ML in Python.

Machine Learning Algorithms:-

Regression:
→ Simple linear regression
→ multiple linear regression
→ Regression trees

Classification:

→ Logistic regression
→ KNN
→ SVM
→ multiclass prediction
→ Decision trees.

Clustering
→ K-means.

Machine learning is the subfield of computer science that gives "computers the ability to learn without being explicitly programmed."

- Arthur Samuel.

Major Machine learning techniques:-
→ Regression/Estimation:
Predicting continuous values
Ex:- CO_2 emission from a car's engine

→ Classification
Predicting the item class/category of a case
Ex:- if a cell is benign or malignant.

→ Clustering
Finding the structure of data, summarization

Ex:- for customer segmentation in the banking field

→ Associations
Associating frequent co-occurring items/events

Ex:- grocery items that are usually bought together by a particular customer

→ Anomaly detection
Discovering abnormal and unusual cases
Ex:- for credit card fraud detection

→ Sequence mining
Predicting next events: click stream (Markov model, HMM)
Ex:- for predicting the next event, for instance the click stream in website.

→ Dimension Reduction
Reducing the size of data (PCA)

→ Recommendation systems
Recommending items
Ex:- movies or books recommendations on platforms

Python libraries for ML

↓

general purpose programming language.

→ Numpy → a math library to work with N-dimensional arrays in Python.

↳ Ex:- for working with arrays, dictionaries, functions, datatypes and working with images

→ SciPy → ~~collection~~ numerical algorithms & domain specific toolboxes.

→ Matplotlib → popular plotting package that provides 2D & 3D plotting.

→ Pandas

→ Scikit learn

Data preprocessing → Train / Test split → Algorithm setup → Model fitting

↓

Prediction → Evaluation → Model export

Supervised vs. Unsupervised

Supervised learning.

→ We teach the model with knowledge so that it can predict unknown or future instances.

We teach the model by training it with some data from a labelled dataset.

Two types of Supervised learning
→ Classification
→ Regression

Classification is the process of predicting a discrete class label or category

Regression is the process of predicting a continuous value as opposed in classification

Unsupervised learning: (unlabelled data)

The model works on its own to discover information.
It learns on datasets and draws conclusions on unlabelled data.
U.L techniques
→ Dimension reduction
→ Density Estimation
→ Market basket analysis
→ Clustering

Clustering

→ is grouping of data points or objects that are somehow similar by
- It's covering structure
- Summarisation
- Anomaly detection.

Regression:

There are 2 types of variables
→ dependent variable & independent variable

dependent variable
- State, target or final goal
- (Y) should be continuous value (not discrete)

independent variable
- Explanatory variable
- (X)

Simple regression:

one independent variable is used to estimate a dependent variable
→ simple linear or non linear regression
Ex:- predict CO_2 emission Vs. Engine size of all cars.

Multiple Regression

When more than one independent variable is present.
Ex:- predict CO_2 emission Vs. Engine size & cylinders of all cars.
multiple linear or non-linear regression.

Application of Regression

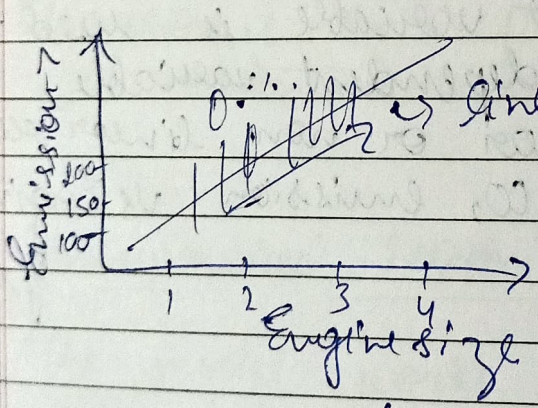
- sales forecasting
- satisfaction analysis
- real estimation
- employment income.

Simple Linear Regression

Linear regression is the ~~linear~~ approximation of a linear model used to describe the relationship b/w 2 or more variables.

In single linear regression → 2 variables a dependent & an independent variable

Dependent value should be continuous & can't be discrete value.



⇒ linearly related

$$Y = \theta_0 + \theta_1 X_1$$

→ slope or gradient of fitting line

- Response variable (predicted or dependent)
- intercept
- a single predictor (independent)

θ_0 & θ_1 are called the coefficients of linear eqn.

x_i = $x_1 = 5.4$

$y = 250$ (actual CO_2 emission)

$\hat{y} = \theta_0 + \theta_1 x_1$

$\hat{y} = 340$ (predicted emission)

Error $= y - \hat{y}$

$= -90$ (Residual error)

mean squared Error (MSE) $= \frac{1}{n} \sum^{n} (y_i - \hat{y}_i)^2$

The objective of linear regression, is to minimize the MSE eqn & we need to find the best (suitable) parameters θ_0 & θ_1

$$\theta_1 = \frac{\sum_{i=1}^{s} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{s} (x_i - \bar{x})^2}$$

Ex:- Engine size

$\bar{x} = (2.0 + 2.4 + \ldots)/19 = 3.03$

$\bar{y} = (196 + 221 + 136 + \ldots)/19 = 226.22$

CO_2 emission

$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$

$$\theta_1 = \frac{(2.0 - 3.03)(196 - 226.22) + (2.4 - 3.03)}{(2.0 - 3.03)^2 + (2.4 - 3.03)^2}$$

$$\theta_1 = 39.$$

$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$

$$\theta_0 = 226.22 - 39 * 3.03$$

$$\theta_0 = 125.74$$

The two parameters for the line.

θ_0 → bias coefficient.

$$y = 125.74 + 39 X_1$$

CO_2 Emmision = $\theta_0 + \theta_1$ Engine size

Pros of linear regression

→ very fast
→ no parameter tuning
→ easy to understand.

Model Evaluation in Regression models:

→ Train & Test on the same dataset
→ Train / Test split.

Ex:

$$\text{Error} = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|$$

Take the CO_2 emmission & train the model & then take few parameters to test along with their actual values (y), compare it with the predicted values ($\hat{y}$)

$$\text{Error} = \frac{(233 - 234) + (255 - 256) + \dots}{4}$$

$$\text{Error} = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|$$

1) Train & test on the same dataset
→ train the model on the entire dataset, then test it using a portion of the same dataset.
→ high training accuracy & low "out of sample" accuracy

(Test set is a portion of train set)

Training accuracy → percentage of correct prediction the model makes when using the test dataset
(High Tr Ac → results in overfit)

Out of sample accuracy → is the percentage of correct predictions that the model makes on data that the model has not been trained on.
Train & test on same data causes low "out of sample" accuracy → overfit.

Date
Page

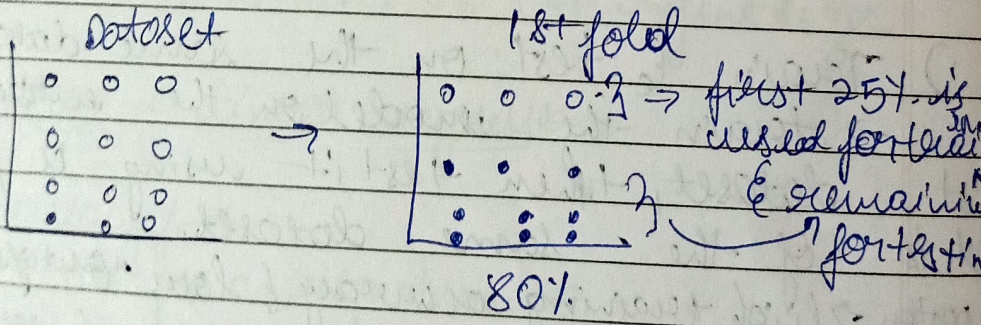
Over-fit → the model is overly trained to the dataset which may capture noise and perform a non-generalized model.

2) Train/Test split Evaluation approach.

→ involves splitting the dataset into training and testing sets (separately).

- it is mutually exclusive.
- more accurate evaluation on out-of-sample accuracy
- highly dependent on which datasets the data is trained & tested.

3) K-fold cross-validation.



Accuracy 84%.

3rd fold → 82%, 4th fold → 86%.

Accuracy: Avg →
→ 83%.

→ k-fold cross-validation performs multiple train-test splits using the same dataset where each split is different.

Then the result is averaged to produce more consistent out of sample accuracy.

Evaluation metrics in Regression models

Error in regression is the difference between the data points & the trend line generated by the algorithm.

Mean Absolute Error (MAE) = $\frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$

Mean Squared Error (MSE) = $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

Root MSE (RMSE) = $\sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$

Relative Absolute Error (RAE) = $\frac{\sum_{j=1}^{n}|y_j - \hat{y}_j|}{\sum_{j=1}^{n}|y_j - \bar{y}|}$

(or residual sum of squares)

→ mean value of y

Relative squared error (RSE) $= \frac{\sum_{j=1}^{n} (y_j - \hat{y}_j)^2}{\sum_{j=1}^{n} (y_j - \bar{y})^2}$

$R^2 = 1 - RSE$

↓

represents how close the data values are to the fitted regression line.

higher R^2 → better the model fits the data.

Simple Linear Regression:-

Importing packages:

```
import piplite
await piplite.install(['pandas'])
await piplite.install(['matplotlib'])
await piplite.install(['numpy'])
                      ['scikit-learn'])

import matplotlib.pyplot as plt
import pandas as pd
    "    pylab as pl
    "    numpy as np
% matplotlib inline }  //Enables drawing of
                      //mat plotlib figures in IPython environment
```

// To download data:

path = "

.csv"

deals with HTTP requests

from pyodide.http import pyfetch

fetch the url & return response

async def download(url, filename):

response = await pyfetch(url)

if response.status == 200:

with open(filename, "wb") as f:

f.write(await response.bytes())

// return response

// Reading the data in.

await download(path, "FuelConsumption.csv")

path = "FuelConsumption.csv"

df = pd.read_csv("FuelConsumption.csv")

df.head()

↳ // take a look at the dataset

dataframe in pandas

// Data Exploration

df.describe() # to summarise the data

cdf = df[['ENGINESIZE', 'CYLINDERS', ...]]

cdf.head(9)

cumulative distribution function

(finds cumulative probability for given value)

#To visualize and plot

viz = cdf[['CYLINDERS', 'ENGINESIZE']]
viz.hist()
plt.show()

#Now, plot each features against the Emission, to see linear relationship.
plt.scatter(cdf.FUELCONSUMPTION_COMB, cdf.CO2EMISSIONS, color='Blue')
plt.xlabel("Fuel Consumption_Comb")
plt.ylabel("Emission")
plt.show()

#Split dataset into train and test sets. 80% for training & 20% for testing.
We can create a mask to select random rows using np.random.rand() function:

msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]

#Train data distribution
plt.scatter(train.ENGINE, train.CO2EMISSIONS, color='Blue')
plt.xlabel("ENGINE size")
plt.ylabel("Emission")
plt.show()

using sklearn package to model data

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
train_x = np.asanyarray(train[['Engine size']])
train_y = np.asanyarray(train[['CO2 emissions']])
# for converting the inputs to numpy arrays
regr.fit (train_x, train_y)
print ('coefficients: ', regr.coef_)
print ('Intercept: ', regr.intercept_)
```

parameters of fit line.

```
predictions = regr.predict(test_x)
print("Mean Absolute Error: %.2f" % np.mean(
    np.absolute(predictions - test_y)))
```

Multiple Linear Regression

→ Independent variables effectiveness on prediction

→ Predict impacts of changes.

We can predict continuous values

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

$$\hat{y} = \theta^T x$$

$$\theta^T = [\theta_0, \theta_1, \theta_2, \ldots]$$

$$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \end{bmatrix}$$

optimised parameters are the ones which lead to a model with the fewest errors

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

(need to minimise MSE value)

→ How to estimate θ?

→ ordinary least squares.

→ An optimisation algorithm.

→ gradient descent

Classification:

→ Supervised learning approach
→ categorising some unknown items into a discrete set of categories or 'classes'
→ The target attribute is a categorised variable

→ Classification determines the class label for an unlabelled test case.

Classification algorithms →
→ Decision Trees
→ Naive Bayes
→ Linear Discriminant Analysis
→ K-Nearest Neighbour
→ Logistic Regression
→ Neural Networks.
→ SVM ⇒ Support Vector Machines.

KNN:

The K-Nearest Neighbors algorithm is a classification algorithm that takes a bunch of labelled points & uses them to learn how to label other points.
→ A method of classifying cases based on their similarity to other cases.
→ Cases that are near each other are neighbors

Algo → KNN

→ Pick a value for K.

→ Calc the distance of unknown case from all cases

→ Select the K-observations in the training data that are nearest to the unknown data point

→ Predict the response of the unknown data point using the most popular response value from the K-nearest neighbors

Evaluation metrics for classification

Accuracy measurements

→ Jaccard index

y : Actual labels

$\hat{y}$: Predicted labels

$$J(y, \hat{y}) = \frac{|y \cap \hat{y}|}{|y \cup \hat{y}|} = \frac{|y \cap \hat{y}|}{|y| + |\hat{y}| - |y \cap \hat{y}|}$$

$y = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]$

$\hat{y} = [1, 0, 0, 0, 0, 1, 1, 1, 1, 1]$

$$J(y, \hat{y}) = \frac{8}{10 + 10 - 8} = 0.66$$

F1-score:

$$\text{Precision} = \frac{\text{True Positive}}{(TP + FP)}$$

$$\text{Recall} = \frac{TP}{(TP + FN)}$$

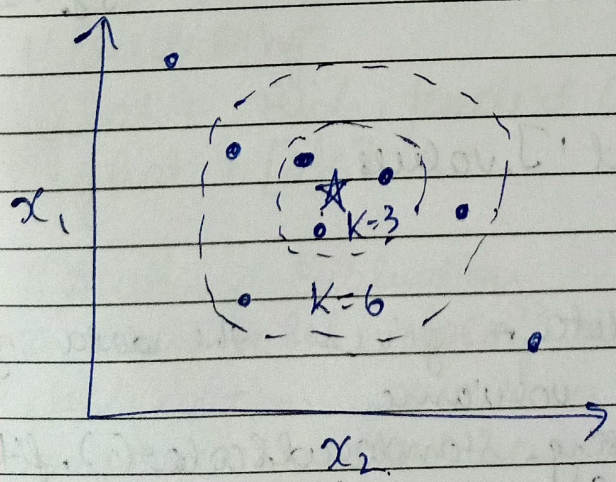
$$\text{F1-score} = \frac{2 \times (\text{precision} \times \text{recall})}{(\text{precision} + \text{recall})}$$

Both Jaccard & F1-score can be used for multi-class classifiers.

Log loss.

- performance of a classifier where the predicted output is a probability value between 0 and 1.

$$\text{Log loss} = \frac{-1}{n} \sum (y \times \log(\hat{y}) + (1-y) \times \log(1-\hat{y}))$$



To build a classifier, to predict unknown cases (by using K nearest neighbour)

// load data from CSV file

!wget download (path, "teleCust1000t.csv")
df = pd.read_csv ('teleCust1000t.csv')
df.head()

// Data visualisation & analysis.
df['custcat'].value_counts()

O/P:- 281 Plus service, 266 - Basic service, 236 Total service & 217 E- service customers

df.hist (column = 'income', bins = 50)

df.columns

(To use scikit-learn, we have to convert the pandas data frame to a Numpy array)

X = df [['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed' ...]].values
X = [0:5]

y = df['custcat'].values
y = [0:5]

// Normalize data ⇒ gives the data zero mean & unit variance.
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X = [0:5]

// Train, Test, split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 10)

print ('Train set :', X_train.shape, y_train.shape)
print ('Test set :', X_test.shape, y_test.shape)

// Classification
// K nearest neighbour (KNN)

from sklearn.neighbors import KNeighborsClassifier

// Training

K = 4
neigh = KNeighborsClassifier(n_neighbors = K).fit(X_train, y_train)

neigh

// Predicting
yhat = neigh.predict(X_test)
yhat = [0 5]

// Accuracy Evaluation
In multilabel classification, accuracy classification score is a function that computes subset accuracy = jaccard_score (function.) It calculates how closely the actual labels and predicted labels are matched in the test set.

```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_
      score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_
      score(y_test, yhat)
```

To calculate the accuracy of KNN for different values of K.

```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))

for n in range(1, Ks):
    neigh = KNeighborsClassifier(n_neighbors=n).
            fit(X_train, y_train
    yhat = neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score
            (y_test, yhat
    std_acc[n-1] = np.std(yhat == y_test)/
            np.sqrt(yhat.shape[0])

mean_acc
```

Decision Tree

→ Each internal node corresponds to a test
→ Each branch corresponds to a result of the test
→ Each leaf node assigns a classification

Algo

→ Choose an attribute from your dataset.
→ Calculate the significance of attribute in splitting of data
→ Split data based on the value of the best attribute
Repeat again

Building Decision Tree:

→ More Purity
→ Less Entropy
→ Less Impurity.
→ Pure node.

Entropy → measure of randomness or uncertainty.
Lower the entropy, the less uniform the distribution, the purer the node.

$$\text{Entropy} = -p(A)\log_2(p(A)) - p(B)\log_2(p(B))$$

probability

Information gain is the information that can increase the level of certainty after splitting

Information Gain = (Entropy before split) - (weighted entropy after split)

```
from sklearn.tree import DecisionTreeClassifier
import sklearn.tree as tree
// Download data
path = '    .csv'
path = "drug200.csv"
// Read the data
my_data = pd.read_csv("drug200.csv", delimiter=",")
my_data[0:5]

// Pre processing
// using my_data as Drug.csv
// X as the Feature Matrix
// y as the response vector

X = my_data[['Age', 'Sex', 'BP', ...]].values
X[0:5]

from sklearn import preprocessing
le_sex = preprocessing.LabelEncoder()
le_sex.fit(['F', 'M'])
X[:, 1] = le_sex.transform(X[:, 1])

le_BP = preprocessing.LabelEncoder()
le_BP.fit(['LOW', 'NORMAL', 'HIGH'])
X[:, 2] = le_BP.transform(X[:, 2])

:
X[:, 3] = le_chol.transform(X[:, 3])
X[0:5]
```

```
y = my_data["Drug"]
y[0:5]

// Setting up the DecisionTree

from sklearn.model_selection import
    train_test_split

X_trainset, X_testset, y_trainset, y_testset =
        train_test_split(X, y, test_size=0.3,
                         random_state=3)

// Modelling.

drugTree = DecisionTreeClassifier(criterion="entropy",
                                  max_depth=4)

drugTree.

drugTree.fit(X_trainset, y_trainset)

// Prediction
predTree = drugTree.predict(X_testset)

print(predTree[0:5])
print(y_testset[0:5])

// Evaluation
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy:",
      metrics.accuracy_score(y_testset, predTree))

// Visualize
tree.plot_tree(drugTree)
plt.show()
```

Logistic Regression.
→ is a classification algorithm for categorical variables (like 0/1, binary).

When is logistic regression suitable
→ if data is binary.
Y/N, T/F, 0/1.
→ if you need probabilistic results.
→ when you need a linear decision boundary.
→ when we need to understand the impact of a feature.

Building a model for customer churn

$X \in R^{m \times n}$

$y \in \{0, 1\}$

$\hat{y} = P(y=1|x)$

label / actual value

Sigmoid function
→ Logistic function.

$$\sigma(\theta^T X) = \frac{1}{1 + e^{-\theta^T x}}$$

$\sigma(\theta^T X) = 1$

$\sigma(\theta^T X) = 0$

$[0, 1]$

O/P of the model:
→ $P(y=1|x)$
→ $P(y=0|x) = 1 - P(y=1|x)$

Logistic Regression.
→ is a classification algorithm for categorical variables (like 0/1, binary)

When is logistic regression suitable
→ if data is binary.
Y/N, T/F, 0/1.
→ if you need probabilistic results.
→ when you need a linear decision boundary
→ when we need to understand the impact of a feature.

Building a model for customer churn

$X \in R^{m \times n}$

$y \in \{0, 1\}$

$\hat{y} = P(y=1/x)$

label / actual vector

Sigmoid function
→ logistic function.

$$\sigma(\theta^T X) = \frac{1}{1+e^{-\theta^T x}}$$

$\sigma(\theta^T X) = 1$

$\sigma(\theta^T X) = 0$

$[0, 1]$

O/P of the model:
→ $P(y=1/x)$
→ $P(y=0/x) = 1 - P(y=1/x)$

→ $\sigma(\theta^T x) \longrightarrow P(y=1|x)$

Training process (for customer churn example)

→ Initialise θ — $\theta = [-1, 2]$

→ Calculate $\hat{y} = \sigma(\theta^T x)$ for customer — $\hat{y} = \sigma([-1,2] \times [2,5]) = 0.7$

→ Compare the o/p of $\hat{y}$ with actual o/p of cust, y & record it as error — Error = 1 − 0.7 = 0.3

→ Calculate the error of all customers — Cost = $J(\theta)$

→ Change the θ to reduce the cost — θ_{new}

→ Repeat

Logistic Regression Training:

$\sigma(\theta^T x) \Longrightarrow P(y=1|x)$

→ change the weight → Reduce the cost

→ cost function $\text{cost}(\hat{y}, y) = \frac{1}{2}(\sigma(\theta^T x) - y)^2$.

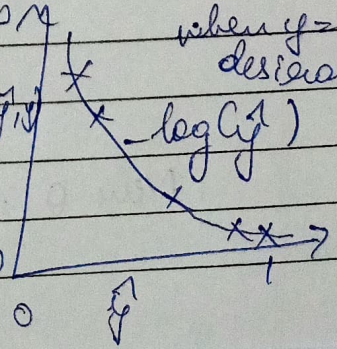
$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}(\hat{y}, y)$$

Plotting the cost function of model.

- model $\hat{y}$
- actual value $y = 1$ or 0

→ If $Y=1$, & $\hat{y} = 1 \rightarrow$ cost = 0

→ If $Y=1$ & $\hat{y} = 0 \rightarrow$ cost = large



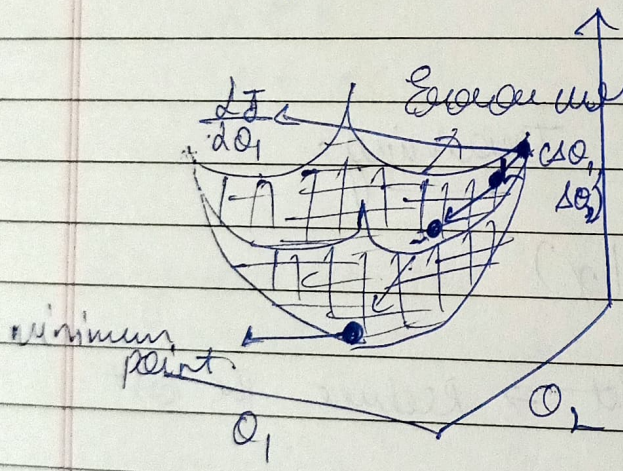
$$cost(\vec{y}, y) = \begin{cases} -\log(\vec{y}) & \text{if } y = 1 \\ -\log(1-\vec{y}) & \text{if } y = 0 \end{cases}$$

$$\Rightarrow J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} y^i \log(\vec{y}^i) + (1-y^i) \log(1-\vec{y}^i)$$

Minimise the cost func^n of the model.

→ using Gradient Descent.

A technique to use the derivative of a cost function to change the parameter values, in order to minimise the cost or error.



$$\frac{\partial J}{\partial \theta_1} = \frac{-1}{m} \sum_{i=1}^{m} (y^i - \vec{y}^i) x_i$$

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial \theta}{\partial \theta_K} \end{bmatrix} \rightarrow \text{gradient vector.}$$

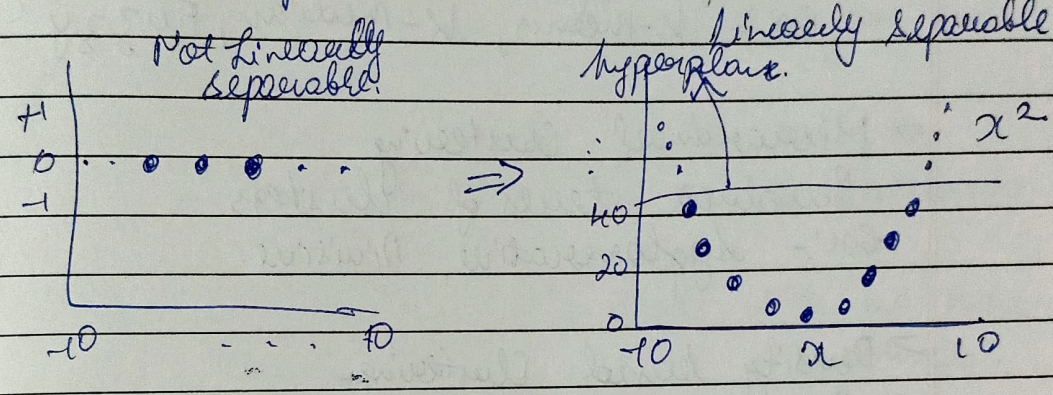
New θ = old $\theta - \eta \nabla J$

Support Vector Machines

SVM is a supervised algorithm that classifies cases by finding a separator.
→ first maps data to a high dimensional feature space
→ second, finding a separator.

The SVM algo o/ps an optimal hyperplane that categorises new examples.

Data transformation.



$$\phi(x) = [x, x^2]$$

Mapping data to higher dimensional space → is Kernelling
- linear
- poly
- RBF
- sigmoid.

→ they are accurate in high-dim spaces
→ memory efficient.
→ no probability estimation.

Application → image recognition, text assignment, Detecting spam, sentiment analysis, gene classification, regression

Clustering

→ finding clusters in a dataset, unsupervised

Cluster → A group of objects that are similar to other objects in the cluster & dissimilar to data points in other clusters.

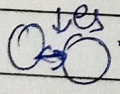
Clustering Algorithms

→ Partitioned based Clustering
Relatively efficient
Ex:- K-means, K-median, Fuzzy C-means

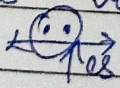
→ Hierarchical Clustering
Produces trees of clusters
Ex:- Agglomerative, Divisive.

→ Density based Clustering
Produces arbitrary shaped clusters
Ex:- DBSCAN.

K-means Clustering



- intra cluster distances are minimised
- inter cluster distances are maximised



$$Dis(x_1, x_2) = \sqrt{\sum_{i=0}^{n} (x_{1i} - x_{2i})^2}$$

(Minkowski distance)

→ Initialize K (K=3) centroids randomly
→ distance calculation → (distance matrix is formed)
→ Assign each point to the closest centroid.

SSE: the sum of the squared differences between each point & its centroid

$$SSE = \sum_{1}^{n} (x_i - c_j)^2$$

→ Compute the new centroids for each cluster.
(that is, each cluster center will be updated to be the mean for datapoints in its cluster).
→ Repeat the steps until there are no more changes.

K-means Accuracy.
- External approach
  compare the clusters with the ground truth, if it is available.
- Internal approach
  average the distance b/w data points within a cluster.

Elbow method is used for finding the right 'K'.