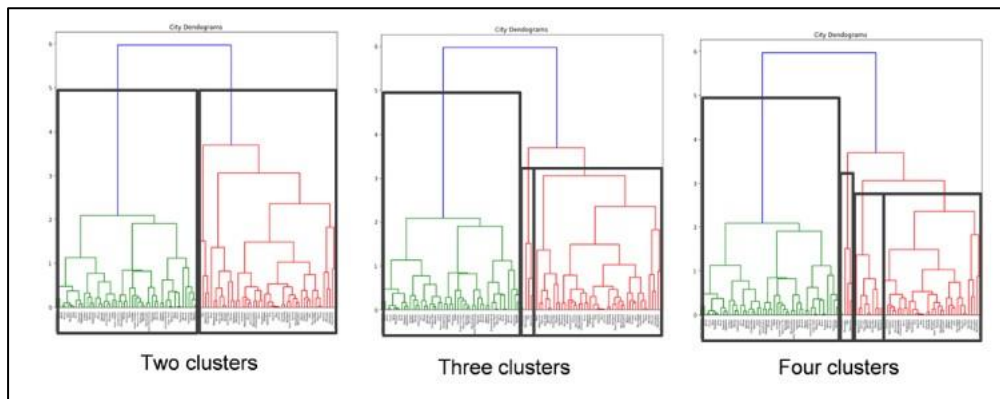
TYPES OF HIERARCHICAL CLUSTERS

⑨ Agglomerative clustering (from bottom layer to top layer)

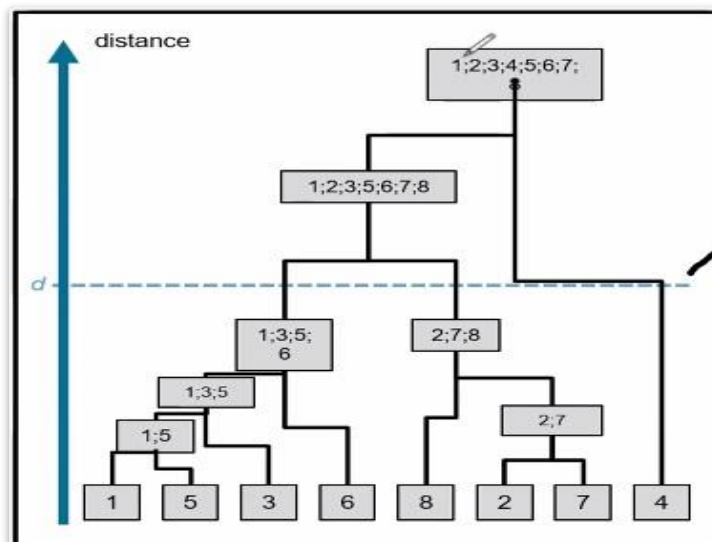
- Bottom up
- N points cluster at the start
- Merges cluster

⑨ Divisive clustering (from top layer to bottom layer)

- Not that popular because of the kind of clusters it creates (all datapoints become separate clusters)
- Top down
- One cluster at the start
- Splits cluster



- Clusters can be obtained by cutting the tree at the selected height d (merge effort)
-



Advantages and Disadvantages:

| Pros | Cons |
|---|--|
| It can handle many clusters well | It doesn't work well with complex shapes |
| Flexibility in choice of cluster metric | New points cannot be assigned to a cluster without recalculating the model |
| Works well with uneven cluster sizes | Number of clusters must be defined |
| Possible connectivity constraints | Complex analysis of the result |
| Good overview of the resulting clusters based on the threshold distance | Single-link effect ("chains") depending on the metric |
| <u>Able to detect outliers</u> | |

Main cons: doesn't work well with larger datasets, sensitive to outliers, computationally demanding **Linkages:** At the time of computing the distance between datapoints in order to determine whether to merge clusters, we use different linkage methods. Linkage specifies how the distance between two clusters is measured

- **Average:** Linking 2 clusters based on the average distance between all pairs of elements
- **Complete:** linking 2 clusters based on the MAXIMUM distance between 2 elements
- **Single:** linking two clusters based on the MINIMUM distance of two elements
- **Ward:** Instead of

distance, it measures the variance of clusters and tries to MINIMISE variance. It calculates the distance between two clusters as the SSE (sum of squared errors) increases.

DBSCAN Clustering

(Density Based Spatial Clustering of Application with Noise)

Searches for continuous areas of high intensity i.e. clusters separated by areas with low density

- Select this based on: scatterplot, the density of clusters should be same
- User has to select hyperparameters of **epsilon** and **min_samples**
- DBSCAN algorithm figures out: core, border and noise points.

min_samples: minimum number of datapoints within the distance (eps) in order to declare a point as a CORE point
eps (Epsilon): radial distance from a centric data point (RADIUS LENGTH) (centroid is a single point)

-For each datapoint, the algorithm counts how many instances are located within a small distance (epsilon) from it.

- **DENSITY REACHABILITY:** It establishes a point to be reachable from another if it lies within a distance (or epsilon) from it
- **CONNECTIVITY:** It involves a transitivity-based chaining approach to determine whether points are located in a particular cluster.

Instance when DBSCAN is the more appropriate model than K Means: In a nested circles graph, Kmeans can never do justice, but dbscan would be perfect: as it groups based on density.

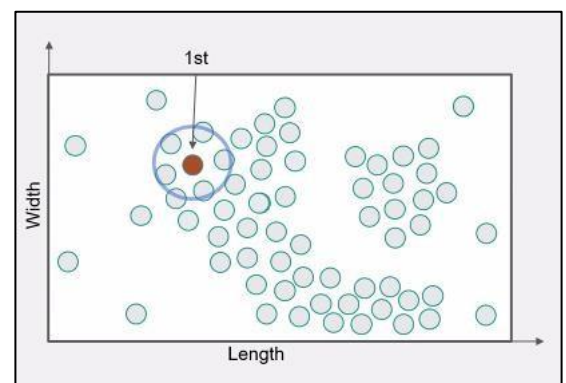
Assumption: If density is not suitable, it may not do correct sampling

Properties Of DBSCAN:

- Finds areas of higher density than surroundings
- Finds arbitrarily shaped clusters (different shapes)
- Slower than k means, but still fast

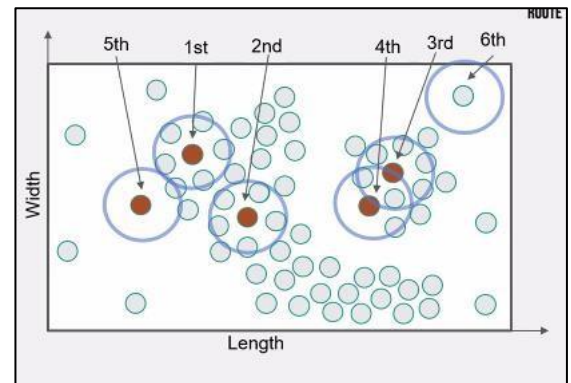
How it works:

1. Starts randomly with one point, the brown one
2. Draw a blue circle around it
3. The blue circle overlaps, at least partially, close to 7 points (including brown).
4. We decide the radius of the circle! Best radius value is finalised using hyperparameter tuning



Let's now take the second brown point, draw a blue circle and see that at least 9 points overlap it (including brown).

- Similarly, for all remaining points, we count the number of points



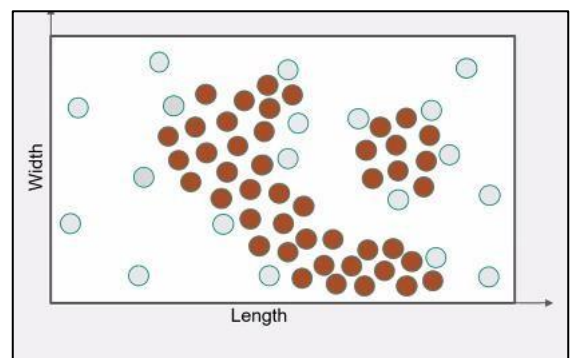
TYPES OF POINTS

- **CORE POINT:** has high density circle
- **BORDER POINT:** its circle overlaps the minimum number of datapoints
- **NOISE POINT:** no points, like outliers

Types of Points

- **Core:** This is a point at least min_samples points within distance eps from itself.
- **Border:** This is a point that has at least one core point at distance eps .
- **Noise:** This is a point that is neither a core nor a border, and it has less than min_samples points within distance eps from itself.

- All these brown ones are core points
- The gap between them is like a river flowing between 2 villages and it's an area of low density: separating areas of high density into different clusters.
- The cluster size increases by joining contiguous (neighbouring) Core points
- If core points are near another cluster with a core point: we increase the size of that



Think: how political campaigns happen in an area: as they gather more supporters, their cluster size increases – radius keeps increasing

- We cannot add noise points to a cluster (they are excluded by DBSCAN).
- But non-core points (border points) are ultimately added.
- Border points keep changing based on the minimum points we consider for a datapoint to be a CORE point!

| Pros | Cons |
|---|---|
| Number of clusters doesn't have to be known | Clusters can't have different densities |
| Arbitrarily-shaped clusters | Hard parameter tuning (especially for epsilon-neighborhood distance) |
| Assumes existence of noise | Cluster assignment for border points (which are in the range of multiple clusters) depends on execution order |
| Can be used for outlier detection | |
| Outlier resistant | |
| Works well with uneven cluster sizes | |

Major Advantages:

- **It handles outliers very well by itself**
- Works well with uneven cluster sizes (kmeans cant do this)

Main drawbacks:

- Cluster assignment is highly dependent on the first point we assign as core point.
- Its highly sensitive to parameters (min_samples, epsilon) – changes the clusters
- If density varies a lot between clusters, it is impossible to capture all clusters properly.

Bonus: Feature Extraction (Pre-processing step)

- If there is very large data with numerous features, it can be reduced to fewer dimensions in order for an algorithm to run smoother and to make visualisations possible.
- **CURSE OF DIMENSIONALITY:** Although we like more information in our dataset, increasing the dimensions (attributes/columns) in the plots makes data presentation more complex. Even adding one more dimension to our model, can slow down our model/algorithm (think: cross validation splitting). This increases space and computational time complexity. Interdependency (correlation) among attributes is also one cause of the algorithms (especially Naïve bayes).

Techniques Of Dimensionality Reduction:

- PCA : Principal Component Analysis
- LDA : Linear Discriminant Analysis

Here, we can't visualise all 5, we want to reduce it to 2-3 features.

DIMENSIONALITY REDUCTION – reducing number of columns and resolves curse of dimensionality

The process of reducing the number of random variables under consideration, by obtaining a set of principal components or variables ensuring that they provide similar information

| Customer ID | Gender | Age | Annual Income (k\$) | Spending Score (1-100) |
|-------------|--------|-----|---------------------|------------------------|
| 1 | Male | 19 | 15 | 39 |
| 2 | Male | 21 | 15 | 81 |
| 3 | Female | 20 | 16 | 6 |
| 4 | Female | 23 | 16 | 77 |
| 5 | Female | 31 | 17 | 40 |
| 6 | Female | 22 | 17 | 76 |
| 7 | Female | 35 | 18 | 6 |
| 8 | Female | 23 | 18 | 94 |
| 9 | Male | 64 | 19 | 3 |
| 10 | Female | 30 | 19 | 72 |
| ... | ... | ... | ... | ... |

Benefits:

1. Takes less space to store the dataset (faster computation)
2. Requires less computation training time
3. Enhances visualisation
4. Removes the redundant features by taking care of multicollinearity

Principal Component Analysis (PCA)

- The linear method
- Unsupervised learning
- The variance of data is maximum in lower-dimensional space

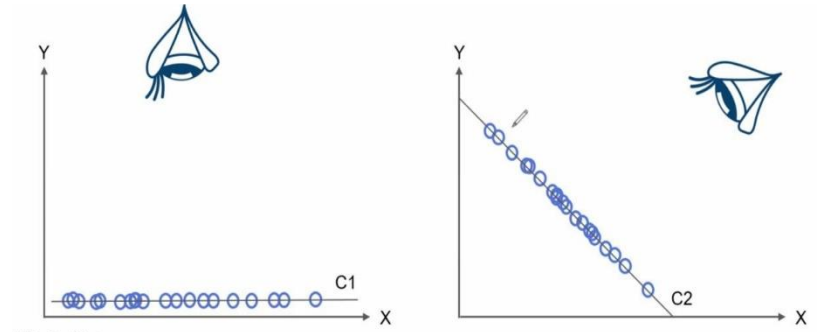
Linear Discriminant Analysis (LDA)

- The linear method
- Supervised learning
- Creates decision boundaries and maximizes the separation between multiple classes

A. PCA – Principal component analysis: checks for high variance of features

- Converts data from a high dimensional space to a low dimensional space by selecting the most important attributes that capture maximum information about the dataset.
- PCA is based on orthogonal linear transformation – a mathematical technique to project the features to a new axis. The attribute that describes the most variance is the 1st PC, and then the attribute with the second highest variance is 2nd PC, So on. Until the entire dataset can be expressed in terms of PC's.
- Transforms the data into a set of values that consists of uncorrelated variables (because for learning, the independent columns should be uncorrelated (if col2 and 3 are +vely correlated, then their increase or decrease would impact the target column in the same way!))
- Features that have maximum variance are selected. The first component has the largest possible variance. **Each succeeding component has the highest possible variance under the condition that it is orthogonal to the existing component (independent of other columns).**

Orthogonality means “uncorrelated.” An orthogonal model means that all independent variables in that model are uncorrelated. If one or more independent variables are correlated, then that model is non-orthogonal.



- We have to tell the algorithm how many columns we want. Eventually, we land up with a visual of a linear line
- The number of producing PC's depends on the users requires but they must be less than the existing dimensions in the dataset
- We calculate eigen vectors and eigen values in feature extraction.

PCA Advantages and Disadvantages

| Pros | Cons |
|--|---|
| <ul style="list-style-type: none"> • Removes correlated features • Improves algorithm performance • Reduces overfitting • Improves visualization • Is simple and intuitive to use • Does not require hyperparameter tuning | <ul style="list-style-type: none"> • Makes independent variables less interpretable • Requires data standardization as a prerequisite • Involves information loss • Has high computational costs • Doesn't work well for fine-grained classes • Hard to model due to non-linear structure |

Note:

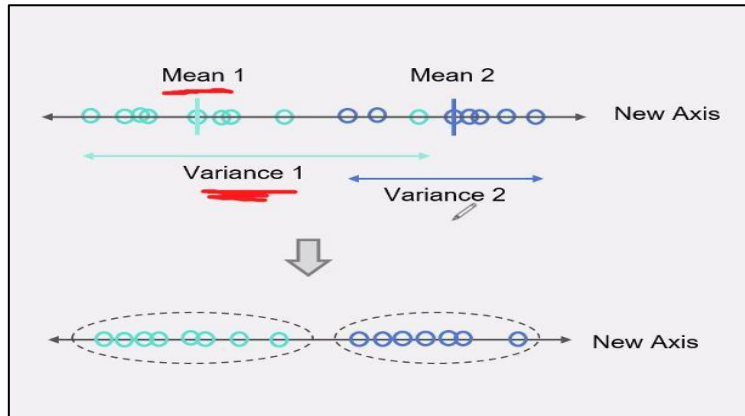
Linear structure: something related to a straight line

Non-linear structure: An equation that doesn't form a straight line (eg: logistic (sigmoid curve, log)

B. LINEAR DISCRIMINANT ANALYSIS: max distance of means, min variance

- Goal: Reduce dimension
- Process: Maximise separation between classes in target column ○ **Supervised learning thus, requires the class information as input** ○ **Average and variance are 2 important calculations:** We want the new axis to created so that the distance between the 2 means is **MAXIMUM**, and variation within each class in **MINIMUM (max mean, min variance)**

- Classes should be normally distributed!



(The two colors show different classes)

LDA Advantages and Disadvantages:

| Positive Aspects | Negative Aspects |
|---|--|
| <ul style="list-style-type: none"> • Simple, fast portable algorithm • Beats limitation of Logistic regression for multi-class classification | <ul style="list-style-type: none"> • Works only on normally distributed predictors/features • Sometimes not good for few categories features |

Difference Between PCA And LDA (Both Are Linear)

| PCA | LDA |
|--|--|
| <ul style="list-style-type: none"> • <u>Unsupervised</u> learning • Captures the direction of <u>maximum variation</u> in the data set • Does not make any assumption | <ul style="list-style-type: none"> • <u>Supervised</u> learning • Finds a feature subspace that maximizes the separability between the groups • Makes assumptions about <u>normally distributed classes</u> |