DS

Linear Regression

Linear Regression is one of the supervised ML algorithms. In supervised ML, there is a dependent variable(s) that has to be predicted through independent variable(s). Please note that Linear Regression will prove correlation. not causality. Remember what Levitt said!
https://lnkd.in/fJC6aQp
Linear Regression will help us in **establishing relationship (h(x))** between dependent variable (on Y Axis) and independent variable or explanatory variable (on X Axis). For instance, Uber predicts the price to be paid (Y) based on a number of features like distance, traffic, battery level, purchasing capacity etc., (X). Note that linear regression is done for continuous variables (both X and Y). If there are categorical variables, you'll have to transform those categorical variables into dummy variables.

Based on number of dependent and explanatory variables, linear Regression is of various types.

- **Simple** linear regression- one dependent and independent variable. Also called **Uni-variate** linear regression.
- **Multiple** linear regression- multiple independent variables.
- **Multivariate** linear Regression- 2 or more dependent variables.

You can solve a regression problem in two ways:

- **Stats way:** Here you derive what is the slope of the equation and intercept.
  - This is an algebraic way of finding the parameters at which J is minimum. This is called **NORMAL EQUATIONS METHOD.** There is no need to chose alpha, no need to iterate with this method. Moreover, there is no need to do feature scaling. However, for larger datasets those including large number of features, the iterative Gradient Descent Method is the best.
  - 

    => Finding parameters of Linear Regression Equation through Normal Equation method.
  - There is a danger of **non-invertibility** here. Use pseudo-inverse function in Matlab/Python to provide inverse even if the matrix is non-invertible.
- **Stochastic way or iterative way:** Basically, you plot the (x,y) points on the graph and then draw a line iterating across various slopes and y-intercepts to ensure '**good fit'.** Good fit is that line which produces **least** error between the regression-model generated values (h(x)) and the actual values (y). This is achieved by minimizing the **Cost Function (J)** over both slope and intercept. If you observe, J is a function of both **slope** and **intercept.** When you plot J against both these variables, you get a **parabolic contour (Convex)** from which you can visually see the minimum value of J.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

=> This is Squared Error Function.
There are various other types of Cost Functions as well.

But you do not want to manually plot J for each data set and then find where it has minimum value. Moreover, in multi-variable linear regression problems, there are many more slope parameters. To our rescue, there is an automatic way of finding the parameters at which J is minimum => **GRADIENT DESCENT!** In this we **try** various values of our parameters in steps and then find out the parameter value at which our J is minimum. This is how the descent happens:

It is also a good practice to plot minJ and Number of iterations to check if your algorithm is properly working. Instead of giving a condition of when your iteration should stop, having a look at this plot will give you more intuitive sense about convergence.

Feature Engineering: You need not consider the features as they are given. Sometimes number of features can be reduced by applying mathematic operations within them. Eg: Say two features of length and breadth are given => You can reduce it to single feature of Area.
Also remove redundant features, all the independent features should be truly independent of each other. Domain expertise helps here.

Polynomial Regression: This can be done by converting simple polynomial regression into multivariate linear regression! Eg: Make x^2 as another feature. Do feature scaling though.

Validating your Regression Model:

- **Residual Plots:** x on X-Axis / Residuals on Y-Axis.
  - This residual plot should be linear, normaly distrbuted and homoscedasticity for the regression model to be valid.
  - Otherwise, re-do regression.
- **Hypothesis Testing:**
  - **Null:** No relation between y and x => Slope = 0
  - **Alt:**
- **R-Square:** It is the co-efficient of determination. It is the percentage of variation in 'y' that is explainable by 'x' variable.
  - $R^2 = (\frac{SSR}{SST}) = 1 - (\frac{SSE}{SST})$
    - SST = SSR + SSE => nothing but SS_{yy}
    - SSE = Sum of squared errors => $SSE = \sum (y_i - y')^2$
    - SSR = Regression sum of squares => $SSR = \sum (y' - \bar{y})^2$
  - Higher the R2, better the regression, representing that the line is closer to the actual y values. The scikit learn package in python implements this method of linear

regression. https://youtu.be/WuuyD3Yr-js
    - If R2 = 0; it means you cannot predict y using x
  - Just blindly aiming for high R2 is not what a good ML engineer should do. Using 100s of features to get high R2 is only making the system complex, time consuming and costly. Also, many times adding a new feature will not help you much. For instance, say considering 5 features alone will give you a R2 of 80% and adding a 6th feature will give you a R2 of 80.1%. How will you find such features that are not very important for us? ==> **Adjusted R2!**
    - Adjusted R2 =

$$AdjustedR^2 = 1 - \frac{\frac{SSE}{n-k-1}}{\frac{SS_{yy}}{n-1}}$$

      ; where n-1 = number of columns (degrees of freedom); k= number of y values.
    - This value will start reducing when unneccesary features are in the model. Hence, highest Adjusted R2 will not be equivalent to the model with highest R2.
- On sk-learn there is an inbuilt validation available.

Linear Regression Code on Python - ML way: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html => Performs Ordinary Least Squares (OLS) regressio

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.1,random_state=0)
LR= LinearRegression(fit_intercept = True, normalize = False, copy_X=True )
LR.fit(x_train,y_train)
LR.score(x_test,y_test) ==>> Gives out R2 value
prediction = LR.predict(predict_this)
```

Linear Regression Code on Python - Stats way:

```
import statsmodels.api as sm
X = sm.add_constant(df_x)
lr_model = sm.OLS(y,X).fit()
predictions = lr_model.predict(X)
summary = lr_model.summary()
```

Few issues we face and how to avoid: http://r-statistics.co/Assumptions-of-Linear-Regression.html
https://onlinelibrary.wiley.com/doi/full/10.1111/ceo.12358

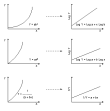
- **Multi-collinearity:** https://medium.com/@raj5287/effects-of-multi-collinearity-in-logistic-regression-svm-rf-af6766d91f1b
  - In linear regression, all features should be independent. The goal is to check how Y varies for unit change in a feature i.e the weight / coefficient for each feature.
  - But when the features are collinear or dependent on each other, a unit change in one feature will impact another features as well. There by, we will not be able to obtain correct values for the weights.
    - Eg: Say Your marks (Y) are dependent on hours studies (x1) and hours slept(x2). But even

Eg, say your marks (Y) are dependent on hours studies (x1) and hours slept(x2). But even x1 and x2 are dependent on each other. If x2 increases, x1 reduces. Thus the weights assigned to each of the features will not correctly describe the relationship between Y and that feature.

- Also p-value reduces, variance becomes unstable (**(Research))**
- During this, R-square value will improve when a feature is dropped.
- Step down regression will help us avoid this issue. Take all features first; drop each;
- **Variance Inflation Factor:** Do regression within the X's => Take each feature and do regression with other features. If you find high R-square anywhere, it means the features are not really independent. Do feature engineering after this.
- **NOTE:** Multi-collinearity is not a problem every time. Some times, we create polynomial features if the data is not linear. Obviously x and X-square will have collinearity, but that will not hurt you. Multi-collinearity is problem when it exists between two supposedly independent variables (x and z)

```
# If you want to check VIF
vif=pd.DataFrame()
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif['VIF Factor'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['Features'] = X.columns

#If you want to remove features, step-by-step, if VIF is above a threshold
from statsmodels.stats.outliers_influence import variance_inflation_factor
def calculate_vif_(X, thresh=5.0):
    variables = list(range(X.shape[1]))
    dropped = True
    while dropped:
        dropped = False
        vif = [variance_inflation_factor(X.iloc[:, variables].values, ix)
               for ix in range(X.iloc[:, variables].shape[1])]


        maxloc = vif.index(max(vif))
        if max(vif) > thresh:
            print('dropping \'' + X.iloc[:, variables].columns[maxloc] +
                  '\' at index: ' + str(maxloc))
            del variables[maxloc]
            dropped = True


    print('Remaining variables:')
    print(X.columns[variables])
    return X.iloc[:, variables]
```

- **Non-Linearity:**
  - Perform log transform --



  - Or take polynomial expression as another variable => **x^2 = z =>** Becomes a multi-variate

linear regression.

```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree= , include_bias=T/F, )
X_poly = poly_features.fit_transform(X)
#This will add new polynomial features 'up to' the degree specified. When multiple features
(x1,x2) are present even x1*x2 feature will be created.

If there are n features and degree=d, (n+d)!/n!.d! features will be created
```

- **Heteroscedasticity:**

```
from yellowbrick.regressor import ResidualsPlot
visualiser = ResidualsPlot(LR)
visualiser.fit(X_train,y_train)
visualiser.score(X_test,y_test)
visualiser.poof()
```

    - Perform log transform on y and/or x to over come this problem.
- **Outliers:**
    - See box-plots/ remove if needed.
        - But box-plots help us identify only uni-variate outliers.
    - For multi-variate outliers?
        - Scatter plots will help in visualising.
- **High leverage points:**
    - Cooks distance plot
    - http://home.iitk.ac.in/~shalab/regression/Chapter6-Regression-Diagnostic%20for%20Leverage%20and%20Influence.pdf
- **Influence points:**
- **Correlation of error terms:**
    - Durbin Watson should be near 2
- **Residual Errors should be normally distributed:**

```
from scipy.stats import anderson
result = anderson(err)
print('Statistic: %.3f' % result.statistic)
p = 0
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < result.critical_values[i]:
        print('%.3f: %.3f, data looks normal (fail to reject H0)' % (sl, cv))
    else:
        print('%.3f: %.3f, data does not look normal (reject H0)' % (sl, cv))
```

Regularisation of Regression: Add a regularization term to the Loss function.

- **Ridge - L2 =>** Supports well distributed coefficients.

- Not good when there are outliers in data

```
from sklearn.linear_model import Ridge
l_ridge = Ridge(alpha=0.3, random_state=0)
l_ridge.fit(X_train, y_train)
```

- **Lasso - L1 =>** Supports sparsity of coefficients. Insignificant features are eliminated by giving them zero coefficients.

```
from sklearn.linear_model import Lasso
l_lasso = Lasso(alpha=0.3, random_state=0)
l_lasso.fit(X_train, y_train)
```

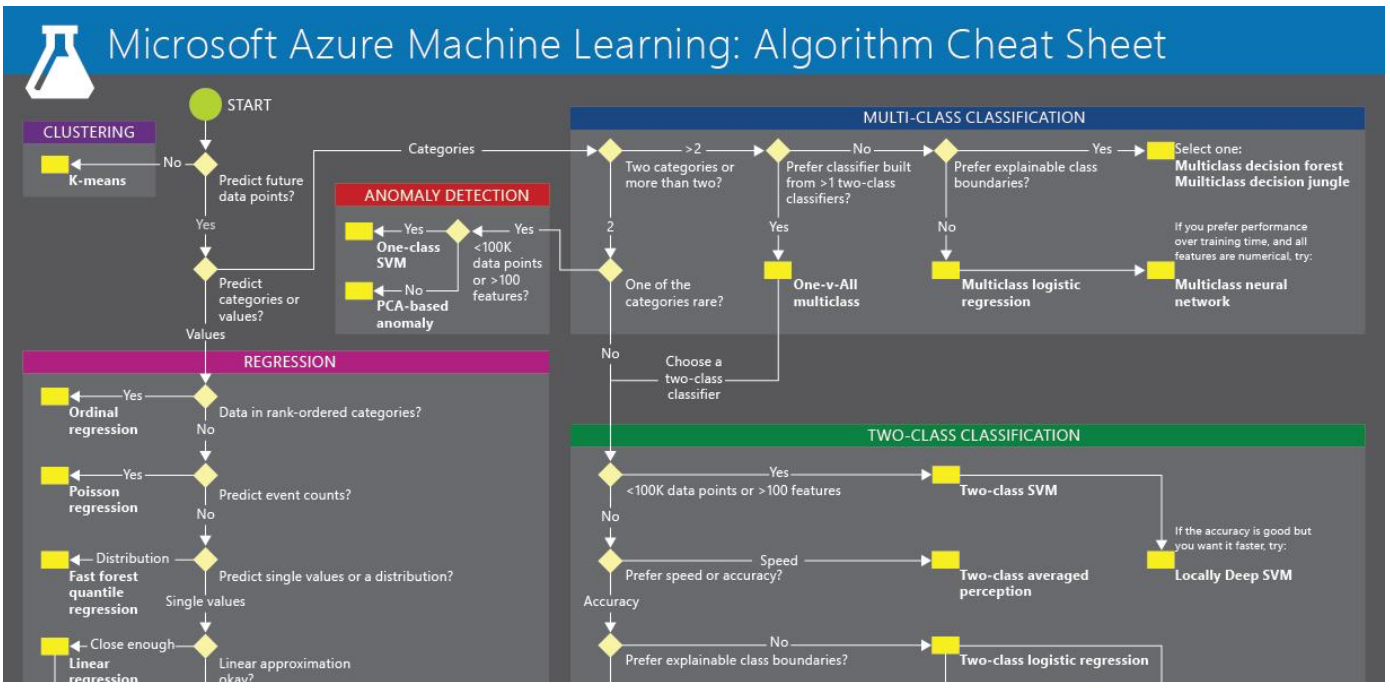
-

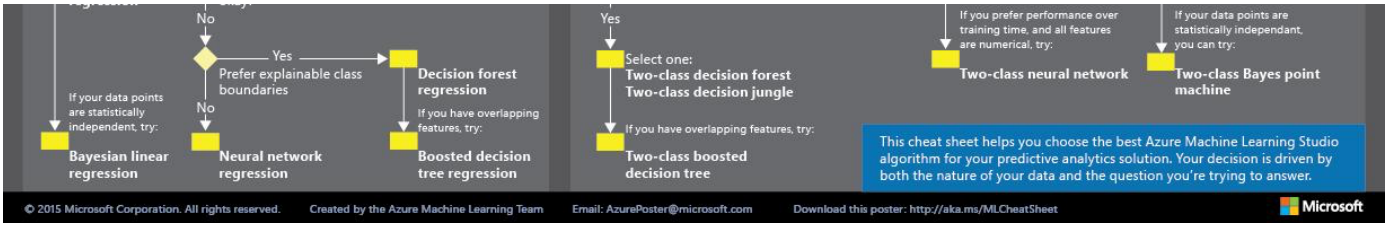
ExcelMisc

import sys
!{sys.executable} -m pip install imblearn

sparse matrix.

randomsapling of rows in pd.read_csv





Important Excel Functions to know:

- Flash Fill
- INDEX-MATCH
- VLOOKUP
- Conditional Formatting
-

Statistics

Descriptive Statistics: You describe a data - give mean, mode, median, max, min, standard deviation, variance.

- Outliers detection - using boxplots.
- Probability -- Bayes theorem
- Distributions -- Normal, Bernouli, Geometric, Poisson.

Inferential Statistics: You take a sample space from a population to infer some information. Eg: You want to find mean water usage in a city, you cannot collect data from all the households. Instead, you would take samples and infer the water use information based on this sample space.

- T Test, Central limit theorem
- Confidence Interval = Prediction strength
- Hypothesis Testing- Type I, II errors
- Chi Square Test
- Correlation vs Co-Variance.

Statistical Modelling => In Machine Learning Note - Machine Learning - Teaching Computers to learn and think for itself

- Linear Regression
- Logistic Regression
- Time Series Modelling

DESCRIPTIVE STATS

Mean, Median and Mode are three CENTRAL TENDENCIES.

MEAN: Average

- Effect of outliers is huge. Hence using mean to describe a data is wrong when there are outliers. **OUTLIER DETECTION through BOX PLOTS** is thus crucial. If you have to include Outliers, then prefer describing data through **median.**

MEDIAN: Arrange the numbers in the increasing order. Middle value. It is also called 50th percentile.

- If n= odd, median =

$$\left(\frac{n+1}{2}\right)^{th}$$

term

- if n=even, median = Average of

$$\left(\frac{n}{2}\right)^{th}$$

and

$$\left(\frac{n}{2}+1\right)^{th}$$

term.

- But median fails when data is highly skewed. To understand this skewness, check quartiles as well:
  - Q1 = 25th Quartile
  - Q3 = 75th Quartile
  - Q3-Q1 =Inter-quartile range.

MODE: Most frequent occurring item in the data.

- This is the only central tendency that can be used for categorical data.

To obtain all the above for data in Pandas=> df.describe()

But is central tendency enough to describe data? What if three students' marks data have same mean, median and mode? How can you select the best student of the three using the central tendency? Thus the next set of descriptive statistics involves describing the variability of the data.

Spread, Variance & Standard Deviation, Z-Score describe VARIABILITY of the data

SPREAD: (Max-Min) **-- Range.**

- Again, Outliers hugely affect the spread. If you cannot eliminate the outlier because they are natural, then you need **Variance** to describe the data efficiently.

VARIANCE: You are seeing how data records are varying from the mean on an average. You square it cuz, there will be negatives as well.

$$\sigma^2 = \sum \frac{(x_i - \mu)^2}{n}$$

Where, n = number of records;

μ

= Mean ;

- **Standard deviation =**

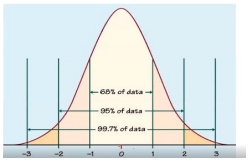
  σ

Z-SCORE: It gives how far each data record is from the mean, in terms of number of standard deviations

$$z = \frac{(x - \mu)}{\sigma}$$

- Helps in **OUTLIER DETECTION**. Generally if the data record is more than 3

  σ

  away from the mean, i.e., **z-score = 3**, then it can be called as outlier.
- 

  => Ideal case.
- When you want to compare Z-scores of different values, standardisation is important.

PROBABILITY:

$$P = \frac{(\ No.\ of\ favourable\ outcomes\ for\ a\ condition)}{All\ possible\ equally\ likely\ outcomes}$$

- **Frequentist:**
  - When you see a probability problem after conducting the experiment frequently.
  - **Eg**: Probability of head on a coin => You see this as tossing a coin multiple times to come to a conclusion that 1/2 is the probability.
  - Hence Frequentist approach includes finding long frequency distributions.
  - Fix parameters first

- **Bayesian:**
  - When you see a probability problem as dependent on the past history. This includes studying subjectivity or bias in the question, if any.
  - **Eg**: Probability of sun rising in the east => It is 1! = past history says that.
  - Hence, Bayesian approach calculates the Prior
  - Parameters are fixed later.
- Probability of Singular event
- Probability of Multiple events
- Empirical vs Classical Probability
  - Empirical => Real-Time Data => Close to Bayesian approach. Eg: Who is more likely to win? India or Pak
  - Classical => Close to Frequentist. Eg: Tossing a coin.
- **Dependent and independent events**. There is a degree of subjectivity in deciding if the events are dependent or not.
  - where A and B are **dependent events.**
    - Joint probability, **P(A and B) = P(A) . P(B|A) = P(B) . P(A|B) => Bayes Theorem**
  - $P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$
  - where A and B are **independent** events i.e A will not affect the probability of B happening. **P(A|B) = P(A); P(B|A) = P(B)**
    - Eg: When you toss a coin multiple times, each toss is independent of the other.
    - $P(A \text{ and } B) = P(A) \cdot P(B)$
      - If this is 0, then A and B are called disjoint events.
- Conditional probability: **P(A|B) =**

$$\frac{P(A \text{ and } B)}{P(B)}$$

  - https://www.khanacademy.org/math/statistics-probability/probability-library/conditional-probability-independence/v/conditional-probability2
  - Basically, what you are doing here is that you are reducing the total sample space (denominator), because one event has already happened. There is no need to consider the entire sample space.
  - If A and B are independent events
    - P(A|B) = P(A)
  - tree method is best to solve conditional probability problems. Ensure that the tree is equally branched always. Alternatively, use tables.
  - Bayes Theorem => Reverse Probability.
    - **P(B|A) = P(B) . P(A|B) / P(A)**
    - P(B) = Prior
    - P(A|B) = Likelihood or test evidence
    - P(B/A) = Posterior

        - P(A) = Normalisation
    - If you observe, the probability of an event is thus changing when conditions change. This is the foundation of machine learning.
        - How will you change the condition? = By repeating the event, thereby generating more conditions. That is why we iterate while using Bayesian probability until we get high probability.

CO-VARIANCE:

CORRELATION:

- **Pearson**:

$$r = \frac{\sum(X-\overline{X})(Y-\overline{Y})}{\sqrt{\sum(X-\overline{X})^2}\sqrt{(Y-\overline{Y})^2}}$$

Where, $\overline{X}$ = mean of X variable
$\overline{Y}$ = mean of Y variable

{Covariance of X and Y / Std of X * Std of Y}

- **Spearman:** Same as above, but instead of X and Y, it is Rank(X) and Rank(Y)
- Kendall
- The above three tests help in finding correlation

DISTRIBUTIONS

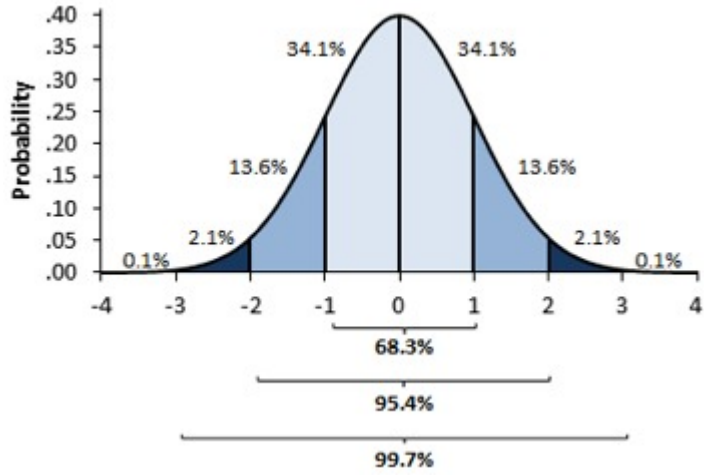
- Find **PMF** or **PDF =>** Checks plots note.
    - PMF for discrete
    - PDF for continous.
- Check which distribution the data is patterned: Binomial, Gaussion, Poisson, Normal. Real-world data will not be 'perfect' distribution. There will be skewness. Make necessary changes to suit these distributions. Otherwise invent new distribution and find formulas for mean, median, probability etc.

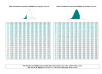
BINOMIAL: This is for discrete data

- When there are only two possibilities of an event (Such events are called **Bernoulli events**) in sample space => Yes or No | Success or Failure |
- n = sample space length.
- r = Number of required outcomes in the sample space.
- p,q => Two outcomes of an event. **p= probability of required outcome / q = 1-p**
- **P(X=r)** = Probability that there are r required outcomes in the sample space of n events is:

- $\binom{n}{r} p^r q^{n-r}$
  - This is also called Binomial probability
- Expectation **E = n.p**
- Variance **Var = n.p.q**
- If p = 0.5 | No skew.

NORMAL (GAUSSIAN): This is for continous data | Bell curve

- Mean = Mode = Median
  - If there is skewness, they will be different.
- 68-95-99.7 => Thumb of rule
- Zero Skew and Kurtosis
- It's called 'Normal' because, most of the natural data will follow this pattern. So check if the sample is normal or not and then you can infer much information about the data.
- 

$$= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$
$\mu =$ Mean
$\sigma =$ Standard Deviat
$\pi \approx 3.14159\cdots$
$e \approx 2.71828\cdots$
- Area under the graph between certain range of X will give the probability. But integrating the above equation is a painful task.
  - **LOOKUP Tables!** => For standard values of mean and SD (0,1), it is the table of calculated areas for different values. Remember that area under the curve of normal curve will not change when mean and SD are changed. It is a symmetric equation!
  - 
  - So any normal equation can be converted into the standard normal equation using **Z-score!** That is why the shape of the graph is not changing.

GEO-METRIC DISTRIBUTION: It is a part of the binomial distribution. But here you are not bothered about 'r' required outcomes, but about the first required outcome.

- n = Number of trials.
- r = Number of the trial in which we get the first required outcome. (So fail r-1 times)
- **P(X=r)** = Probability that there is a first success in the rth trial:

$$q^{(r-1)} \cdot p$$

- **P(X>r)** = Probability that there is a success after r trials

$$q^{(r)}$$

(Failure pakka in first r trials, what happens after it is not a matter of interest in this question)

- **E(X) =**

$$\frac{1}{p}$$

- **V(X) =**

$$\frac{q}{p^2}$$

POISSON DISTRIBUTION: This is also for discrete data. This is used when time component is present in the question.

- **Applications:** Ticketing.
  - If 100 tickets are sold on an average (lambda) in so and so time interval, what is the probability that 200 tickets (r) get sold.
- PMF = **P(X=r)** =>

$$\frac{\left(e^{(-\lambda)}\lambda^{r}\right)}{r!}$$

where lambda = average in a time interval; r = required result.

- CDF = **P(X<=r)** =>

$$\left(e^{-\lambda}\right)\sum_{i=0}^{r}\left(\frac{\lambda^{-i}}{i!}\right)$$

Continuity Correction: When you want to approximate a discrete distribution into a continous (normal)

Untitled Attachment ⚠

---

import scipy.stats as stats

To perform stat on Python

- **UNIFORM DISTRIBUTION:**
  - **stats.uniform.rvs(loc = , scale= , size=) =>** Generates 'size' random numbers in uniform distribution of:
    - start = loc

        - end = scale.
    - **stats.uniform.cdf(**x =, loc = ,scale=**) =>** Find Cumulative probability upto x value.

- **NORMAL DISTRIBUTION**
    - **stats.norm.cdf(**x = z_score, loc, scale**) =>** Will give the cumulative probability P(X<x)
    - **stats.norm.ppf**(q=P(X<x)) =>Will give the z-score value of a normal function upon passing cumulative probability
        - stats.norm.pdf(x = z_) => Gives PDF

- **POISSON DISTRIBUTION**
    - **stats.poisson.cdf**(k = lamda, mu = r) =Gives Cumulative probability P(X<r) for a poisson distribution
    - **stats.poisson.ppf**() => Gives lambda value for lambda function.
    - **stats.poisson.pmf**(k = lamda, mu = r) => Gives P(X=r) for a poisson distribution.

- **CORRELATION:**
    - **df.C1.corr(df.C2, method=) =>**Gives out the correlation coefficient between C1 and C2
    - **cor_mat = df.corr(method = 'pearson/kendall/spearman') =>** returns a correlation matrix.

```
mask = np.zeros_like(cor_mat, dtype=np.bool) #zero matrix of size cor_mat; all 0s become
false
mask[np.triu_indices_from(mask)] = True #Upper triangle becomes true
ax = sns.heatmap(cor_mat, mask=mask, cmap='coolwarm',center=0, square=True, linewidths=0.5,
annot=True, cbar_kws={'shrink': .4, 'ticks' : [-1, -.5, 0, 0.5, 1]}, vmin=-1, vmax=1)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

    - To print top correlated features from the correlation matrix:

```
#Threshold value for correlation
threshold = 0.5
corr_list = []
cor_mat = df.corr()
#Search for the highly correlated pairs
for i in range(0,cor_mat.shape[0]):
    for j in range(i+1,cor_mat.shape[1]): #avoid repetition
        if (cor_mat.iloc[i,j] >= threshold)  or (cor_mat.iloc[i,j] <= -threshold):
            corr_list.append([cor_mat.iloc[i,j],i,j]) #store correlation and columns index

#Sort to show higher ones first
s_corr_list = sorted(corr_list,key=lambda x: -abs(x[0]))

#Print correlations and column names
for v,i,j in s_corr_list:
    print ("%s and %s = %.2f" % (num_cols[i],num_cols[j],v))
    print('P-Value is: ',pearsonr(df.Hillshade_9am, df.Hillshade_3pm)[1])
```

INFERENTIAL STATISTICS

What is a sample? You cannot work or rather collect the entire data about the population. Eg: Exit

What is a sample? You cannot work or rather collect the entire data about the population. Eg: Exit Polls!, you cannot ask entire population whom they voted for. So you select a sample and work on it and infer information about the entire population through this sample. But there are various ways in which you can select a sample:

- **Random sampling**: Random index is used to pick out a sample from population.
  - use np.randn
  - Is the sample really random? => Big research question.
- **Systematic sampling:** Pick every n'th item in the population as sample
- **Stratified sampling:** Sample classes are representative of population classes.
- etc.....

- **Sampling bias:**
  - **Respondent bias or Survey bias:** Tendency of a person to answer questions in a survey untruthfully or misleadingly.
  - **Non-response bias:**
- **Various real-time applications of sampling or surveys: [Six Sigma]**
  - Quality control checks in manufacturing units.
  - Exit poll survey
  - NSSO Surveys = Consumer Expenditure, PLFS etc**.**
- **Terminology in inferential statistics:**
  - **Parameter:** A descriptive measure of the entire population. Mean, mode, variance etc.
  - **Statistic:** A descriptive measure of the sample collected. Mean, mode, variance etc.

Sampling distribution of sample means:

- Take a sample => Calculate means => put it in a list.
- Repeat the above step for multiple samples from the population.
- See the distribution of the sample means list. [**CENTRAL LIMIT THEOREM**]
  - The mean of this distribution will be =~ mean of population distribution.
  - Also, if the population distribution is normal, then the sampling distribution of sample means will be normal.
  - Even if the population distribution is not normal, then a good sampling distribution of sample means can still be normal.
  - If the SD of population distribution is

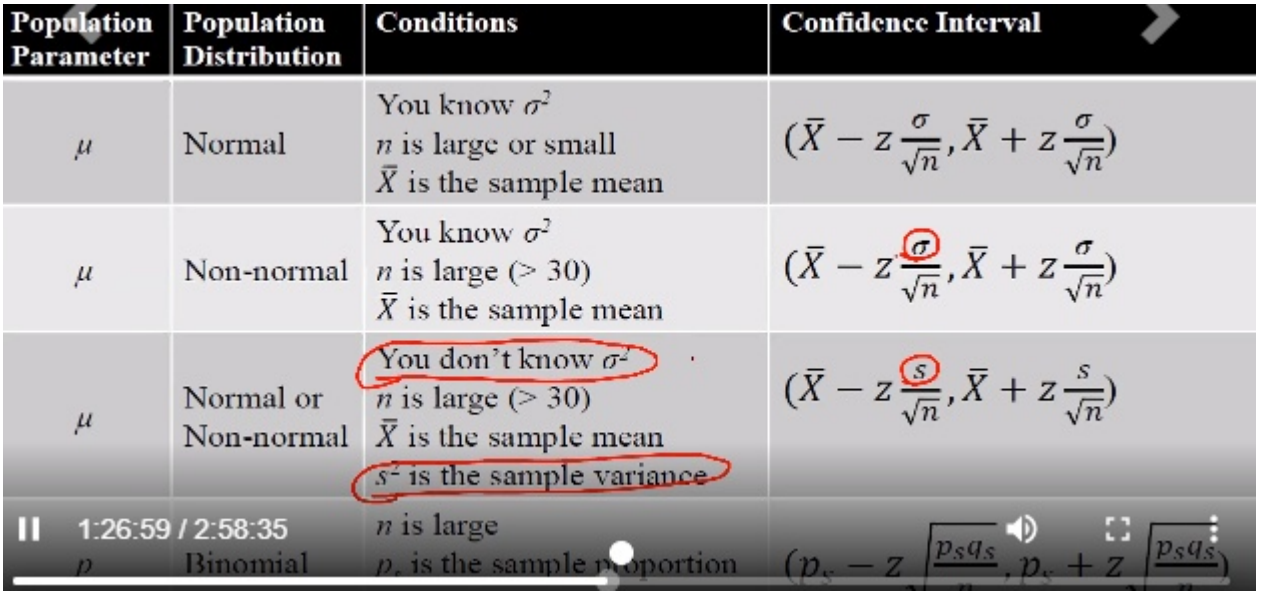
    σ

    , then the SD of sampling distribution of sample means will be

    $\sigma / \sqrt{(n)}$

    , where **n is the size of the sample space**.
    - For Sampling distributions, the SD is called **SE (Standard Error),** just to differentiate.
  - **Margin of Error = (Z-Score) * SE.**

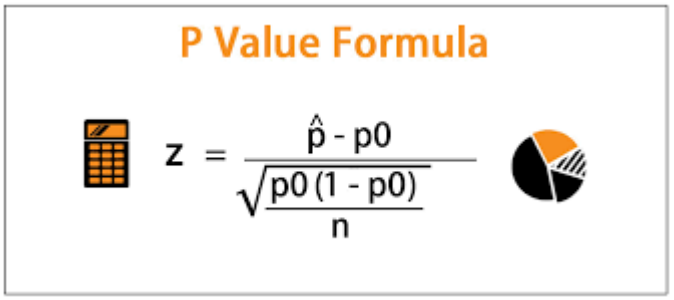
Confidence Intervals:

- 95% of the samples will fall within +/- 2 S.E of the sampling distribution. Why?
    1. Sampling distribution is a normal curve! with Sd = SE. For any normal curve this is the golden rule.
- Why use CI?
    - See, no one knows the population data and thus population mean
    - You don't even have resources to perform n number of samples to generate sampling distribution curve and then apply Central Limit theorem to obtain population curve.
    - You can only perform one sampling in any real case scenario. But how can you be sure that this sample will help you infer correct information about the population? ===>>> **CONFIDENCE INTERVAL**
    - Your population mean will lie between your sample +/- 2 S.E. You can say this statement with 95% Confidence, cuz 95% of the samples will lie between +/- 2SE of the population mean.
    - You want 99% Confidence? ==>> +/- 3 SE!
        - Margin of Error => 3* SE
    - You want 90% Confidence?
        - Find Z-Score! for 0.95 CDF!
- 

Population Parameter	Population Distribution	Conditions	Confidence Interval
μ	Normal	You know σ^2 n is large or small $\bar{X}$ is the sample mean	$(\bar{X} - z\frac{\sigma}{\sqrt{n}}, \bar{X} + z\frac{\sigma}{\sqrt{n}})$
μ	Non-normal	You know σ^2 n is large (> 30) $\bar{X}$ is the sample mean	$(\bar{X} - z\frac{\sigma}{\sqrt{n}}, \bar{X} + z\frac{\sigma}{\sqrt{n}})$
μ	Normal or Non-normal	You don't know σ^2 n is large (> 30) $\bar{X}$ is the sample mean s^2 is the sample variance	$(\bar{X} - z\frac{s}{\sqrt{n}}, \bar{X} + z\frac{s}{\sqrt{n}})$
p	Binomial	n is large p_s is the sample proportion	$(p_s - z\sqrt{\frac{p_s q_s}{}}, p_s + z\sqrt{\frac{p_s q_s}{}})$

1:26:59 / 2:58:35

Z-Score if Proportion(p) is given:



==> check numerator once.

HYPOTHESIS TESTING: Verifying a claim about the population using sample data.

- NULL Hypothesis: Claim is not false.
- Reject Null Hypothesis: The claim is wrong.
    - **Under the tail = reject the null. Otherwise fail to reject the null.**
        - Positive one-tailed test => when null hyposthesis contains <
        - Negative one-tailed test => when null hyposthesis contains >
        - Two tailed test => when null hyposthesis contains =
- **P-Value:** Probability of obtaining the observed results (sample), assuming that the Null hypothesis is true.
    - If P value is less than alpha (significance level or 100-CI) => Reject Null hypothesis.
- **Type 1 Error:** rejecting a true null hypothesis
- **Type 2 Error:** Accepting a false null hypothesis.

You cannot use Z-Distribution in two cases: ===> USE T-Distribution then.

- **Population variance is not given**
- **Sample size is small <30**

T-Distribution: When the sample size is small, the variance of the sample **(s)** will not truly describe the variance of the population. In such scenarios, T-distribution is used instead of Z-distribution. Because, Z-distribution will be skewed.

$$t = \frac{(x - \mu)}{\frac{s}{\sqrt{(n)}}}$$

==> Actually no change 😜 / just changed sigma with s.

So far we have compared only means of the samples and population. But variance of the data should also be compared. There is no point if we are able to infer correct mean and very diverge variance. Hence comes the tests to compare variance.

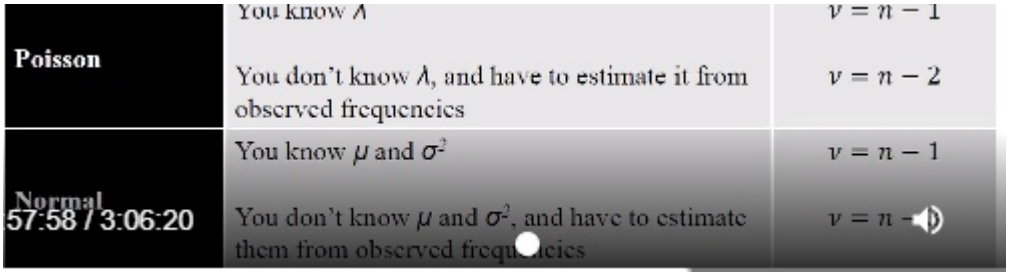
- Chi Square test
- F-Test

These are the **sampling distribution of variances**

CHI-SQUARE DISTRIBUTION: Just square all the elements of the sample.
Chi square =
Degrees of freedom. => How many levers you give to control the data.

χ^2 goodness of fit works for any probability distribution

Distribution	Condition	ν
Binomial	You know p (probability of success or the proportion of successes in a population)	$\nu = n - 1$
	You don't know p and have to estimate it from observed frequencies	$\nu = n - 2$



TESTING INDEPENDENCE OF A VARIABLE:
Done with chi square test. Helps in judging what is a more important feature.

F-TEST:
Chi square test will help us in comparing variance of the population (expected) and the sample (observed). But what if we have to compare variances of two populations? => **F-test!**

$$F = \frac{s_1^2}{s_2^2}$$

= This will be 1 when oth populations have same variance.

STATISTICAL MODELLING

- **R-Squared:**
  -

Data Pre-processing and Feature Engineering

- **Data Pre-Processing:**
  - Remove duplicate data-points => As they create unnecessary bias in the model.
  - If two features are highly correlated, remove one.
  - If the variance of the feature is very less, then this feature cannot explain the variance in the target variable. So better to remove it.
  - When there is imbalanced datasets, you could do oversampling or undersampling.
    - Oversampling => Create duplicate data points or use techniques like SMOTE (Research this)
    - Undersampling => Remove similar data points.
  - Missing values
    - If a feature has huge number of missing values, better remove the feature.
    - If a feature has very less number of missing values, you can safely remove those rows.
    - Otherwise, impute the missing values with mean, median, mode, nearest etc

Otherwise, impute the missing values with mean, median, mode, nearest etc.

- Encoding categorical variables, since most of the ML models take only numeric data as input.
  - One-Hot Encoding and Label Encoding will help when the number of categories are less.
  - But when the cardinality of the categorical variable is more, go for **supervised ratio (research this)**
- Feature Scaling:
  - Differences in the Scale will have a predominant affect in many algos. Eg: K-NN
  - So, all features should be scaled.
    - Min Max Scaling
    - Z-Score Scaling.
- Dimensionality Reduction:
  - **PCA: Prinicipal Component Analysis:** Reducing the number of features by creating a new feature that is a linear combination of original features.
- Ensure that the train and test data sets are similarly distributed. You don't want to train your model on one type of data and test it on another.

OUTLIER ANALYSIS

- First print boxplots to see outliers

```
fig , ax = plt.subplots(7,2,figsize=(20,50))
ax = ax.flatten()
index=0
for i in list(X.columns):
    sns.boxplot(X[i],ax = ax[index])
    index=index+1
```

- Then, to know % of outlier points in each feature

```
for i in X.columns:
    X.sort_values(by=i, ascending=True, na_position='last')
    q1, q3 = np.nanpercentile(X_scaled[i], [25,75])
    iqr = q3-q1
    lower_bound = q1-(1.5*iqr)
    upper_bound = q3+(1.5*iqr)
    outlier_data = X[i][(X[i] < lower_bound) | (X[i] > upper_bound)] #creating a series of
outlier data
    perc = (outlier_data.count()/X_scaled[i].count())*100
    print('Outliers in %s is %.2f%% with count %.f' %(i, perc, outlier_data.count()))
```

- To see outliers in dataframe:

```
y_outliers = df[abs(zscore(df[target])) >= 3 ]
y_outliers
```

- SVM is also a good model to detect outliers in the dataset. This is actually used extensively for use cases of Anomaly detection.

Feature Engineering includes:

- Feature Selection

    - RFE
- Feature extraction: combining existing features to create a new one.
    - Dimensionality reduction
- Create new features with new data.
    - Combine multiple features to create a new one.
        - Combine those having collinearity
        - Combine Sparse Classes

ENCODING CATEGORICAL VARIABLES

One-Hot Encoding:

```
# One-Hot encoding
pd.get_dummies(df.C1) #Where C1 is the column with categorical variables.
```

Label Encoding:

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
label_encoder.fit(df.C1) #Where C1 is the column with categorical variables.
label_encoder.transform(df.C1) #
```

VARIANCE ANALYSIS

- If the variance of the feature is very less, then this feature cannot explain the variance in the target variable. So better to remove it.

For inference problems:

If from the sample data you want to infer something about the population data, then the conditions for inference should be first met if you want to perform any parametric tests like **t-tests** etc. What are these conditions for inference?

- **Normality:** Your sample should be part of the **sampling distribution** which is normally distributed. Only then your sample can be helpful in inferring information about population.
    - If we know that the parent population is normally distributed, then we assume that sampling distribution is also normally distributed, irrespective of the sample size.
    - Sampling distribution is approximately normal as long as the sample size is large(>30). This is because of **central limit theorem.**
- **Random:** Your sample should be the outcome of a randomized experiment.
- **Independent:** Individual observations should be independent of each other. If sampling is done without replacement, then sample size should be less than 10% of the population size.

If these conditions are not met, you should have to go for non-parametric tests to conduct hypothesis testing.

- If the sample space is normally distributed and you perform linear regression in the stats model, you'll get t-test scores for each of the features. You can remove the insignificant features then, or collect more data.

FEATURE SELECTION

Why feature selection? Why not select all the features?

- **Curse of dimensionality:** As the number of dimensions increases, there is the problem of over-fitting in many ML models. Hence we have to select the most important features.
- **Occam's Razor:** For easily explainable models, we need lesser number of features.
- **Garbage in = Garbage out:** Just think, what is the use of giving 'ID' as a feeature input. Unnecessarily the ML model will give a weight to this feature in its prediction. Thus eliminate such non-informative features.

How many features to select

```
from yellowbrick.model_selection import RFECV
visualizer = RFECV(LR)
visualizer.fit(X_train, y_train)
visualizer.show()
```

There are **three major approaches** of feature selection:

- **Filter based:** We filter the features based on some metric like **correlation, chi-square.**
  - chi-square helps in determining dependency between two variables. If a feature and target variable are independent, then why use that feature?

```
# Feature selection using Pearson Correlation Coefficient.
def cor_selector(X, y,num_feats):
    cor_list = []
    feature_name = X.columns.tolist()
    # calculate the correlation with y for each feature
    for i in X.columns.tolist():
        z= np.column_stack((X[i], y))
        cor = np.corrcoef(z.T)[0,1]
        cor_list.append(cor)
        #print(cor_list)
    cor_list= [0 if np.isnan(k) else k for k in cor_list]
        # feature name
    cor_feature = X.iloc[:,np.argsort(np.abs(cor_list))[-num_feats:]].columns.tolist()
        # feature selection? 0 for not select, 1 for select
    cor_support = [True if i in cor_feature else False for i in feature_name]
    return cor_feature
cor_feature = cor_selector(X, y,5)
```

```
# Feature selection using Chi-Square.

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler
X_norm = MinMaxScaler().fit_transform(X)
chi_selector = SelectKBest(chi2, k=num_feats)
chi_selector.fit(X_norm, y)
```

```
chi_support = chi_selector.get_support()
chi_feature = X.loc[:,chi_support].columns.tolist()
```

- **Wrapper based:** Recursive Feature Elimination

```
from sklearn.feature_selection import RFE
model = LogisticRegression(solver='liblinear') # Or anyother
rfe = RFE(model, NUM_FEATURES)
fit = rfe.fit(X, y)
print("Model Feature Ranking:", fit.ranking_)
# calculate the score for the selected features
score= rfe.score(X,y)
print("Model Score with selected features is: %f (%f)" % (score.mean(), score.std()))

feature_names = np.array(X.columns)
print('Most important features (RFE): %s'% feature_names[rfe.support_])
```

- **Embedded:** Built-in feature selection models. Lasso and RF has their own models.
  - Random Forests.

Why not use all the feature selection methods at a time?

```
feature_selection_df = pd.DataFrame({'Feature':feature_name, 'Pearson':cor_support,
'Chi-2':chi_support, 'RFE':rfe_support, 'Logistics':embeded_lr_support, 'Random
Forest':embeded_rf_support, 'LightGBM':embeded_lgb_support})

# count the selected times for each feature
feature_selection_df['Total'] = np.sum(feature_selection_df, axis=1)

# display the top 100
feature_selection_df = feature_selection_df.sort_values(['Total','Feature'] ,
ascending=False)
feature_selection_df.index = range(1, len(feature_selection_df)+1)
```

To know feature importance from any algorithm:

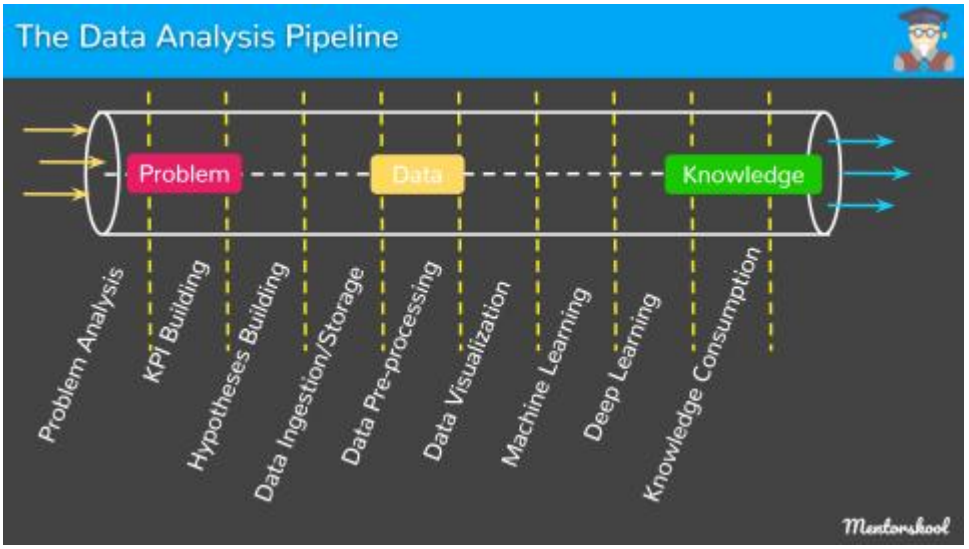
```
def get_feature_importance(clsf, ftrs):
    imp = clsf.feature_importances_.tolist()
    feat = ftrs
    result = pd.DataFrame({'feat':feat,'score':imp})
    result = result.sort_values(by=['score'],ascending=False)
    return result


get_feature_importance(classifier, features)
```

You can check if the target is variable is dependent or not on the **categorical feature variables** by performing a **chi-square contingency test.** Then you can select only those variables that are dependent. You do not need features that show effect on the features

dependent. You do not need features that show affect on the features.

```
def is_dependent(feature_col,target_col):
    table = bos_df.iloc[:,[feature_col,target_col]].values
    print('Feature Name : {0}'.format(bos_df.columns[feature_col]))
    chi_squared_stat, p_value, dof, expected=chi2_contingency(table)
    print(chi_squared_stat)
    probability=0.95
    critical=chi2.ppf(probability, dof)
    if abs(chi_squared_stat)>=critical:
        print('Dependent : Reject Hypothesis 0 (null Hypothesis)')
    elif abs(chi_squared_stat)<critical:
        print('Independent : Fail to Reject Hypothesis 0 (null Hypothesis)')
    alpha = 1.0 - probability
    print('Significance {0}, {1}'.format(alpha, p_value))
    if p_value<=alpha:
        print("Dependent : Reject Hypothesis 0 (null Hypothesis)")
    else:
        print('Independent : Fail to Reject Hypothesis 0 (null Hypothesis)')

is_dependent(3,len(df.columns)-1)
```

- Domain Knowledge is important for Feature Engineering.
  - Helps you know which features are important.
  - Helps you differentiate between correlation and causality.
- During classification problems, plot each feature's histogram with the different classifications as level of detail. If there is huge overlap, then it is not a good feature.
- Find correlation



WORKING WITH IMBALANCED DATA

Under-sampling:

- Remove instances from the majority class to find a balance.
- When we have huge data, this would be fine. Eg: Say we have 15Lakh-5Lakh imbalanced data set

- When we have huge data, this would be fine. Eg. Say we have 15Lakh-5Lakh imbalanced data set, undersampling will help as the resultant dataset will still have 10 lakh data points.
- You can either do it **randomly**
- Cluster the abundant class == Read Sergey on Quora.

```
# Under-Sampling
from sklearn.utils import resample
df_majority = df[df.<target>==0] #Majority Class
df_minority = df[df.<target>==1]

df_majority_downsampled = resample(df_majority,
                                   replace=False,    # sample without replacement
                                   n_samples=6636,     # to match minority class
                                   random_state=587) # reproducible results
# Combine minority class with downsampled majority class
df_downsampled = pd.concat([df_majority_downsampled, df_minority])
```

Over-Sampling:

- Up-sample the minority class to create a balance.
- You can do it **randomly or SMOTE**

Ensemble different resampled datasets:

-

SMOTE:

Data Masking or Obfuscation:
Hiding or modifying the data fields in order to protect it. However the data should be usable even after Obfuscation.

Causality correlation check:

- Counter factual analysis will help prove causality.
- RCTs =>
- Spurious correlation- there is a third causative factor that is causing both correlated factors.
- Regression will give you correlation not causation.

Data Processing:

- Data cleaning.
  - Trailing spaces.
  - Capitalization
- Check the type of data
  - int, float string etc.

- Categorical, numeric.

FEATURE SCALING

```
# Feature Scaling - Standardisation by removing mean and scaling to unit variance i.e
Standard normal!
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler(copy=True, with_mean = True, with_std=True) #Mean is taken as 0 if
False, std as 1 if False.

df_scaled = scaler.fit_transform(df) #Fit to data and then transform
df_scaled = scaler.fit(df) #Computes mean and std to be used for future scaling.
df_scaled = scaler.inverse_transform(df) #Scale back the data to original format
df_scaled = scaler.transform(df) #Perform transformation by centering and scaling.
```

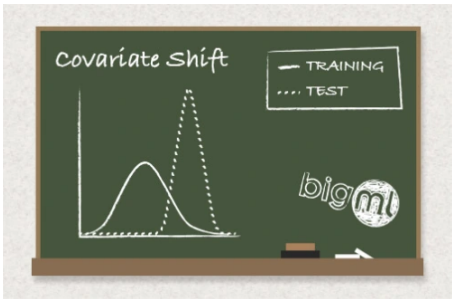
ENSURE THAT TRAIN AND TEST DATA ARE SIMILARLY DISTRIBUTED

```
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, stratify=y,
random_state=42)
```

What is stratify?

Various Data Shifts:

- One of the critical assumptions in building a robust model is that the test data or production data is from the same distribution as the training data.
- **Co-variate shift:** Your model is living in the past!



. Models are being trained and even tested on old data, but the new production data is from completely a different distribution.

- **Concept shift:**
- It is crucial to identify these shifts between our training and test/production data. Otherwise we keep giving predictions to a differently distributed data using old data. Read this to know hoe to identify it https://www.analyticsvidhya.com/blog/2017/07/covariate-shift-the-hidden-problem-of-

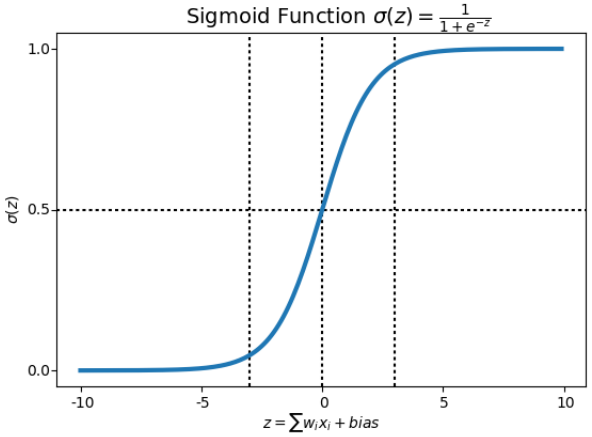
identify it https://www.analyticsvidhya.com/blog/2017/07/covariate-shift-the-hidden-problem-of-real-world-data-science/

- **Kolmogorov-Smirnov test** can also be used for this.

```
from scipy.stats import ks_2samp

for i in list(X.columns):
    print('For feature {}, the p-value is {}'.format(str(i),
ks_2samp(X_train.loc[:,i],X_test.loc[:,i])[1]))

#You'll get p-value for each feature. If it is less than alpha, reject the null hypothesis,
which is both test and train are of same distributed.
```

Logistic Regression

We use **logistic function or sigmoid function** to develop a regression model for classification. This is because, we need h(x) values within 0 and 1 for our classification. But in linear regression, h(x) exceeds this limits, giving us wrong classification. But sigmoid function has values only within 0 and 1. Hence we are actually performing sigmoid transformation over linear regression. The resultant h(x) will give us probability of y=1 i.e if h(x_predict) = 0.7, it implies 70% chance of it being classified as 1.

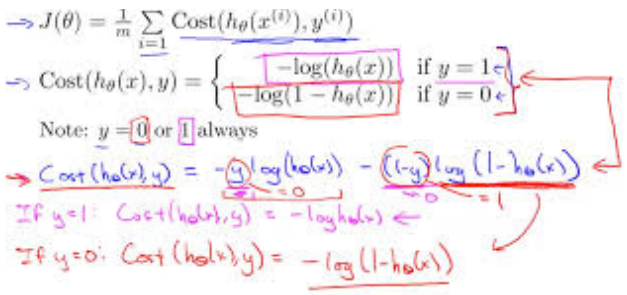


$$h_w(x) = p = \frac{1}{1+e^{-\mu}}$$

Notice that the sigmoid will be <0.5 if z is less than 0. Also

Odds Ratio = S = p/(1-p) ; where p is the sigmoid function, which gives probability of y=1 in our ML problem.
Hence S = **e^(z) ==> ln(S) = z**

Now, what is the cost function aka **error term** or **residual** in this case? **=> Negative log likelihood.**



$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

This cost function makes complete sense. When h(x) is close to 0, -log(h(x)) approaches very high values thereby increasing the cost. Therefore, whenever the model estimates 0 for y = 1, high cost will be returned. This is what we want! Similarly, if y=0, and our model predicts h(x) close to 1, again high cost will be returned.

BAD NEWS: There is no algebraic way of minimizing the J (as Normal Equations for Linear Regression).
GOOD NEWS: This cost function will be a **convex function.** Go ahead and apply gradient descent (or any optimization algorithm) and you'll find the global minimum.

Residual deviance: -2*J(w)

Validating Logistic Regression Model:

- **AIC = D + 2k || Akaike's Information Criterion**
  - **D =** Sum of residual deviance over all points.
  - **k** = Number of parameters including the intercept.

BIAS - Variance TradeOff:

- Any error has three components
  - Bias
  - Variance
  - Random / Irreducible error => You cannot do anything about it
- Too complex model (ver-fit) will reduce the bias but increase variance. vice versa. Find optimum complexity

Over-fit:
Hyper parameter tuning == C.

.co_ef

Logistic Regression on Python:

```
from sklearn.model_selection import cross_validate, train_test_split
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=0)
logreg = LogisticRegression(solver='liblinear')
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
y_proba = log_reg.predict_proba(X_test) #This will give out probability tuple (p of y=0, p
of y=1), not classification
```

For different decision boundaries:

```
#Step-1 => Obtain recall rate and precision rate for different threshold values
recalls=[]
precisions=[]
thresh_range =np.linspace(0,1,100)
for thresh in thresh_range:
    y_predt=[]
    for k in y_pred_proba:
        if k>thresh:
            y_predt.append(1)
        else:
            y_predt.append(0)
    y_predt = np.array(y_predt)
    recalls.append(recall_score(y_test, y_predt))
    precisions.append(precision_score(y_test, y_predt))


#Step-2 => Plot the recalls and precalls on a single plot to see where they both meet. That
point will be the optimum threshold value
plt.plot(thresh_range,recalls,label='Recall')
plt.plot(thresh_range,precisions,label='Precision')
plt.legend()
plt.xlabel('Threshold')
```

For Hyperparameter (C) tuning:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV


param_grid = {'C': np.logspace(-5, 8, 15)}
log_reg = LogisticRegression(solver='liblinear')
log_reg_cv = RandomizedSearchCV(log_reg,param_grid , cv=5, random_state=0)
logreg_cv.fit(X_train, y_train)
print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
# Take the C value from here and use it in Logistic Regression

LR = LogisticRegression(solver='liblinear', C=<>, random_state=0)
```

Softmax Regression: Also called Multinomial logistic regression. Used for multi-class classification.

NOTE: This is not multi-output classification. You cannot use it to recognise multiple people in one picture. It can only predict one class at a time.

Here the loss function is called as **Cross-entropy** function. The idea is to penalise the model when it gives low probability for target class.
Cross entropy function becomes the loss function of logistic regression function when K=2

```
from sklearn.linear_model import LogisticRegression
softmax_reg = LogisticRegression(multi_class='multinomial', solver='lbfgs')
softmax_reg.fit(X,y)
```

Applications of Classification Problems:

- Bankruptcy prediction.
-

DATA VISUALISATION and DASHBOARDING

Engaging the visual cortex of clients.
DATA Exploration is the first essential step of data-science. Descriptive statistics are not enough, you should know the distribution.
The potential of data visualization becomes extra-ordinary when it is also **interactive.**
https://seaborn.pydata.org/tutorial/distributions.html

Major Types of visualization:

- Frequency Distribution - Histograms, etc.
- Pie Chart
- Time Series
- GIS
- Comparison
- Deviation
- Ranking
- Correlation

LINE PLOT:

- **plt.plot(x,y)** => Line plot with 'x' on X axis | y on Y-Axis
- Plot multiple line plots if you want multiple lines in a single plot

- Plot multiple line plots if you want multiple lines in a single plot.

TIME-SERIES:

- First ensure that your date format is in Yr-month-date
  - **df['Date'] = pd.to_datetime(df['Date']) =>**Helps in this conversion
- Also set date as your data frame index. Hence it is a good practice to save your date and other variable as a separate dataframe.
  - **df2.set_index('Date', inplace=True)**
  - **df2.index = pd.to_datetime(time_revenue.index)**
- Then plot => df.plot() => Gives out a line plot. Basically .plot() is used to plot any two columns together.
  - For black dot plot, use **style='k.'** inside **plot()**

BAR CHART: One Axis=> Categorical Variable | Other Axis => Numeric- Can be discrete or continuous.

- **df.plot(kind = 'bar', label='')** => For plotting the dataframe as bar chart.
- **sns.countplot(C1)** => Shows a bar chart with count of C1 values.
  - **df.C1.value_counts().plot(kind='bar')** => Also shows the same. But seaborn has better visualisation.
- **plt.bar(x,y)**
- When you grouped by your data frame into multiple columns, you'll have to plot an aggregate column value of first group and also the second group within first group. To do this =>
  - **df_groupedby['C1'].mean().unstack().plot(kind='bar', cmap=)** => This will show each subgroup value within group
  - If you add **stacked=True** in **plot =>** Stacked plot.

For a bar chart with each bar in different color.

```
fig,ax = plt.subplots(figsize=(10,5))
sns.barplot(kiva_loans['sector'].value_counts(),kiva_loans['sector'].value_counts().index)
plt.xlabel('Number of Loans given')
plt.ylabel('Sector')
plt.show()
```

HISTOGRAM

Useful in understanding the frequency of the data. Works best when the data is discrete or categorical.

- **sns.distplot(C1, kde= ,)** => Shows how values in C1 is distributed as a histogram and PMF -- univariate distribution.
  - **df.C1.plot(kind='hist') =>**Also shows how values in C1 are distributed as a histogram. But seaborn has better visualisation.
- From histograms, you can also derive **PMF graphs => Probability Mass Function**
  - Divide Y-Axis by population size. Y-Axis will show the probability for finding a particular data point in the population data.
- Dis-advantages of PMF/ Histogram:

- Dis-advantages of PMF / Histogram:
  - When there are many discrete data points. OR simply put, when the spread of the data is high.
  - Not good with continuous data.
- **Cumulative Mass Function (CMF)**: Area under the PMF curve for a certain region.

When the data is continuous, you will sometimes need to see the histogram to observe distribution. In Tableau we use bins for this purpose. In Pandas, **pd.cut will help.**

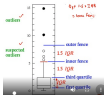
PROBABILITY DENSITY FUNCTION (PDF) GRAPHS: Good for continuous data

- Here probability is the area under the curve, not Y-Axis.
- Different PDFs:
  - Gaussian

NOTE => Bar chart vs Histogram = Know the difference.

BOX-PLOTS:

- Very helpful in **OUTLIER DETECTION.**



  - The line in the middle of the box-plot is the **median**.
  - Note that outliers could be on either side**.**
- **sns.boxplot(C1)** => Shows how values in C1 is distributed as a boxplot-- univariate distribution.
  - If you want to see this distribution in y axis => **sns.boxplot(y=C1)**
  - Box plot can show two variables across x and y axes. **sns.boxplot(x=C1, y=C2)** => It shows variation of y for each point on x.
  -
  -
- If you want to plot a box plot for all columns in a dataframe

```
df.boxplot(rot=90)
```

PIE CHART:

```
labels = list(kiva_loans['repayment_interval'].unique())
sizes = list(kiva_loans.repayment_interval.value_counts().values)

explode = (0.1, 0.1, 0.1, 0.1) #Explode the slice.
fig, ax = plt.subplots(figsize=(10,10))
kiva_loans['repayment_interval'].value_counts().plot(kind="pie",
                                                     labels=labels,
                                                     startangle=90,
                                                     autopct='%1.1f%%',
                                                     explode=explode,
                                                     ax=ax)
ax.get_yaxis().set_visible(False)
plt.tight_layout()
```

```
plt.show()
```

Interactive Pie-chart: Kiva_loans_advanced.

POLAR CHARTS

HEAT MAPS:

Using seaborn:

- **seaborn.heatmap(*data*, *vmin=None*, *vmax=None*, *cmap=None*, *center=None*, *robust=False*, *annot=None*, *fmt='.2g'*, *annot_kws=None*, *linewidths=0*, *linecolor='white'*, *cbar=True*, *cbar_kws=None*, *cbar_ax=None*, *square=False*, *xticklabels='auto'*, *yticklabels='auto'*, *mask=None*, *ax=None*, ***kwargs*)**
  - https://seaborn.pydata.org/generated/seaborn.heatmap.html
  -

SUBPLOTS

- **plt.subplots(*nrows=1*, *ncols=1*, *sharex=False*, *sharey=False*, *squeeze=True*, *subplot_kw=None*, *gridspec_kw=None*, ***fig_kw*)**
  - https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplots.html
  - This returns a figure object and a set of subplots or Axes (2D). You can assign a variable to them in order to manipulate them.

```
fig, axes = plt.subplots(3,2, figsize=(10,15), sharey=True)

df_g = titanic_df.groupby(['Pclass'])
df_g['Survived'].count().plot(ax=axes[0,0],kind='bar',color=['r','g','b'])
axes[0,0].set_xlabel('Passenger Class')
axes[0,0].tick_params(rotation=0)

df_g = titanic_df.groupby(['Sex'])
df_g['Survived'].count().plot(ax=axes[0,1],kind='bar',color=['r','g','b'])
axes[0,1].set_xlabel('Sex')
axes[0,1].tick_params(rotation=0)

df_g = titanic_df.groupby(['Embarked'])
df_g['Survived'].count().plot(ax=axes[1,0],kind='bar',color=['r','g','b'])
axes[1,0].set_xlabel('Embarked')
axes[1,0].tick_params(rotation=0)

df_g = titanic_df.groupby(['Age_cat'])
df_g['Survived'].count().plot(ax=axes[1,1],kind='bar',color=['r','g','b'])
axes[1,1].set_xlabel('Age')
```

```
axes[1,1].tick_params(rotation=0)

df_g = titanic_df.groupby(['Fare_cat'])
df_g['Survived'].count().plot(ax=axes[2,0],kind='bar',color=['r','g','b'])
axes[2,0].set_xlabel('Fare')
axes[2,0].tick_params(rotation=0)

fig.suptitle('Survival Rate vs Other features')
plt.show()
```

- **How to put a constant reference line in the plot?**
  - **ax1.axhline(y=0 , color='k')** => Constant y=0 horizontal line.

For colors:

- Using **plt.plot(c= , cmap =)**
  - c = b, g, r, c, m, y, k, w
  - For color maps, create a color map first and then write it inside the plot.
    - **cmap = matplotlib.colors.LinearSegmentedColormap.from_list("",['red, 'green' 'yellow'])** {This is a custom made cmap, there are many default ones available in library }
- Using **seaborn**
  - **cmap = sns.diverging_palette(220,10, as_cmap=True)**

- Marker styles.
  - plt.scatter(marker = , s)
    - s = size.
    -
- You can modify how you want to present ticks i.e labels on the Axis. :
  - plt.xticks(rotation =)
- Spines
  - ax = plt.gca() => Axis is isolated
  - ax.spines['left'].set_linewidth(3)
  - ax.spines['left'].set_color()
  - etc
- **Labelling:**
  - **plt1.set(xlabel = 'xxxx', ylabel = 'yyyy')** => Naming X and Y axes
  - plt.xlabel
  - plt.ylabel
  - **plt.title('Title', fontweight='bold', size= 18)** => Name title of plot | Note that you should not use plot variable for this.

- **plt.legend**
- **plt.xlim(10 , 30)**
- **plt.ylim(10 , 30)**
- For interactive matplotlib tool bar => use magic
  - % matplotlib notebook
- Annotation
  - plt.text()
  - plt.annotate()
- For 3-D Plots:
  - from mpl_toolkits.mplot3d import Axes3D
  - fig = plt.figure()
  - ax =Axes3D(fig)
  - ax.scatter(X.High,X.Low,y,color='k')

Manage your seaborn figure this way:

- **sns.set(context= , style= , palette= , font= , font_scale= , color_codes= T/F , rc=)**
  - **https://seaborn.pydata.org/generated/seaborn.set.html**

Saving the plot => **plt.savefig**('xxx.png', dpi=)

Gmplot

make_blobs

Dash boards:

- Streamlit
- Plotly.js

SCATTER PLOTS

- **plt.scatter(C1, C2)**
- **fit = np.poly.fit(x,y,1)** => To calculate **slope** and y-**intercept** of the fit line.
  - yfit = [n*fit[0] for n in x] + fit[1]

Pairplot is a very powerful tool to visualise the points in a 2-D feature space as a scatter plot. All the features can be seen in multiple plots.
https://seaborn.pydata.org/generated/seaborn.pairplot.html

```
import seaborn as sns
sns.pairplot(df,hue=,palette=,markers=[],vars=[],x_vars=[],y_vars=
[],kind='scatter'/'reg',plot_kws = {'alpha': 0.8, 's': 80, 'edgecolor': 'k'})
```

- Hue is like **Level of Detail (LOD)** in Tableau. Allows us to color different classes in different colors.

- 'reg' will draw a linear regression line accross the classes.

DECISION BOUNDARY PLOT

This function will help in plotting decision boundary for classifiers (clf) after training(.fit) them.

```
# clf is a classifier, X is a 2-D dataframe, Y is the target 1-D dataframe,
def plot_decision_boundary(clf, X, Y):
    h = 0.02
    lab = X.columns
    x_min, x_max = X.iloc[:,0].min() - 10*h, X.iloc[:,0].max() + 10*h
    y_min, y_max = X.iloc[:,1].min() - 10*h, X.iloc[:,1].max() + 10*h
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    fig, ax = plt.subplots(figsize=(7,7))
    #plt.contourf(xx, yy, Z, cmap=cmap, alpha=0.25)
    ax.contour(xx, yy, Z, colors='k', linewidths=0.7)
    scatter = ax.scatter(X.iloc[:,0], X.iloc[:,1], c=Y.iloc[:,0])
    legend1 = ax.legend(*scatter.legend_elements())
    plt.xlabel('{}'.format(lab[0]))
    plt.ylabel('{}'.format(lab[1]))
```

GEOGRAPHIC PLOT

```
import plotly.graph_objects as go



df222 = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master
/2011_us_ag_exports.csv')



fig = go.Figure(data=go.Choropleth(
    locations=state_df['state'], # Spatial coordinates
    z = state_df['age'].astype(float), # Data to be color-coded
    locationmode = 'USA-states', # set of locations match entries in `locations`
    colorscale = 'Reds',
    colorbar_title = "Police Killings",
))



fig.update_layout(
    title_text = '2015 Police Killings by State',
    geo_scope='usa', # limite map scope to USA
)
```

Annotation:

```
# To annotate in bar plots.
for i, v in enumerate(C1.values):
    plt.text(0.8,i,v,color='k',fontsize=19)
```

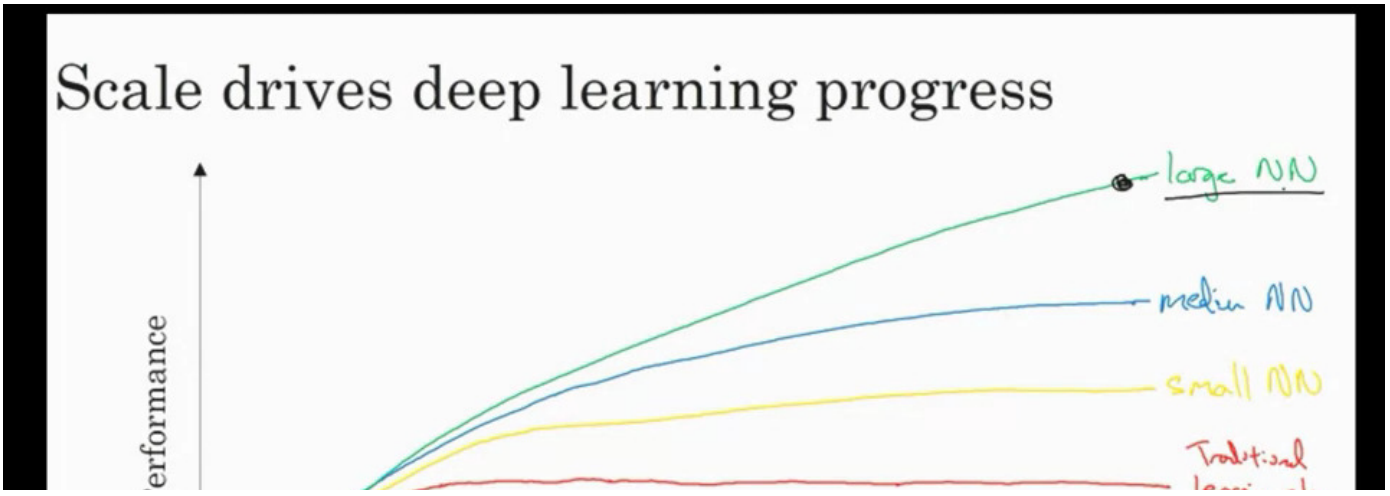
ANN = Artificial Neural Networks | Deep Learning

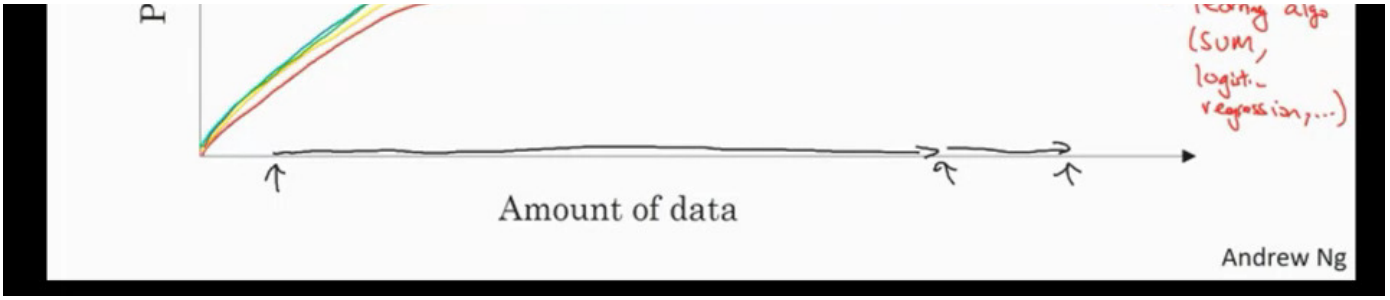
What is Deep Learning?
Deep Learning refers to training of Artificial Neural Network.
Well, what is an Artificial neural network?

Artificial Neural Network is an inspiration of the Human Brain. Our brains consist of **100 Billion neurons**; each neuron is linked to thousands of other neurons. Thus there is a very complex neural network in our brains making decisions every second. In the movie PK, Aamir Khan explains the different types of "Acha" in Hindi, which confuses him as an alien. But our brain is so comfortable in detecting these various types of "Acha's", only because our complex neural network was trained in understanding it, while the poor alien Aamir Khan's neural network was not trained. Similarly, Our brain is able to raise red flags when we see someone with a knife in public, because our neural network has been trained of potential violence possible. If we can replicate the same training of neural networks to help the computers learn from data, then that is called as **ARTIFICIAL NEURAL NETWORKS!**

De-Mystifying Neural Network:
Neural Network is nothing but a **function.** It takes in an input and gives out an output. For example, consider the simple uni-variate problem of **predicting housing prices based on size of the house.** A neuron would take size of the house as input (x). It implements a function and gives out the output f(x).

Why Choose Neural Networks over other Machine Learning Algorithms?



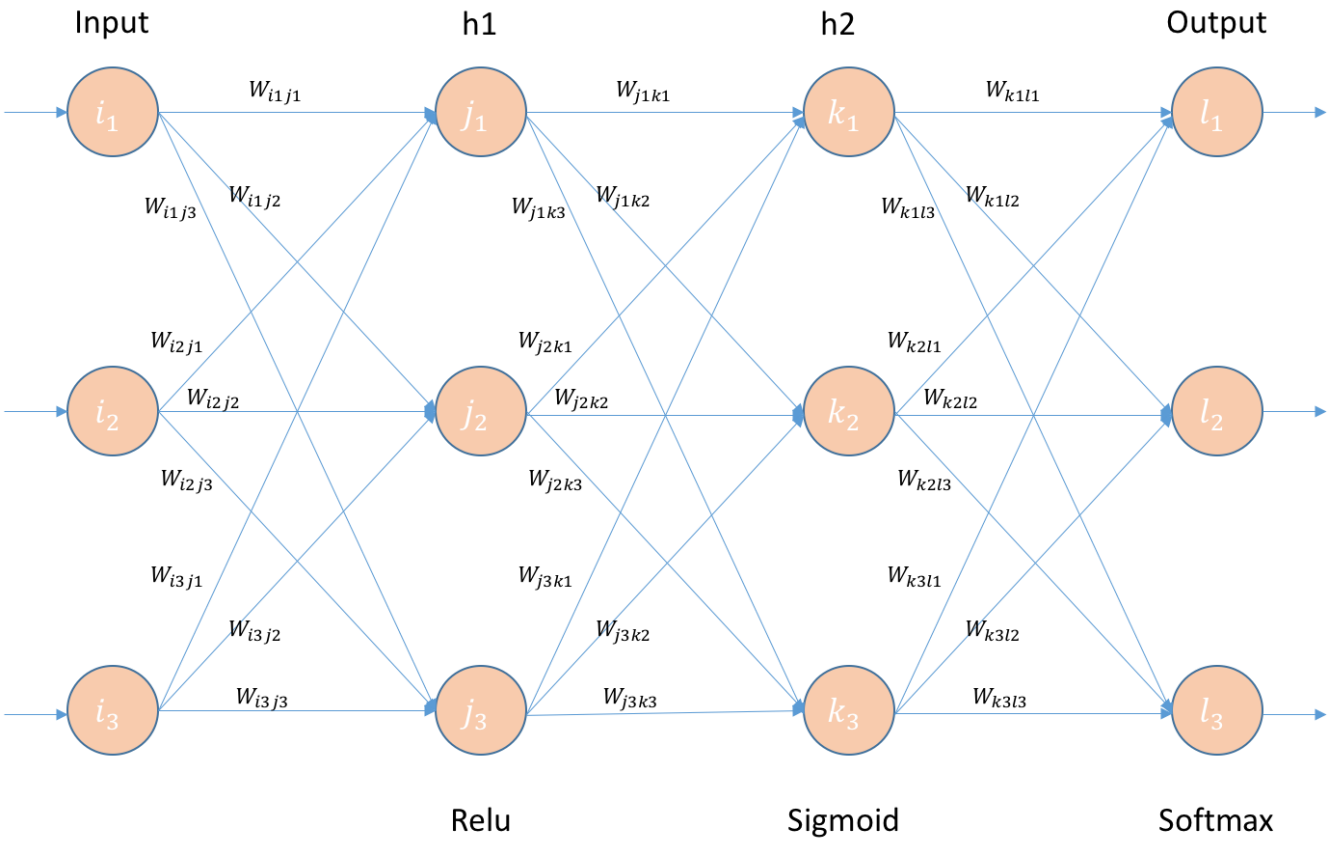


When the data is more, neural networks performance is immensely greater than the ML algorithms. But for small data applications, don't waste computing power with unnecessarily training ANNs. Just stick to ML instead.

In Computer Vision and other advanced AI applications, we deal with vast amounts of data. For instance, consider a simple example of **cat detector.** Here, the input (x) is the image and output is a **binary classification** (y= 0,1). Even if we say that the image resolution is 64*64 pixels, a single image be represented as a vector of dimension 12288 (64*64*3). This is huge data given that we have to train the model over many cat images. Thus, Deep Learning is essential for Computer Vision, advanced NLP etc.

What is a Deep Neural Network | Deep Learning?

When there are 2 or more hidden layers with each layer containing large number of neurons.



Note that input layer is only to provide input features. There is no computation taking place at this level.

What is training a neural network?:

Training means that, when you give a set of inputs(x) and outputs(y); the neural network learns **how to**

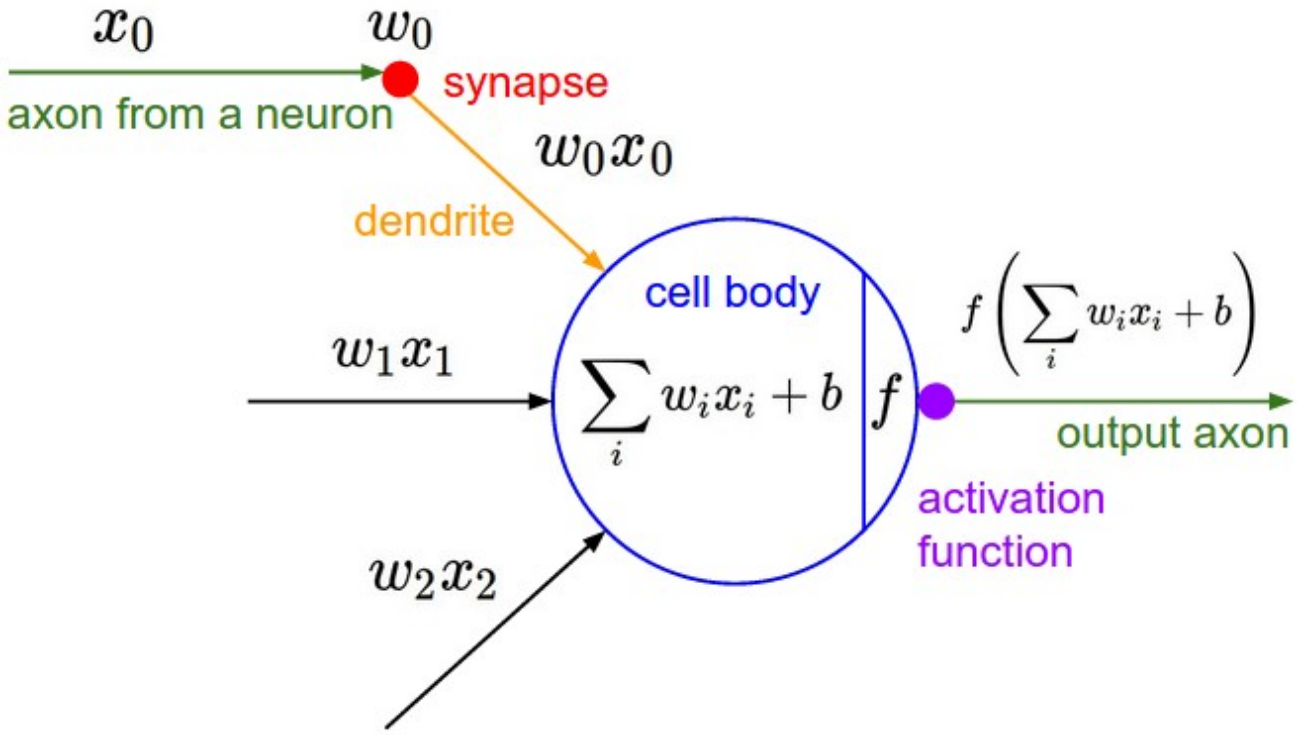
map the function f(x) between x and y. Considering the same housing prices prediction example, if you give a data on set of house sizes and its associated prices, training an ANN is to be able to define f(x) that maps sizes to prices.

Few important pre-requisites before we dwell more into Deep Learning:

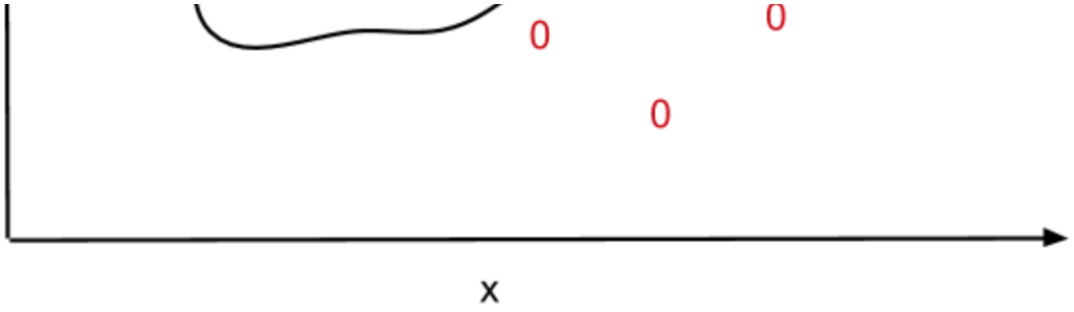
- **Vectorization** = Eliminate unnecessary for loops. In deep learning, we work with huge amounts of data. Non-vectorized implementation using for loops will take a lot of time. Thus the ability to write efficient vectorized code is a must in the deep-learning era.
    - Vectorisation takes help of **"Parallel computing"** and is thus fast.

What is an activation function?

- The neurons in the Neural Network should pass information from one layer to the another.
- If we pass the summation of [W.X + b] with out any transformation, the **network remains a Linear Regression.**



- But non-linearity is essential in various applications. For instance consider a binary classification as in the following figure.



A straight linear line will fail to classify this data. Thus we would require non-linear solutions to this problem. So how to add this non-linearity?

- **Activation Functions!** These are nothing but non-linear transformations at every neuron. Based on our use-case, we will choose our activation function.
  - Sigmoid
  - Tanh
  - ReLU = Rectified Linear Unit.
  - Leaky ReLU

Why is ReLU most preferred activation function?

Building Framework for a Neural Network:

1. **Design a Neural structure:** This will include-
   - Number of input neurons
   - Hidden layers
   - Number of neurons in hidden layers
   -
2. Initialise the parameters like weights, biases at each neuron. This can be done randomly.
3. Create a loop that will constantly improve the parameters.
   1. **Forward Propagation:**
      - Each layer of neurons forward compute.
      - Calculate the f(x) at the output neuron. This f(x) will not be the expected 'y' value initially because our initial weights were randomly selected.
      - Calculate the **loss function** to estimate the error between predicted value and expected value.
   2. **Back-Propagation:**
      - Calculate gradients/derivatives
      - http://neuralnetworksanddeeplearning.com/chap2.html
      - https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c
   3. **Update the parameters** using gradient descent optimization.

Architectures:

- Recurrent NN
- Convolutional NN

Implementation of NN:

- **ANN for Classification problem**

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import learning_curve
```

https://github.com/Frixoe/xor-neural-network/blob/master/XOR-Net-Notebook.ipynb

---

---

import Pandas as pd

Pandas is the library in python that allows you to work with structured data - tables. It creates a dataframe, with rows and columns.

Let **df** be a data frame with C1,C2... Columns or Vectors.

CREATING A DATA FRAME:

- **df =**pd.DataFrame(data=, columns=, index= , dtype=, copy=)
  - data can be list of lists, dictionary, list of series etc.
    - If dictionary, then keys become columns.
- df = **pd.read_csv**('xyz.csv', index_col= , header=, delimiter=)
- df = **pd.read_sql_query**('SQL Query', cnx) => Queries data from SQL database
  - First we will have to connect Python to SQL database. cnx variable does that:
    - cnx = sqlite3.connect('.....')
- df = **pd.readjson**(url/filepath, orient =)
- **See your data:**
  - **df.head()** => see first 5 | **df.tail()**
  - **df.sample(5) =>** any 5
  - **df.dtypes** => see data type of each column.
  - **df.shape** => Returns the shape of the dataframe as a tuple
  - **df.describe()** =>Returns **descriptive statistics** => count, mean, std, min, 25%, 50%, 75%, max of each column of the data frame.
    - Default percentile stat is 25%, 50% and 75%. But you can add more by **df.describe(percentiles = [0.25,0.50,0.75,0.85,0.90,1])**
    - Use include='all' if you want descriptive stats for categorical variables also. Otherwise only numeric variables get described.

        - **pd.set_option('display.max_columns', None)** => use this when you have many columns.
    - **df.info()** => Returns the data-types of each column of data frame, memory used etc.
    - pandas profiling

INDEXING AND SLICING:

- **df.loc[[x1,x2,x3] , [y1,y2,y3]] =>** Returns partial data frame with selected rows (x1,x2,x3) and columns (y1,y2,y3)
    - This is used to filter through booleans.
        - Use & and | if needed.
- **df.iloc[[1,2,3], [1,2]] =>** Returns the 2nd,3rd,4th rows and 2nd,3rd columns as a dataframe.
    - This is used to filter through booleans.
- **df.columns** => Returns a list of column names of the data frame. You can use **[0/1/..]** to select particular column name.
    - You can use **df.columns** to rename column names by assignment. But you'll have to rename all the column names if you use this approach. Instead, there is a better approach:
        - **df.rename(columns = {'C1' : 'D1', 'C2' : 'D2'}, inplace = True)** => Returns a data frame after renaming C1,C2 to D1,D2 etc.
    - **df.C1 =>** Returns a Series of C1 in data frame | You can use [0/1/..] to select particular key in C1.
        - You can either do it as **df[['C1']]**
    - **df.C1.values** => Returns a list of values in C1 | You can use [0/1/..] to select particular key in C1.
    - **df.C1.value_counts()** => Returns a data frame with a count of each key repeated in the Column C1
    - df.C1.count() => Returns total number of non-null values.
    - **df.C1.unique()** => Returns an array of unique keys in C1.
    - **df.C1.nunique() =>** Returns number of unique keys in C1
- If you want to change index:
    - **df.set_index**([C1], inplace= , append, drop)
    - **df.reset_index**(inplace=, drop=True)
        - drop = True is essential sometimes, other wise reset_index throws an error.
    - **df.index** = index_column
        - Helps in giving names to rows just like columns have names.

FILTER:

- In df.loc use booleans.

METHODS:

- **df.C1.replace(x,y,inplace = True) =>** Replaces all 'x' with 'y' in the Column C1.
- **df.insert(column_position, 'D1', D1) =>** Inserts a new column D1 in the dataframe.
- **df.assign(NewCol=default_value)**

- **df.drop(['column_name(s)'], axis=0/1, inplace=True)** => Drops the columns from the data frame
- **df.duplicated()** => Helpful in finding duplicates. Returns True or False
- **df.sort_values**(by=[], ascending=[], inplace=)
- Working will Null Values => Important for data cleaning:
  - If you want to write a null value while creating a dataframe => **np.NaN**
  - **df.isnull()** => Applies **isnull?** operation of each element of the df and returns a boolean dataframe
    - You can sum() it to know number of NaNs in each column
    - Alternatively, you can use **df.notna()**
    - Use it within sns.heatmap for a visualisation fo missing value.
      - **sns.heatmap**(df.isnull(), yticklabels=False, cbar = False, cmap = '')
  - **df.dropna(axis= , inplace=, thresh=)** => if there is NaN in any row, that entire row is removed.
    - If you want to remove column instead, use **axis=1**
    - **thresh** will be used if you want to remove a row if it has less than 'thresh' non-null values.
  - But if you cannot delete the rows that have NaNs; you should intelligently fill it:=> **df.fillna( )**
    - Use Mode of the column if the column is a categorical data
    - Use Mean to fill
    - Use parameters like:
      - **df.fillna(method = 'ffill'/'bfill'/pad, limit= ,)**
    - Sometimes you will have to fill the null values not with a single value, but multiple. Eg: say there are multiple modes, you should use all modes to fill nulls. Then take all modes into a list-lst, then use **random.choice(lst)** to return one mode randomly. Use this inside fillna()
  - K Nearest Neighbors (KNN) to fill missing values.
  - Build an ML Model to predict missing values.
- df.astype('category'/'numeric')

FUNCTIONS: You can apply a function on keys of the dataframe. **df.apply(func) | df.C1.apply(func)**

- **df.describe()**
- **df.sum() =>** Returns sum of each column of the df as a **Series**
- **df.mode()**
- **df.mean(axis=)**
- **pd.cut(df.C1, [0,15,30,40], labels=['a','b','c']) =>** Creates bins / groups of the continous variables in C1. Often you save it as another column in df. Helpful as you can't plot bar graphs with continous data, convert it into bins
- pd.qcut
- **pd.to_datetime(df.C1)**
  - Converts the C1 column into datetime object. Helps in timeseries plotting. Otherwise, python will not recognise dates.
  - Other wise, while read_csv only, there are certain parameters to ensure that pandas reads datetime format.

    - parse_dates = ['C1']
    - infer_datetime_format = True.
- **df.C1.dt.year** => Returns only year from the date time object
- **df.C1.dt.month** => Returns only month from the date time object
- **df.C1.dt.week** => Returns only week from the date time object
- **df.C1.dt.day** => Returns only day from the date time object
- pd.melt() => Pivot
- Use **swifter** package to increase the speed in which you want to apply functions on your pandas dataframe.

GROUPBY:

- **df.groupby([C1,C2..]) =>**Creates data frame after grouping values in C1 and then C2 .....
- Only groupby is a waste, you have to work on the grouped dataframe. So first save **df_g = df.groupby([C1,C2..])**
    - **df_g.mean() =>**Returns mean of each group for each column. You can also select particular column by **df_g[C3].mean()**
    - **.sum()**
- **df.unstack()** => To unstack columns
- **df.stack()** => To stakc
- **pd.crosstab(C1, C2, margins=T/F, normalize='index'/'columns'/'all')**
    - Normalize helps in expressing in % terms;
        - 'index' will help in normlaising by rows
    - margins=True => Shows Total columns as well.
- **pd.pivot_table(df , values=['C1','C2'] index=[], columns=[], aggfunc={'C1':np.mean,'C2':np.sum } , fill_value=0)**
    - **df =>** dataframe in question
    - values => data that you want to put in the pivot table.
    - index => Row(s) you want. It can be hierarchy also
    - aggfunc => How you want df to be in the pivot? np.sum ?

MERGING DATA FRAMES:

- **df.append(df2, ignore_index =)** => Returns a new dataframe after adding **rows** of df2 to df.
    - This is ideal when df2 and df have same features.
- **pd.concat([df1,df2], ignore_index=, keys= , axis=0/1) =>** Returns a new dataframe after adding **rows** of df2 to df
    - This is ideal when df1 and df2 have same features.
    - **Keys** helps to identify each of the data frame in future.
- **pd.merge(df1, df2, left_on =**left_table_key **, right_on =**right_table_key **, how=**left/right/outer/inner**)** => Like JOINS in SQL. To join dataframes based on a column.

Bamboolib

Bamboolib

Q.) How to add a row at a certain index in Pandas?
Unfortunately, there is no in-built way to do this. We have to write a function to get this done.

STYLING DATA FRAMES:
https://towardsdatascience.com/style-pandas-dataframe-like-a-master-6b02bf6468b0

Data Preparation.

1. How to extract tables from PDFs? Using camelot
2.

OPTIMIZATION

GRADIENT DESCENT

The core of Gradient descent is minimizing the cost-function (J) of various ML algorithms.

repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

j=0,1 | alpha = step size indicating speed of the descent | Derivative gives out slope of the tangent[

] at that point making the descent along the slope. Hence convergence here is when the derivative becomes negligibly small or 0, indicating that J reached it's minimum.This is like climbing down a hill.
NOTE-1: The initial value of parameters that you chose for applying Gradient Descent plays an important role. It can so happen that the Gradient descent takes you to 'local minimum' of J instead of 'global minimum'. Keep this in mind. Of course, for simple linear regression there is no chance of multiple local minimum. This situation is for multi-variable regression.
NOTE-2: If the step size is large, there is a danger of over-shooting and the descent never converges.

In **multivariate linear regressions,** the cost function changes as it includes multiple variables. Other than there is no other change. But for quicker and effective working of gradient descent, it is nice if the features i.e the different independent variables are on a similar scale. This is called **Feature Scaling.** We often get every variable between -1 and 1. Alternatively, **Mean Normalisation** is also one method of feature scaling. Moreover step-wise regression is a good practice. Do not take all independent variables

feature scaling. Moreover step-wise regression is a good practice. Do not take all independent variables at once. Start with the largest predictor, and keep adding one by one. Calculate Adjusted R-Square at each step.

When you use Gradient Descent optimisation, ensure that all the features are in similar scale. Otherwise, it will take longer time to converge as the cost function will be of elongated shape.

```
# Feature Scaling - Standardisation by removing mean and scaling to unit variance i.e
Standard normal!
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler(copy=True, with_mean = True, with_std=True) #Mean is taken as 0 if
False, std as 1 if False.

df_scaled = scaler.fit_transform(df) #Fit to data and then transform
df_scaled = scaler.fit(df) #Computes mean and std to be used for future scaling.
df_scaled = scaler.inverse_transform(df) #Scale back the data to original format
df_scaled = scaler.transform(df) #Perform transformation by centering and scaling.
```

NLP

Natural Language Processing:

Use-Cases:

- Automated gmail replies
- Chatbots
- Search Suggestions

Main approaches in NLP:

1. **Rule-based Methods:**
    1. **Regular Expressions (RegEx) =>** Here, we try to find **search patterns** in a string. The "Find and Replace" option in MS Word is a application of RegEx.
    2. Context free grammar
2. **Probabilistic Modeling and Machine Learning:**
    1. **Likelihood maximisation**
    2. **Linear Classifier**
3. **Deep Learning**
    1. **RNN**
    2. **CNN**

Opening text files:

```
import urllib.request
urllib.request.urlretrieve("url.txt", "name.txt")

with open("name.txt", 'r') as txt:  #When you use with function, you need not close.
    text = txt.read()
```

NLTK= Natural Language Toolkit => One of the best libraries.

```
import nltk
nltk.download('all') #Do this once to download all packages
```

TEXT UNDERSTANDING:

Text has huge ambiguity.

- Same word - different meanings: For instance, orange can be both color or fruit.
- Different words - same pronouncement: By, Buy, Bye
- Immense diversity in dialects, slangs also cause this ambiguity.
- Scope Ambiguity:
- Clause attachment
- **SCIENCE OF LANGUAGE:**
  - **Phonetics**
  - **Morphology**
  - **Lexical Analysis**
  - **Syntactic Analysis**
  - **Semantic Analysis**
  - **Pragmatics**
  - **Discourse**
- **Parsing**

- **Tokenisation:**
  - str.split()

```
tokens = nltk.tokenize.word_tokenize(string) #This will return a list of every distinct word
(including ,)

from nltk.corpus import RegexpTokenizer as regextoken  #Tokenise based on regex
tokenizer = regextoken(r'\w+')
tokens = tokenizer.string()

from nltk import sent_tokenize #Sentence Tokenizer
sents = sent_tokenize(string)
```

- **Stemming**

```
from nltk import SnowballStemmer
from nltk import PorterStemmer
```

```
from nltk import LancasterStemmer

snowball = SnowballStemmer(language='english')
porter = PorterStemmer()
lancaster = LancasterStemmer()

[snowball.stem(token) for token in tokens]
```

- **Lemmatisation**

```
from nltk import WorkNetLemmatizer
lem = WordNetLemmatizer()
[lem.lemmatize(t) for t in words]
```

- **POS Tagging**

```
from nltk import pos_tag
pos_tag(tokens, tagset='universal')
```

- **Stopwords = useless | But they are needed when you are doing sentiment analysis / context**

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
filtered_sentence = [w for w in tokens if w not in stop_words]
```

```
#SpaCy is powerful library that helps in parsing
conda install -c conda-forge spacy
python -m spacy download en #Run this on cmd


import spacy
nlp = spacy.load('en_core_web_sm') #Loading language


doc = nlp(text) #Create a document object
doc.text[ : 50]
list(doc.sents) #Lists of different sentences in the doc
list(doc) # Tokens
```

```
from nltk.corpus import wordnet as wn
```

---

https://techcommunity.microsoft.com/t5/AI-Customer-Engineering-Team/Speeding-up-the-Development-of-Natural-Language-Processing/ba-p/1042577

Sentiment Analysis

Analyse the text and tell if it is negative, positive or neutral. Apart from that we can also give three attributes to each text:

- Polarity: The degree of positive or negative opinion
- Subject: the thing that is being talked about.
- Opinion holder: The user that is expressing the opinion

Types:

1. **Aspect based Sentiment Analysis:** Say the review is "battery life of the camera is too short". This review is negative about the camera, but more precisely about the battery. If you can provide such level of detail, it will be more helpful for the businesses.
2. **Intent Analysis:** See what is the actual intent conveyed in the message.

- **https://monkeylearn.com/sentiment-analysis/**
- **https://devopedia.org/sentiment-analysis**
- Pitfalls:
  - Anna Hathaway news increased stock price of Hathaway company :p

https://lnkd.in/e_qWXct

Time Series Forecasting

Time series forecasting is different from time series analysis. In forecasting, we predict the future based on historical data. In analysis, we analyse the historical data.

How is time series forecasting different from other machine learning models? ML models treat all the historical data equally to make a prediction about the future. But Time Series Forecasting adds the time dimension to the data set.

Basic Terminology: Important key-words that we will use in Time Series Analysis and Forecasting

1. **Lag:** Number of time-units (days/weeks/months) required to predict the outcome at the present time-unit. For example, if previous 3 days of sales data is used to predict sales today, then Lag=3 days.
2. **Stationarity in Time Series:** A stationary time series is something in which the **way** in which the timeseries changes does not change over time.

It has **constant mean and constant Standard Deviation**. It is the **most common assumption** in

. It has **constant mean and constant Standard Deviation.** It is the **most common assumption** in Time Series Analysis. **ADFULLER STATISTICAL TEST IS NEEDED FOR IT**

3. **Differencing:** Making a non-stationary time-series into a stationary one. Compute the difference between consecutive observations. This difference can be stationary. Basically, Differencing removes TREND.

Basic Python for Time Series Analysis / Forecasting:

```
from datetime import datetime #This package is necessary if you want to work with dates.
my_date = datetime(1996,2,13,15,50,30) #year/month/day/hour/minute/second. Default=0, if not
specified. This will return a datetime object
my_date.day
my_date.hour


#You can save an array of dates using numpy
np.array(['1996-02-13','1972-02-09','2006-12-15'], dtype='datetime64[M]') #Here [M] is the
level of detail, you can leave it for computer to pick it up automatically.


#You can also create date ranges
np.arange('2020-01-01', '2020-01-15', 1, dtype='datetime64[D]')
np.arange('2000', '2020', 2, dtype='datetime64[Y]')


#Often you'll get dates in pandas dataframes as text. You'll have to convert it into
DatetimeIndex object to make any time series analysis or forecasting
pd.to_datetime(df) #Converts the df column(s) into datetime object
pd.read_csv( 'abc.csv', parse_dates=['C1'], infer_datetime_format=True) #Converts the C1
column into datetime object
pd.date_range('2/10/2019', periods=7, freq='D') #Creates a pandas dataframe with datetimes


#Finally
df.set_index('Date')
```

- **df.C1.dt.year** => Returns only year from the date time object
- **df.C1.dt.month** => Returns only month from the date time object
- **df.C1.dt.week** => Returns only week from the date time object
- **df.C1.dt.day** => Returns only day from the date time object

---

PLOTTING TIME SERIES:

- **For Histograms to see how the distribution of dates:**

```
df.date.hist(bins= , color=)
```

Dealing with missing values: Imputation with mean, median, mode simply doesn't work here. This is time series- you'l have to consider the trend and seasonality.

AR Model: Auto-Regressive model. The value at the present time instance is determined through a regressive model of values at previous time instances. However, you don't take all the previous time instances and give weights to them. You use **PACF (Partial Autocorrelation Function)** to determine which all previous time instances are to be considered in the AR model equation.

But why are we using PACF and not ACF (Auto-Correlation) ?
If you want to predict the price of gold today(Gt) based on price at time instance (t-2), then ACF gives both direct and in-direct correlation between Gt and G(t-2). But PACF gives only the direct affect of G(t-2) on Gt. Thus PACF is needed.

PACF also is the **co-efficient** that we can directly use in the AR model.

MA Model:

Auto Correlation Function will help here. (Why?)

ARIMA: Auto regressive Integrated Moving Averages

- It is applied over the stationary time series without seasonality.
- It has three parameters. (p,d,q)
  - d: **Difference Order:** Number of difference operations required to convert the timeseries into stationary.
  - p: **Order of AR:**
  - q: **Order of MA:** Lag of error component.

SARIMA:

ARIMA does not work well with seasonality in data. Thus comes the Seasonal ARIMA.
Apart from (p,d,q). SARIMA also needs (P,D,Q,M)
P,D,Q => p,d,q for seasonal part of time series.
M => Number of periods in each season.

Statistical Tests:

Augmented Dickey Fuller unit root test: For- To test assumption of stationarity

```
from statsmodels.tsa.stattools import adfuller
test_result = adfuller(df['<Time series values>'])
```

```
#H0 = It is not stationary
#H1 = Stationary.

labels = ['ADF test statistic','p-value','#Lags used','Number of Observations used']

for value, label in zip(test_result,labels):
    print(label+' : '+str(value))
if test_result[1] <= 0.05:
    print('Reject Null Hypothesis. Data is stationary')
else:
    print('Failed to reject null hypothesis. Data is non stationary')
```

Basic Python

- Want to know the data type of the variable (var) you have? => Use **type(**var**)**
  - You can change this to reduce the size of your datasets. Eg: Age can be stored as int16

INTs, LISTs, TUPLEs and ARRAYs

- **List vs tuple?**
  - Tuples are immutable.
  - You can concatenate two different lists, not possible with tuples.
- **List and Array?**
  - Arrays can store elements of same data type.

INTs and FLOATs

- Convert **float** into integer by=> **int(**5.9) = 5 => Basically it is like **FLOOR()** in SQL.
  - Alternatively => **math.floor**(5.9) = 5 => Returns the one lesser integer in the number line
- **math.ceil**(5.9) = 6 => Similar to **CEIL()** in SQL => Returns the one higher integer in the number line
- **round**(3.4) = 3 | **round**(3.7) = 4 => Similar to **ROUND()** in SQL. Returns nearest integer value
- **math.trunc**(3.9) = 3 | **math.trunc**(-3.9) = 3 => Returns only the integer part of the decimal.

LIST

Say lst = ['xxx','yyy','zzz',....]

INDEXING and SLICING:

- **lst[1] =>** Returns 2nd element of the list. Cuz indexing starts with 0
- **lst[-1] =>** Returns last element | This is called negative indexing
- **lst**[2:7] => Returns a sliced list from 3rd element to 6th element, 7 not inclusive.
  - If there is no number at one side of the : => Default is either 0 or last index, inclusive
- **lst**[2:7:2] => Returns a sliced list with index step size 2 from 3rd to 6th element.

OPERATIONS:

- lst + lst1 **=>** Returns a new list after concatenating both the lists
- lst * 3 => Returns a new list by repeating elements in lst thrice.
- element **in** lst => Returns True or False if element is in the lst or not.
- element **not in** lst => opposite to the above operator.
- How to replace an element in the list? You can simply over-write! Slice it and assign new value(s).

LIST COMPREHENSION: Alternative way of writing a for loop.

```
[k for k in lst if condition]
```

FUNCTIONS:

- **len(**lst**) =>** Returns number of elements in the list.
- **sorted**(lst) => Returns the lst after sorting it in ascending order. Sorted function works with any data type. Note that it does not modify the given list, only returns a new list.
  - Use argument, **key = function,** to sort by a particular key in the lst. Often used when the lst is a list of tuples/lists/dictionaries.
    - function should take each element of the list as a single argument and return the key that is used for sorting purposes. Use **lambda** as this is a temporary function.
    - As the above method of writing a key is complex, Python has an inbuilt function to do this.
      - from **operator** import **itemgetter**
      - **key = itemgetter(**index_to_be_sorted**)**
  - Use argument, **reverse = True,** for descending order.
- **min**(lst)
- **max**(lst)
- **sum**(lst)
- **set**(lst) => Returns a set with unique elements in the list.
- **list**(s/tup) **=>** Returns s into a list.
  - **list(range**(10)) => Returns a list of 0 to 9. You can also add start/stop values and step size in range function. **range(1,10,2)**

METHODS on Lists: Say **lst** = ['','',''...] => [lst**.**+ TAB for all list of methods available] | Note that METHODS are not Functions, they often return **None,** unless specified.

- lst**.insert**(index, element) => Inserts element at the particular index in the list.
- lst.**index(**element**) =>** Returns the index of the the element.
- lst.**append**(element) => Adds the element, as it is, at the end of the list.
  - or just add lst + lst2
- lst.**extend(**element**) =>** Each character of the element is appended at the end of the list.
  - If element is a list, then each element of the list gets appended to the original list.
- lst.**sort() =>** Sorts the list in ascending order.
  - Use argument, **reverse=True**, for descending order.
  - Use argument, **key=Function ... same like in sorted() function.**

- lst.**pop() =>** Removes the last element of the list and also returns it.
  - The default is the last element, you can also select the index instead.
- lst.**remove(**element**) =>** Removes an element by the value, while .pop removes by the index. However, it doesn't return it.
- .pop() and .remove() can remove only one element from the list. How will you delete multiple elements from the list?
  - **del lst[**index_range**];** Eg **del** lst[2:7]
- lst.**reverse() =>**Just changes the order of the list in reverse direction.
- lst.**count**(element) => Returns the number of times element is repeated in the list.
- lst.**copy**() => Returns a shallow copy.
  - You can also copy by using, lst2 = lst. But in this process, any change you make to the lst2 will happen to the lst as well! So use .copy

---

TUPLE

Say tup = (xxx,yyy,zzz.....)

Tuples are not much used in Data Science.

FUNCTIONS: Almost same as in list.

STRING

Say string = 'jsfksjb'

Ways of writing a string:

- If you want to use single quote (') inside a string, use double quotes (") for the string. Vice Versa.
  - "Sai's" or 'He said, "Get Lost", and left'
- """kwbkvbkjv""" => Muti-line comments
- Escape sequences in strings
  - For new lines in a string use **\n**
  - For tab space in a string use **\t**
  - But some times, like in file path etc., we will have to suppress the escape sequences. We use 'r'
- F-strings => f"My name is {name} and I work as {job}" => Imputes name and job variables in the string.

INDEXING AND SLICING:

- Just treat string as a list of characters. No more changes.

OPERATIONS:

- string1 + string2 => returns a Concatenated string.
- sting * 3=> returns a thrice repeated string

FUNCTIONS:

- **len**(string) => Returns number of characters in the string.
- **list**(string) => Returns a list with each character as the element of list.

METHODS: So many methods are there to modify strings. very useful for data cleaning.

- string.**capitalize()** => Returns a string after capitalising the first word and keeping all other characters in lower.
- string.**upper**() => Returns a string after capitalising all the characters of the string.
- string.**center(**10,'_'**) =>** Returns a new string of length 10 with original string at the center padded on both sides by '_'
- string.**split('x')** => Returns a list after splitting str at x. x will not be included. x can either be space.
  - You can also specify the length of the resulting list as an optional argument. The string will be split only those many times accordingly. string.**split('x',4) {splits into 3 items donno why, for 1 item use 0}**
- ' '.**join(lst) =>** Returns a string after joining the elements of the list with connector **' '**
  - Any other connector string can be used. string, '_', 'haha' anything.
- string**.index('x') =>**Returns the index of first occurence of 'x' in string.
  - You can also specify the indices within which you want to search. **string.index('xx',2,8)** would return index of substring 'xx' if it is there in between 3rd and 8th character.
  - You can also use the method string**.find()** similarly. But it can also return if there is no sub-string in the given string; while string**.index()** would return an error.
- string.**replace**('x' ,'y') => Returns a string in which all 'x' in original string is replaced with 'y'
  - You can also add an optional argument ('x', 'y', 1), if you want to specify number of 'x' you want to replace with 'y'
  - **NOTE:** Strings are immutable through assignment (=)
- string.**strip()** => Returns a string after clearing the trailing spaces on both sides. => Data cleaning.
  - **rstrip**()
  - **lstrip**()
- string.**isalpha**() => Returns False if there is any non-alphabetic character in the string, including spaces. => Data cleaning
- string.**endswith**('xx') => Returns True or False
- string.**startswith**('zz') => Returns True or False.

STRING FORMATTING: Within the strings, you can use certain expressions that can be referred to other variables. When you print such strings, the referred variables are printed in place of those expressions.

- **%d** = to refer to an integer variable.
- **%s** = to refer to a string variable.
- **%f =** to refer to float variable.
- Eg: formatted_string = 'My name is %s and my age is %d' %(name_string, age_int)

PLAYING WITH PRINT

DICTIONARIES

Say d = {'a' : 'Apple', 'b':'Ball', ...}

Here, you can set the index! In dictionaries it is called a **key**. Each **(key, value) tuple pair is** called an

There, you can set the index. In dictionaries it is called a **key.** Each **(key, value) tuple pair is** called an **item**

OPERATIONS:

- Dictionaries are mutable. Just assign whatever you want.

METHODS:

- d.**get**('a') => Returns the value present in key 'a'
  - Or you can directly call it as **d['a'].** Then why .get()? Because when you directly call it and there is no that particular key, Python returns an error. But in .get(), it returns None, or you can also choose what you want to return by passing another argument.
    - d.**get**('z', "This Key is not there")
- d.**items()** => Returns a list of items in d.
- d.**keys()** => Returns a list of keys of d.
- d.**values()** => Returns a list of values of d.
- d.**pop('a') =>** Removes and returns that particular key-value pair.
- d.**clear() =>** Clears the entire dictionary.
- d.**update**(d2) => Adds new key-value pairs from d2 to d.

FUNCTIONS:

- **len**(d)
- **max**(d) **=>** Returns max key, not value.
- **min**(d)
- **zip**(lst1,lst2) => Takes two lists and matches each element to create a dictionary. But you have to use **dict(zip(**lst1,lst2**))** to express it like a dictionary. Otherwise it remains like zip data type.
  - Zip function allows you to iterate over the dictionary. Check loops.

DICTIONARY COMPREHENSION: { x:y for }

LOOPS

Iterables:

- List
  - String acts as list of characters.
  - range
- Tuple
- Dictionary

How to iterate over a list?

- for k in lst:

How to iterate over a list of tuples? i.e even iterate over the elements of the tuple

- for (k,v) in lst: => Then you will have access to both elements of the tuple.

How to iterate over a dictionary?

- for (k,v) in d.items(): => selects each key-value pair as a tuple.
- for (k,v) in zip(d.keys(), d.values()) => selects each key-value pair as a tuple.

IF | ELIF | ELSE: Returns booleans, you can write conditions accordingly.

- Nested loops.

WHILE | ELSE: The loop continues as long as the condition returns True. **Danger of infinite loop**

FOR | ELSE:

for <target> in <iterable>:
  <actions with target>

FUNCTIONS that helps us in efficient loops:

- **enumerate(**lst**)** => Adds a counter to the <target> in the <iterable>. Helps in counting number of iterations.
  - Basically it returns a list of tuples (<index>, <target in lst or any other iterable>), but as a datatype of enumerator. You can however use lit like a list in for loop.
    - for k,v in enumerate(lst)
  - You can also make the index start from any other number by passing additonal argument => enumerate(lst,100)

break: to break the loop.
continue: skip that iteration.

FUNCTIONS

To make the code modular, repeatable.

LAMBDA: Single line nameless function.

```
(lambda x,y : x*y) (3,4) #Here x and y are arguments of the function. Whatever after : is
return value. Pass in the argument right after the lambda function.
#Good practice = enclose lambda inside paranthesis.
```

MAP: When you want to apply a function on every element of an iterable. It returns a map object, which has to be converted into list.

```
map_output = list(map((lambda x:x**2), [1,2,3,4])) # Basically after writing function, write
the iterable on which you want to apply your function.

#When there are two or more arguments in the function - send two arguments as separate
lists.
map_output = list(map((lambda x,y:x*y), [1,2,3,4],[9,8,7,6]))
```

FILTER: Works like map, but it is used when you want to filter an iterable using a function. However, unlike map - it takes only one iterable which has to be filtered.

```
to_be_filtered_list = [1,2,3,4,5,6,7,8]
even_filter = list(filter((lambda x:x%2==0), to_be_filtered_list)) #Filters all Trues.
```

REDUCE: Works same like the map, but this is for rolling computation in an iterable.

```
from functools import reduce
product = reduce((lambda x,y:x*y),[2,3,4,5]) #Output will be 120
#Rolling product takes place here=> 2*3 =6; then this product is stored is multiplied with
4; and so on until the iterable is complete.
# Rolling computation can be done using if conditions as well.
```

MISCELLANEOUS

Exceptions: Helps in not breaking the code

```
try:
    <Whatever you want to do>
except <Error_Name>:
    <What to when this error occurs?>
```

Regular Expression: Like Wild Card search in SQL. https://www.w3schools.com/python/python_regex.asp#findall

```
import re
pattern = 'a...e'  #Starts with a, ends with e, 5 char length. Use brackets for (\s) (\.)
test_string = 'apple'
result = re.match(pattern, test_string) #Result will be True.
#This result (after .match and .search ) has various methods as well
    result.span() => returns start and stop of the first matched string
    result.group() => returns the first matched string.
#If you want to return the matched string instead, use .findall() to return a list of
matched substrings in the string
pattern2 = '\d+' #This pattern will search for numbers.
string2 = 'My age is 23 and my height is 183 cm'
result2 = re.findall(pattern2,string2) # This will return ['23', '183']
for match in re.finditer(pattern2,string2):
    print(math.span())  # will print indices of both 23 and 183
```

- **re.split()**
- **re.sub()**
- **re.search()**
- **Regex expressions: use them in the pattern**
  - **^a =>** To search for a string that starts with a
  - **$e =>**To search for a string that ends with e
  - **\s**
  - **\.**
  - *** =>** Any number of characters = 0 or more.
  - *? => Any number of characters (non-greedy)
  - + => One or more.

- (|||) => Multiple options.

Ensembling

ENSEMBLE LEARNING

Ensemble Learning believes in the "wisdom of crowd" rather than the individual expert's opinion. That is, we train a group of predictors instead of choosing only one. The aggregate prediction of this group of predictors will be our prediction.

Ensembling can be done in two ways = hard voting and soft voting.

- **Hard:** Suppose there are 10 different prediction models in an ensemble. You give a new data point for classification to the ensemble. 6 of them predict class1, while the other predicts class0. Then by hard voting, the ensemble would predict class 1.
- **Soft:** Here voting is not done mere on the prediction outcomes of each model. It may so happen that the 6 class1 predictions were of low prediciton probability (predict_proba), while the 4 class0 predictions are of higher prediction probability. Hard voting misses this nuance. Soft voting thus considers not the prediction outcomes of models but the prediction probabilities of the models, averages them and then predicts the outcome. **This is more accurate than the Hard voting method.**
  - But soft voting works only when all the models in the ensemble makes classification based on prediction probabilities. (Obviously right). Hence, when you use SVC, you should set its probability hyperparameter to **'True'**

Ensembling is often done at the last stages of machine learning projects and is very important to win your client. Even if all your individual predictors are not of great accuracy, ensembling can increase the performance immensely. This is because of **law of large numbers.** But ensembling works best when the individual predictors are independent of each other. But here they work on the same data and thus subject to the same errors. This can lead of majority votes to wrong prediction as well. To avoid this:

- Do ensemble on different type of algorithms as it could make different types of errors.
- **Bagging:** Perform ensemble on the same algorithm but on different random samples of the training data. Random Forests comes here as an ensemble method over Decision Trees.
- **Pasting:** When bagging is done without replacement.

RANDOM FORESTS

Random Forests belongs to the school of **"Ensemble Learning".** Random forests is an ensemble of decision trees. Here, we obtain the predictions of all individual trees and then predict the class that gets most votes. To create more randomness in trees, the randomforest classifier selects not the very best feature at each node but selects only from a sub-set of features.

Python Implementation:

- **Parameter Tuning:**

```
param_dist = {'n_estimators': [100,200],
               "max_features": ["auto", "log"]}
rf = RandomForestClassifier()
rf_cv = GridSearchCV(rf, param_grid = param_dist,  #You can also use Randomized Search
                           cv = 10, n_jobs = -1)
rf_cv.fit(X, y)
print("Tuned Random Forest Parameters: %s" % (rf_cv.best_params_))
```

- **Fitting RandomForest:**

```
from sklearn.ensemble import RandomForestClassifier
Ran = RandomForestClassifier(max_features= 'auto', n_estimators= 200, random_state=0,
criterion= 'gini', max_depth= 3,                                    min_samples_leaf=9)
Ran.fit(X_train, y_train)
y_pred = Ran.predict(X_test)
#print('Accuracy:', accuracy_score(y_pred,y_test))


## 5-fold cross-validation
cv_scores =cross_val_score(Ran, X, y, cv=10)


# Print the 5-fold cross-validation scores
print()
print(classification_report(y_test, y_pred))
print()
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)),
      ", Standard deviation: {}".format(round(np.std(cv_scores),4)))
```

Advantages of Random Forests:

- **Helps in Feature Selection:** You can score each feature.

```
imp_feat = pd.DataFrame()
imp_feat['features'] = X.columns
imp_feat['Feature_Score_rf'] = Ran.feature_importances_
top_10 = imp_feat.sort_values(by=['Feature_Score_rf'],ascending=False)[:10]
top_10
```

EXTREMELY RANDOMIZED TREES (Extra Trees)

In Random Forests, a few features (max_features) are randomly selected at each node for the best-split. But to further increase the randomness of the forest, even the **threshold** at which the split takes place can also be randomly selected, instead of simply opting for the best threshold. This technique **trades more bias for a lower variance.** It is also faster than regular random forests because finding the best threshold value is a very time consuming process that is skipped.

Python Implementation: same as RandomForests

```
from sklearn import ExtraTreesClassifier
```

BOOSTING

The idea is to train predictors sequentially, each trying to correct the predecessor.

Adaptive Boosting:

As long as the individual model is slightly better than pure random guessing, AdaBoost will converge to give a very good prediction. In this method, every data point is given a weight.

1. Initially, each data point is given the weight of 1/m. (m = number of data points)
2. Then, the prediction is made and mis-classifications are noticed, errors are calculated
   -

     Weighted error rate of each predictor is calculated.
     - This will be high if there are more number of mis-classifications.
     - This will be near 0 if the predictor is accurate.
   -

     == **Predictor weight.** Here alpha is set to 0.5 | scikit learn default =1
     - If the predictor is accurate, predictor weight will be high.
     - For random guessing, predictor weight will be 0
     - If accuracy is lesser than random guessing, then the weight will be negative.
3. Then update the weights of data points;
   - No change in weight if the prediction was correct
   - Increase the weight of the data point if it was a mis-classification. w_new = w_old*exp(alpha)
     - That is how every successor predictor corrects the predecessor.
   - Then normalise all weights so that sum=1.
4. Go to step 2.
5. Stop the algorithm after desired number of predictors is reached, or after a predictor of required accuracy is reached.

Final Prediction: For a datapoint, calculate the predictions made by all successive predictors, weigh them using their predictor weights. The prediction that gets max value is selected. This is **hard voting.** You can even use **soft voting** by using **SAMME.R**

Python Implementation:

```
from sklearn.ensemble import AdaBoostClassifier / AdaBoostRegressor

ada_clf = AdaBoostClassifier(DecisionTreeClassifier(), n_estimatores= ,
algorithm='SAMME/SAMME.R', learning rate=)
ada_clf.fit(X_train,y_train)
```

Gradient Boosting

Similar to Adaptive Boosting, but the successive predictor/regressor is not dealing with the weights

Similar to Adaptive Boosting, but the successive predictor/regressor is not dealing with the weights. Each successor predictor is fitted to the residuals generated by previous predictor. See the figure in the book for intuition.

Gradient boosting by default takes the base estimator as decision tree only. So all the hyoer parameters that apply to decision tree, are applicable to Gradient Boosting as well.

subsample hyper parameter: Use it when you don't want to use the entire data for training each tree, there by increasing randomness in the model. Again you are trading higher bias for lower variance. This is called **Stochastic Gradient Boosting.**

Python Implementation:

```
from sklearn.ensemble import GradientBoostingClassifier/Regressor
gbrt = GradientBoostingRegressor(n_estimators = , max_depth=, learning_rate=)


#Choosing n_estimators
errors = [mean_squared_error(y_test,y_pred) for y_pred in gbrt.staged_predict(X_test)]
best_n_estimators = np.argmin(errors)+1

gbrt_best = GradientBoostingRegressor(n_estimators = best_n_estimators)
```

Python Implementation of XGBoost: Extreme Gradient Boosting. Faster. It also automatically takes care of early stopping, you need not choose number of estimators.

```
from xgboost import XGBRegressor

xgb_reg = XGBRegressor()
xgb_reg.fit(X_train,y_train, eval_set=[(X_test,y_test), early_stopping_rounds=5]) #
n_estimators will be chosen when error does not change for 5 consecutive iterations. This is
optional.
y_pred = xgb_reg.predict(X_test)
```

Machine Learning - Teaching Computers to learn and think for itself, without explicitly being programmed.

When a computer can learn from it's Experience (E) of a Task (T) and there by increase its performance measure (P) at that particular task, it is said to be machine learning. For example consider a Machine Learning algorithm of automatic spam filter in our emails. Here is what it does:

- T = Our Task of selecting few emails as Spam and others as not spam.
- E= Computer watching our T
- P = Percentage of emails correctly filtered by the computer

- P – Percentage of emails correctly filtered by the computer.

Types of ML algorithms: Model based learning | Instance based learning.

- Supervised algorithm => Here, we are going to teach the computer how to do something.
  - Linear Regression.
  - Logistic Regression - Classification
  - KNN
  - Random Forest
  - Decision Trees
  - Anomaly detection: Eg: FB login warning from different city suddenly
  - SVMs
- Unsupervised algorithm => Here, the computer learns by itself.
  - Clustering
    - K-Means
    - Hierarchical Clustering
    - Expectation Maximisation
    - DBSCAN
  - Dimensionality Reduction
    - Principal Component Analysis
    - Kernel PCA
    - Locally Linear Embedding
    - T-distributed Stochastic Neighbor Embedding.
  - Association Rule Learning: To find out interesting relations between attributes. For example: Walmart uses it to find out relationships between purchases = people who purchase sauce also buy maggi = such type of relations.
    - Apriori
    - Eclat
    - F Growth
  - SVD
- Reinforcement algorithm
  - https://m.youtube.com/watch?list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ&v=2pWv7GOvuf0
- Recommender systems. Eg: NetFlix recommends cinemas.

CLASSIFICATION

In Classification problems, the **dependent variable (y) is a discrete variable.** For example, estimating whether there is a tumor is malignant or not based on the tumor size (x). Often there can be more than 2 discrete values for 'y'. But in general it is either 0 or 1 representing 'No' or 'Yes'. Moreover there can be more than 1 independent variables as well when 'y' has to be classified based on various factors (x1. x2, x3..).

Use-cases:

- **Customer churn prediction =>** Will customer leave or not?

A.) EDA that you should do first:

A.a) Univariate distribution by target variable: both categorical and numeric features.

```
#For numeric variables
fig, ax = plt.subplots(len(X.columns), 1, figsize=(10, 200))
ax = ax.flatten()
index = 0
for i in range(len(X.columns)):
    col_name =df.columns[i]
    class_0 = df.loc[cancer_df['Condition']==0][col_name]
    class_1 = df.loc[cancer_df['Condition']==1][col_name]
    plt.title('Frequency distrbution of feature')
    sns.distplot(df[col_name],color='black',bins=100,label='Full',ax=ax[index], hist=)
    sns.distplot(class_0, kde=True, color = 'green', bins=100,label='Class0',ax=ax[index],
hist=)
    sns.distplot(class_1, kde=True, color = 'red', bins=100,label='Class1',ax=ax[index],
hist=)
    index +=1
    #plt.legend()
plt.show()

#For categorical variables
figure, axes = plt.subplots(3,3,figsize=(20,15),facecolor='white')
axes = axes.flatten()
figure.suptitle('Frequency of Categorical variables by default rate', size=20)

cat_columns = ['SEX','EDUCATION','MARRIAGE','PAY_1','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6']
index=0
for i in cat_columns:
    sns.countplot(x=i, hue='<target>',data=df, ax=axes[index])
    index = index+1
```

- **In numeric distributions:**
  - See the overlap between classes, if the feature distributions overlap across classes, then that would not be a good feature for logistic regression.
  - Also you can use **swarm plots or box plots or violin plots** to check the overlap.

```
pivot = pd.melt(df,id_vars="<target>", var_name="features", value_name='value')
plt.figure(figsize=(10,10))
sns.swarmplot(x="features", y="value", hue="<target>", data=pivot)
sns.violinplot(y="value", x="features", hue="<target>", data=pivot, split=True,
inner="quart")
plt.xticks(rotation=90)
```

A.b) Bi-variate analysis accross different features using boxplots, scatters, pairplots.

```
import seaborn as sns
sns.pairplot(df,hue=,palette=,markers=[],vars=[],x_vars=[],y_vars=[],kind='scatter'/'reg')

from pandas.plotting import scatter_matrix
scatter_matrix(df, figsize=)
```

B) Algorithms

Ba) LOGISTIC REGRESSION:

B.b) Soft-max Regression:

B.c) Decision Tree for Classification:

B.d) Random Forests:

C) Selecting best model in classification algorithm: Cross validation and Confusion Matrix

- **Confusion Matrix:** https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/
  - Cross tab of Actual and Predicted classifications.
  - **Classification Accuracy** = (TP+TN) / Total
    - But this alone will not help in selecting the best model. Why? consider the following example =>
      - The breast cancer dataset is a binary classification problem where there are 286 women with labelled data if they have cancer or not. 201 women don't have cancer in this data.
      - Now, say you built a faulty model that predicts NO CANCER for every women. This model would in fact be true in 201 cases (TN) in this dataset, thereby giving the accuracy of 201/286 = **70.28%! WOW**
      - But is it helpful? There will be **85 false negatives** in this case i.e women who have cancer, are being predicted as not having cancer. These women will not get medication and they will die.
      - Hence, accuracy alone will not help in selecting the model. In fact, it is the **dataset imbalance** that led to this high accuracy even if the model simply returns NO for each women.
      - In fact, even if our accuracy is lesser than 70%, if it can reduce false negatives, that will be a better model in this problem because false negative here kills a life. This is called **Accuracy Paradox.**
  - **Precision** = TP / Predicted Positives.
  - **Recall** = TP / Actual Positives
  - As your cutoff increases, your precision increases but recall reduces. How to find the right balance between precision and recall? **F1Score**
    - 2*((precision*recall)/(precision+recall))
    - In the same breast cancer problem, say if we build a model that returns YES for every women. Then TP=85, FP=201, There will be a F1-Score of 0.46! => This will be the minimum requirement because you are not killing anyone with this model

Actual \ Prediction	1 = Pos	0 = Neg

1 = Pos	TP	FN
0 = Neg	FP	TN

```
#Cross Validation
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix,
classification_report, cross_val_score

cv_scores =cross_val_score(Ran, X, y, cv=5) #5 fold
print(classification_report(y_test, y_pred))
print("Average 5-Fold CV Score: {}".format(round(np.mean(cv_scores),4)),", Standard
deviation {}".format(round(np.std(cv_scores),4)))
cnf_matrix = confusion_matrix(y_test, y_pred)
precision_score(y_test, y_pred)
recall_score(y_test, y_pred)
```

- **ROC Curve:** FP rate (X-Axis) vs TP rate or **Recall** (Y-Axis) | Receiver Operating Characteristic Curve.
  - Better model is the one with higher area under the ROC curve (AUC)

```
from sklearn import metrics


#Step 1 => Find prediction probabilities with test data
y_pred_proba_lr = logreg.predict_proba(X_test)[:,1]
fpr1, tpr1, _ = metrics.roc_curve(y_test,  y_pred_proba_lr)
auc1 = metrics.roc_auc_score(y_test, y_pred_proba_lr)

#Step 2 => Plot ROC curve
plt.figure(figsize=(10,7))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr1,tpr1,label="Logistic Regression, auc="+str(round(auc1,2)))
plt.legend(loc=4, title='Models', facecolor='white')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC', size=15)
plt.box(False)
```

Then how will you know which model best suits in solving your problem? Here, we use various metrics.

- http://machinelearningmastery.com/how-to-choose-the-right-test-options-when-evaluating-machine-learning-algorithms/

New sklearn verison check karo. https://lnkd.in/eyak_wM

Use RAPIDS for faster fitting of model =>https://www.linkedin.com/feed/update/urn:li:activity:6622471003857661952

/?lipi=urn%3Ali%3Apage%3Ad_flagship3_feed%3ByBdGp21TT5OVFIgyV7oYjw%3D%3D

Decision Trees

These can perform both classification and regression tasks. Moreover, we saw that **SoftMaxRegression cannot produce multi-output tasks** like recognizing all people in an image. But, Decision Tree can do that. Moreover, Decision Tree is a **whitebox** model. We can intuitively understand how the decision is being taken. In contrast, Random Forests, ANNs are **black box models.**

Before heading into the algorithm, what are the important parameters and terminology that we would come across? Knowing this will make our learning simpler.

- **Terminology:**
  - **Root node:** The first node at depth=0
  - **Leaf node:** Node that does not have any child nodes. So basically this node will not ask any further question. **It only predicts.**
  - **samples:** Every node has a 'samples' attribute. It simply means number of data points this node is applicable for.
- **Hyper parameters:**
  - **min_samples_leaf** => Minimum number of samples a leaf node must have.
  - **min_weight_fraction_leaf =>** Same as above but expressed as fraction of the total number of weighted instances .
  - **max_leaf_nodes =>** Max number of leaf nodes
  - **min_samples_split =>** Minimum samples a node must have before it can be split.
  - **max_features =>** Maximum number of features that should be evaluated for splitting at each node.
  - **max_depth** => What is the maximum depth of the tree?? Used as a stopping criterion.
  - **criterion =>** gini or entropy = impurity measures.
    - Mostly they both give similar trees. But Gini is faster.
    - https://homl.info/19

You can tune the best parameters and hyper parameters using GridSearch or RandomizedCV

```
param_dist = {"max_depth": [1,2,3,4,5,6,7,8,9],
              "max_features": [1,2,3,4,5,6,7,8,9],
              "min_samples_leaf": [1,2,3,4,5,6,7,8,9],
              "criterion": ["gini", "entropy"]}


tree = DecisionTreeClassifier()
tree_cv = RandomizedSearchCV(tree, param_distributions=param_dist, cv=5, random_state=0)
tree_cv.fit(X_train, y_train)
print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
```

Entropy: H

$$H = -\sum p_i . log_2 p_i$$

==>> Entropy for a leaf.

p_i = probability of class i in that node samples {But why is this a formula?}

If you want entropy at a level, you should sum entropy at all leaves of that level using weighted averages. The goal is to reduce entropy as we go down the tree. A node's entropy is 0, if it contains instances of only one class. Just like gini=0

Information Gain (IG) = Entropy before split - Entropy after split.

That split which has higher Information Gain is the best. But not every time IG is beneficial! If you split on ID column (or such high cardinal variables), you'll get super high IG (low entropy), but of no use. Hence, we should normalise IG with **Information Content (IC).**

$$IC = -\sum f_i . log_2 f_i$$

, where f is the fraction of split. IC will be high for high cardinal variables.

Gains Ratio = IG / IC

GINI Index: This measures the impurity at the node. A node is said to be "pure" if all the data points at this node belong to the same class. **GINI will be 0 for pure class.**

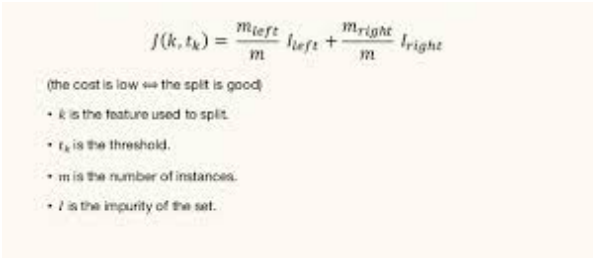
- $$GiNi = 1 - \sum_{1}^{m} p_i^2$$
- There is no split possible once we reach GiNi = 0 at a node.
- The objective of the model will be to **reduce GiNi at leaf nodes.**

Chi-Square / Contingency Table: Useful when target is categorical and attribute is also categorical. In case attribute is numeric, we can bin them.

For every categorical attribute, find the chi-square. It will tell you if the two variables are dependent or independent. Then you can split on those variables that are dependent = **High chi-square**.

Popular Decision Tree Algorithms:

- **CART: Scikit learns uses this.** Classification and Regression tree.
  - Binary split at every node. => Only Yes/No answers.
  - The algorithm automatically chooses the **feature(k) and threshold value(t_k) of the feature** to make splits at each node. It does this by minimising the cost function:



    - l= GINI
  - Uses GiNi Index as criterion.
  - Cost Complexity Pruning
  - **CART algorithm is a greedy algorithm**: It looks for the best split at top-level first and then repeats this at every node. It will give a 'reasonably good' output, but it may not give the lowest possible impurity level over all at deep depth. Such an optimal tree is known as **NP-Complete Problem.** It will take extraordinary time to do that.
- **C5.0**
  - Multi Split
  - Info Gain
  - Pessimistic Pruning.
- **ID3:**

Regression Trees:

- Regression through Decision Tree. Here, the target variable is numeric
- Instead of entropy, consider variance here.

Disadvantages:

- **Sensitive to training data:**
- **Danger of Over fitting:** Decision Tree is a **Non-Parametric Model.** It does not mean that there are no parameters (there are so many as we saw). But the parameters are not defined at the start. **(High degree of freedom).** So the model is free to stick closely to the the data, leading to overfitting.
  - Regularization is thus required to restrict the freedom. **Thus use the hyper parameters**
    - Increasing min_* hyper parameters or reducing max_* hyper parameters will regularize the model.

Advantages:

- No need of data preparation => There is no need of scaling and centering of data.
- Very less computational complexity =>
  - For predictions, the complexity is of order **O(log_2(m))**
  - For training, the complexity is of order **O(n * m log_2(m))**
    - For smaller training datasets, we can set **presort=True** for faster training.
-

When to use Decision Tree?

- When there is very high non-linear data. Linear Regression and SVM help linear data more.
- When number of features are very high but number of records are less, linear models perform better than non-linear models. Non-linear models will over-fit here.
- When number of features are less but number of records are high, non-linear models perform better.

Decision Tree implementation on Python:

```
from sklearn.tree import DecisionTreeClassifier
Tree = DecisionTreeClassifier(criterion= 'gini', max_depth= 5, max_features= 5,
min_samples_leaf= 6, random_state=0)
Tree.fit(X_train, y_train)
y_pred = Tree.predict(X_test)
```

To see your tree:

```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(Tree, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names=X_train.columns,class_names=['No
Default','Default'] )
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Clustering - UnSupervised.

CLUSTERING

Not only data, even audio files can be clustered!

- Partitioning creiteria
- Hard vs Soft
  - **Hard Clustering:** Assigning each data point to a single cluster.
  - **Soft Clustering:** Each data point will be given a score per cluster. This score can be:
    - Distance between the point and the centroid of cluster.
    - Similarity score (Affinity) such as Gaussian Radial Basis Function (RBF)
- Similarity measure
  - Distance based vs Density based.
- **Major clustering approaches:**
  - Partitioning approach

        - Here, we try to minimise the sum of squared distance between points and the cluster center.
        - K-Means, K-Mediods, CLARANS
    - Heirarchichal approach
        - Dendogram = binary tree
        - Diana, Agnes, BIRCH, CAMELEON
    - **Density based approach:**
        - DBSCAN, OPTICS, DenClue

K-MEANS

- **Issues with K-Means**
    - **It is only good with spherical clusters.**
        - Hence it is of extreme importance to do feature scaling. Otherwise, your clusters will be stretched and K-Means will fail. Feature scaling doesn't promise a guaranteed spherical cluster, but generally improves things.
    - **Extremely sensitive to outliers.**
    - **Sub-optimal Solutions:** In K-Means, we select the initial centroids randomly. If you're unlucky the clustering can converge to a local optimum point, leading to sub-optimal solutions.
        1. **init** hyperparameter**:** If you know approximately where the centroids will be, them pass in a array of centroids as initial random centroids.
        2. **n_init** hyper parameter: You can run the algorithm multiple times with different random institutionalizations and then select the best. By default, n_init is set to 10. So Scikit learn runs the algorithm ten times for ten different initializations and stores the best clustering.
        3. But how to know which centroid/cluster is the best? The performance metric **inertia** helps is making this decision. It is the **mean squared distance between each data point and it's closest centroid, as given by the algorithm.**
        4. **K-Means++:** A smarter way to initialize centroids. It increases the computation but reduces the chances of converging to sub-optimal clusters. This approach tends to select centroids that are distant from each other
            1. In fact, SciKit learn is using this method as default method of initializing centroids. If you want to change it, use **init** hyperparameter:
                - Either set it to "random"
                - Or pass an array of centroids.
- **Choosing number of Clusters**
    1. **Elbow Method:**
        - Plot inertia against the number of clusters.
        - See the inflection point.
    2. **Silhouette Score =** Mean silhouette coefficient over all data points.
        - **Silhouette coefficient** of a data point **= (b-a)/max(a,b).**
            - **a =** Mean of the distances of the data point from other data points in the same cluster.
            - **b=** Mean of the distances of the data point from other data points in the next closest

cluster.

- The silhouette coefficient ranges between -1 and 1. +ve indicates that the data point is in the correct cluster, -ve indicates wrong cluster. 0=Cluster boundary.
- Silhoutte score is better than Elbow Method. In elbow, the inertia actually keeps decreasing as we increase k. This is because, if there are more number of clusters, then automatically the Mean Squared distance will be less. But Silhoutte Score reduces if you diverge from the optimum number of clusters.
- You can also use **silhouette diagram** to make the decision about number of clusters.

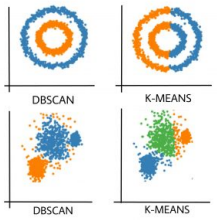
K-Means Implementation:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters =k , init= ,n_init=1)
y_pred = kmeans.fit(X)
y_pred = kmeans.predict(X)

kmeans.transform(X_new) #Returns the score (distance) of each point in X_new for each
cluster.
kmeans.inertia_ #Returns the inertia score of the clustering
kmeans.score(X) #Returns the negative inertia score of the clustering

kmeans.cluster_centers_ #Returns the centroids of the clusters

#To see the silhouette score
from sklearn.metrics import silhouette_score
silhouette_score(X,kmeans.labels_)
-----------------------------------------------------------------------------------------------
----------------------------
#Elbow Method
inertia = []
for i in list(range(2,7)):
    kmeans = KMeans(n_clusters =i)
    y_pred = kmeans.fit_predict(X)
    inertia.append(kmeans.inertia_) #Returns the inertia score of the clustering
plt.plot(range(2,7),inertia, marker='v')
plt.xticks(range(2,7),labels=['2','3','4','5','6'])
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.annotate('Elbow', xy=(,))
plt.show()
```

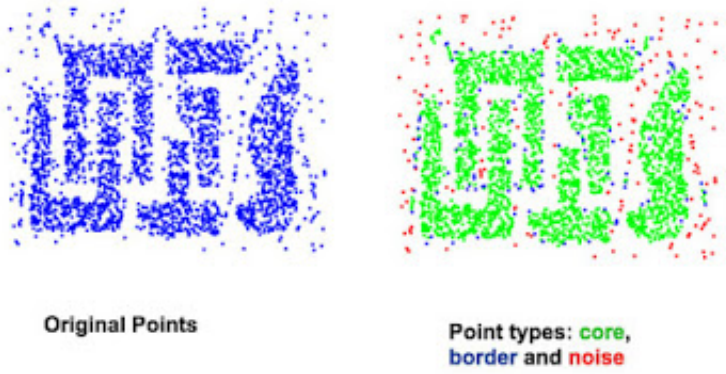
MiniBatchKMeans Implementation: The implementation is just like K-Means. But it is used when there is huge data. Of-course it has higher inertia but is faster.

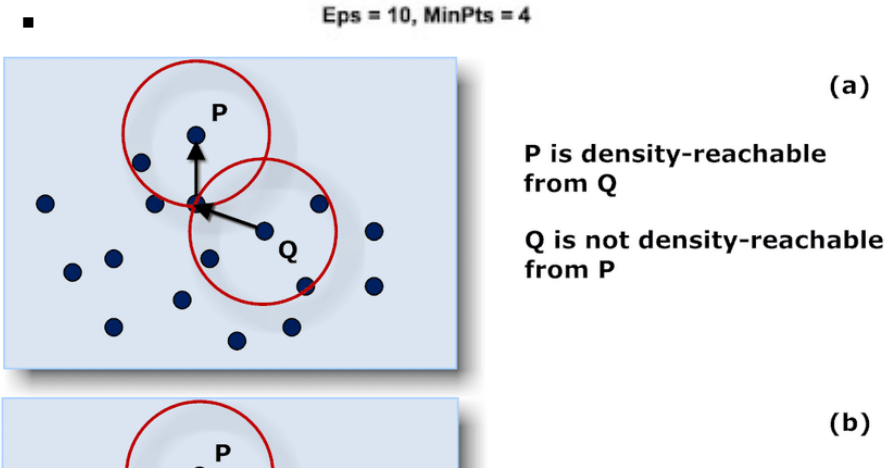
```
from sklearn.cluster import MiniBatchKMeans
```

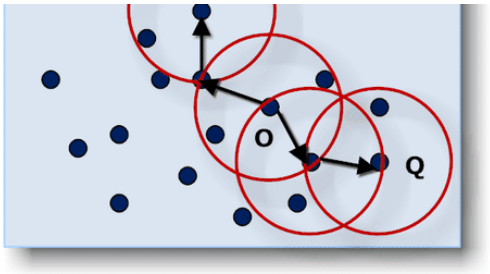
DBSCAN: Density based Spatial clustering of Applications with Noise.
This is good for clustering arbitrary shapes like squares etc. K-Means is suitable only for spherical clustering.



- What is density based clustering? => Here the data points that are close to each other form a cluster.
- Thus, it is good at detecting the outliers or noise, because they do not have neighbours.
- Also, we need not chose number of clusters in DBSCAN.
- Every ML Algo has few parameters that have to be tuned. There are two such in DBSCAN
  - ϵ = radius around a point
  - MinPts = Minimum number of points in the given ϵ
- What will DBSCAN actually do?
  - It will classify the data points into three classes => **Core/Dense, Border and Outliers/Noise**
    - Core => A point will be classified a Core point if there are at least "MinPts" number points in the ϵ radius around it.
    - Border => Not a core point and it should be inside the ϵ radius of another point.
    - Outlier => Neither core nor border.



- 



P and Q are density connected to each other by O

- **(a) Direct Density reachable:** If a point 'q' is within the ϵ radius of a **core point 'p'** ; Then 'q' is the direct density reachable from p.
- **(a) Density reachable:** Two points are density reachable, if there is a path of direct density reachable connection between them.
- **(b) Density Connected:** Two points p and q are density connected if there exists a point 'o' that is density reachable from both p and q. Here 'o' can be a border point as well.

- **The Algorithm:**
  1. Select a point 'p' randomly.
  2. Ignore if 'p' is not a core point. If it is a core point, then all the **direct density reachable points ('q's)** i.e those inside the ϵ radius form a cluster C.
  3. If 'q' is also a core point, then all the direct density reachable points from 'q' are again merged into C. This way we keep extending the cluster to include all those **density reachable points from 'p'**
  4. Sometimes a border point would be a border point of two or more clusters. In such cases, merge all the involved clusters.
  5. Repeat the process until all the data points are visited. The points that are not included => Outliers.
- **BE CAREFUL:**
  - in choosing ϵ and MinPts. Understanding data well is required to be able to do it.
  - If density is not same across data, there is a problem of bad clusters. Hence this algorithm is good for the data that contains clusters of similar density.
- **Implementation:** https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

---

AGGLOMERATIVE CLUSTERING

This model builds a hierarchy of clusters from the bottom-up. It starts with considering each individual data points a cluster. Then, for every iteration, similar clusters keep merging up the tree - until K clusters are formed. Even when your clusters are of any shape, this model will work well. But you'll have to provide a **connectivity matrix** for this algorithm to run fast.

KNN

- Instance based learning. Therefore there are computational challenges.
- Scaling is must Min Max scaling.
- **Various distances:**
  - Euclidean.
  - Minkowski.
  - Cosine Similarity.
  - Manhattan distance. Useful in map based applications.
- Useful in NLP
- Weighted Euclidean / Minkowski
- Create dummies for categorical variables. One hot Encoding / Label encoding
  - Alternatively, Hammind distance.
- Start with k=1
- KD Trees

```
from sklearn.neighbors import KNeighborsClassifier
```

SVM

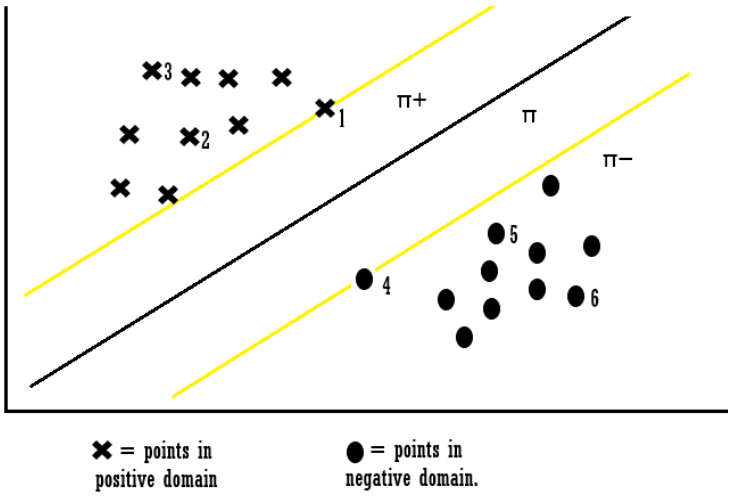
SUPPORT VECTOR MACHINE

- Why SVM?
  - Classification => SVC = Support Vector Classification. Best use of SVM
  - Regression
  - Both Linear and non-linear problems can be solved.
  - SVM is also helpful in outlier detection
- When to use SVM?
  - Many features but only few records
  - Highly sparse data i.e missing values are more
- **Hard vs Soft Margin Classification:**
  - **Hard Margin:** When you want your SVM plane to perfectly classify classes. But if the classes are not linearly seperable this becomes impossible. Also SVM is sensitive to outliers and a hard margin will not give a generalised results. Because, your margin will be supported by the outlier!
  - **Soft Margin:** Here you allow a few margin violations and mis-classifications. You should try to find balance between high margin and less margin violations.
- Hyperplane =>

$$y = w_0 + w_1x_1 + \underset{m}{w_2x_2} + w_3x_3 \ldots .$$

$$= w_0 + \sum_{i=1} w_i x_i$$
$$= w_0 + w^T X$$
$$= b + w^T X$$

- This plane classifies the data.
- If there are n-features; hyperplane will have n-1 dimensions
- W is the vector normal to the hyper plane.
- But this equation, you'll build in logistic regression also. The speciality of the SVM is **MARGIN (M)**
  - If the decision boundaries are close to the data points, there is high chance that the model is not accurate for new data. Hence, **SVM tries to maximise the margin.** This is like laying a road as wide as possible between different classes.
  - Increase width of the plane till you hit a data point on both sides of the plane. That is why SVM does not change when you add more data "off the street". SVM is only supported by those on the street edges.
  - The higher the margin, the better the classifier.
  - M = 2/W.W | For max M, W.W should be minimum
- Assume 3 hyper planes=>



$$\pi = b + w^T X = 0$$
$$\pi^+ = b + w^T X = 1$$
$$\pi^- = b + w^T X = -1$$

- What if data is noisy?
  - Slack
  - C Parameter
    - High C => mis-classification reduces but margin also reduces.
    - Low C => High margin but high mis-classification.

When the classification is non-linear:

- 1. Add a new polynomial feature! This will help in creating linearly separable dataset.

```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree= , include_bias=T/F, )
X_poly = poly_features.fit_transform(X)
#This will add new polynomial features 'up to' the degree specified. When multiple features
(x1,x2) are present even x1*x2 feature will be created
```

```
(x1,x2) are present even x1*x2 feature will be created.


If there are n features and degree=d, (n+d)!/n!.d! features will be created
```

This above polynomial method can be used for any model, not only SVM. As you see, if there are mutliple features, then huge number of polynomical features could be created leading to very slow algorithm and also probably over-fitting. SVM has **kernel trick** to resuce us and is thus a sought after model for non-linear data.

- **Kernel Trick:** Produces the same result as that of adding multiple polynomial features, but not by adding them to the data. This returns the dot product of the original vector and the transformed vector.
    - Polynomial Kernel => K = (x*y+1)^d
    - Radial Basis Function (RBF) Kernel =>
    - Linear Kernel => K = x*y
    - Sigmoid Kernel
    - String kernel
  - Each Kernel would have other parameters that have to be tuned.
  - This is where SVC wins over Logistic Regression. Even if the data is non-linear, Logistic Regression will only linearly classify data. This leads to immense mis-classification.
- **Regularization using Hyperparameters:** When your SVM is over-fitting, you can regularise it using hyper parameter **"C".**
  - **C:** Reducing C will increase the margin and thus allow for more generalisation. But, it will also increase misclassifications.
  - **loss** = set it to "hinge"
  - dual = set it to "False" unless there are more features than training instances.

```
from sklearn.svm import SVC, LinearSVC

from sklearn.model_selection import KFold
folds = KFold(n_splits=5, shuffle=True, random_state=0)
for train_index, test_index in folds.split(X,Y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
svc = SVC(kernel = 'linear' / 'rbf' etc ,C =     )
scv.fit(X_train, Y_train)
y_pred = svc.predict(X_test)

print(classification_report(y_test, y_pred))
```

- Benefits of SVM:
  - Even Highly sparse data can be used = SVM is supported only by the edge points.
  - Other ML algorithms require more data, but SVM can produce accurate predictions with less data.
- Issues with SVM?
  - Sensitive to noise.
  - SVM does not provide a probability like logistic regression.

- Not for multi-class classification.
- Sensitive to scale = You have to do feature scaling. For SVM it is better to use **StandardScaler** of sklearn because it will center the data. As SVM regularises the bias term, it is better to center the data.

Import Numpy as np

Numpy (Numerical Python) is for **arrays**. Arrays are similar to lists but they can store only one datatype elements. Also several math operations are possible on array but not on lists. The data type of elements in an array is **dtype**

CREATING AN ARRAY:

- X = **np.array([**lst1,lst2...] **) =>** Each lst will be a row in the array
- X = **np.empty(**shape = (2,2), dtype = int, order = C/F**) =>** Creates an uninitialized array
- **X = np.zeros(**shape = (2,2), dtype = float, order=C/F**) =>** Creates a zero array
  - **np.zeros_like(df)** => Returns an array of 0s with the size as that of df
    - np.zeros_like(df, **dtype= np.bool**) => replaces the 0 with False, 1 with True.
- X = **np.ones**(shape = (2,2), dtype = float, order=C/F) => Creates an array with 1s.
- I = **np.eye**(size= ,dtype= ,order=) => Identity matrix
- X = **np.linspace**(1,10, num=10, endpoint = True, retstep =False, dtype= ,) => Creates a one-dimensional array of (num=10) evenly spaced numbers over the specified interval (including last value as default, other wise use endpoint = False)
  - If retstep = True; Then, a tuple will be returned with both one-dimensional array and step size of interval.
  - This works a little different from the **range()** function.
    - In range(), you can specify start, stop(not included) and the step size. But you cannot specify number of intervals that you want, which you can do in linspace
    - range() only returns integers and works with integer step sizes.
    - range is for lists, for arrays use arange():
      - X = **np.arange**(start= , stop= , step=, dtype=) => Create a sequence array. Stop will not be included.

INDEXING: Same as lists.

- X.iloc[] => use indices
-

FILTERING:

- Use **np.array**(list comprehension)
- Using **boolean indexing**:

    - **[X>2]** => Applies (>2) condition on every element of the array and returns T/F
    - Use that boolean array to filter.
- **np.where(X>2)** => Returns an indices of elements in X that satisfies the condition.
    - **X[np.where(condition)]** => Filtered array

METHODS:

- **X.shape** => Returns shape of the array X as a tuple.
- **X.reshape(**new_size**)** => Returns a new array with new_size and elements from X.
    - **X.flatten() =>** Returns a new array after flattening the array X into 1-D
    - **X.ravel() =>** Same as flatten but modifies the array X itself
    - **X.T =>**Returns a new array - transpose of X
    - Numpy can take one of the sizes are **-1** meaning unspecified. It will take the size from the array.
- **X.size** => Returns size of the array as an int.
- **X.ndim** => Returns number of dimensions (2-D, 3-D etc) of the array as an int

FUNCTIONS: Always returns

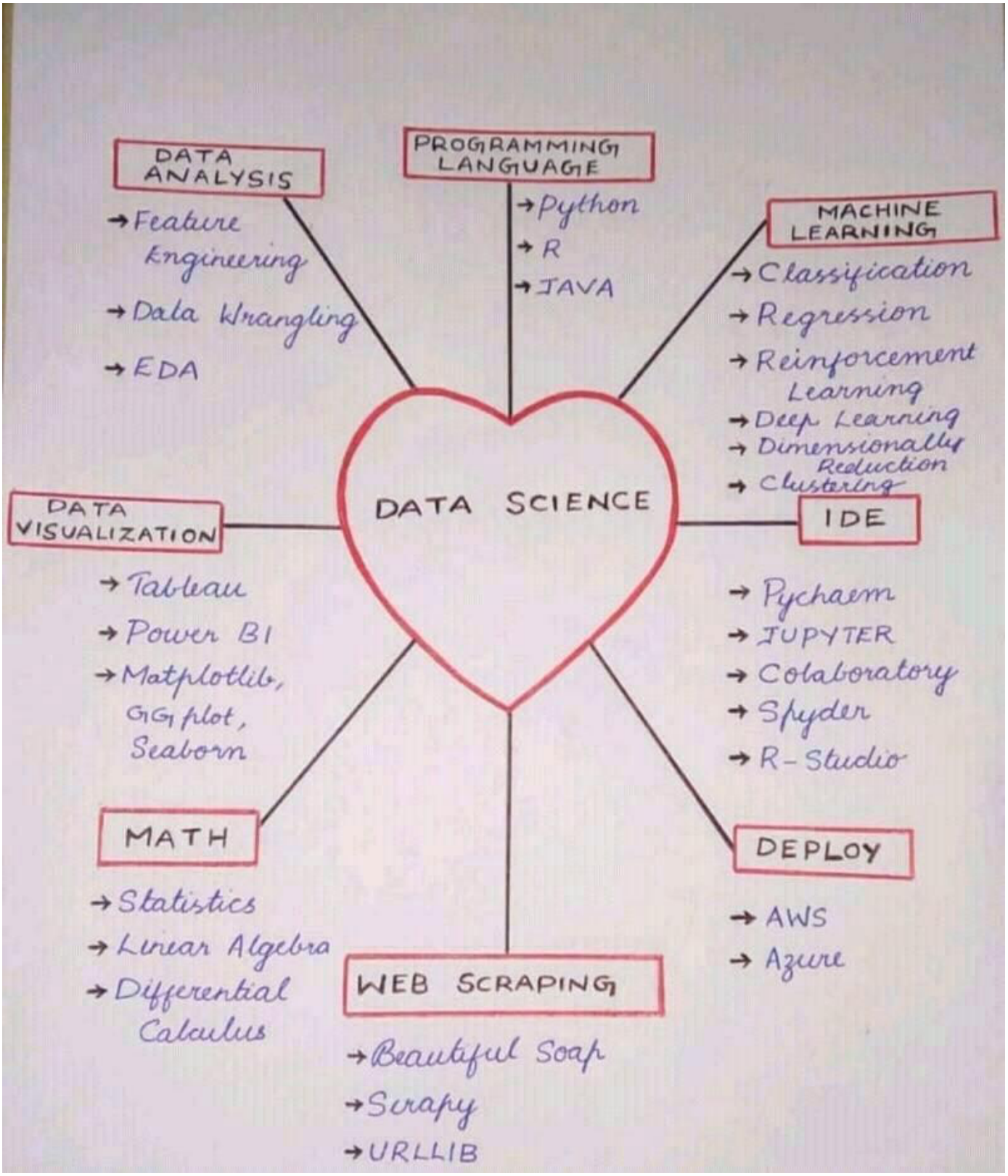
- **np.linalg.inv(X)** => Returns an inverse matrix of X.
    - .pinv() => psuedo invere = for singular matrices.
- **np.triu_indices_from(df)** => Returns the **indices** of the upper triangle of df
- **np.hstack((X,Y))** => Merges X and Y horizontally.
- **np.vstack((X,Y))** => Merges X and Y vertically.
- **np.cumsum**(**X**) => Cumulative, rolling sum of successive elements expressed as a new 1-D array
    - Useful for sales data.
- **np.unique(X)** => Returns a new 1-D array with unique elements of X.
    - Helps in data cleaning => Check if all categorical variables of a column are correctly spelled.
    - Use **axis =0/1** for unique columns and rows as a whole. Returns a new array after deleting duplicate rows/columns
- **np.sum(X)**
    - Use axis for row wise / column wise sums.
- **np.min(X)**
    - Use axis for row wise / column wise min.
- **np.max(X)**
    - Use axis for row wise / column wise max
- **np.argmin(X)** => Returns index
    - Use axis for row wise / column wise
- **np.argmax(X)** => Returns index
    - Use axis for row wise / column wise sums.
- **np.sort(X,axis=)**
    - Here default axis is sort across rows.
- np.argsort(X) => Returns
- **np.save(**'name.npy', X**) =>** Saves the array as a file
- **X = np.load(**'name.npy'**)**

- **Set Functions:**
  - **np.in1d(X,Y)** => Check if each element of X is in Y and return a boolean array of size X.
  - **np.intersect1d(X,Y) =>** returns intersection of X and Y in 1D array.
  - **np.union1d(X,Y) =>** returns a new 1D array combining all elements of X and Y. It even changes the dtype if required, cuz arrays have only dtype.
  - **np.setdiff1d(X,Y)** => X-Y
  - **np.setxor1d(X,Y)** => X XOR Y
- **Math:**
  - **np.sqrt**
  - **np.abs**
  - **np.exp =>** e^
  - **np.sign**
  - **np.sin/cos/tan**
- **np.isnan** -=> Data cleaning.
- **np.isinf**
- **np.isfinite**

RANDOM package in numpy: from **numpy** import **random**

- **random.seed(**any_number**) =>** Same random numbers will be generated when ever we generate them if we first seed the random package.
- **random.rand**(3) => Returns 3 random numbers between 0-1 | Uniform distribution
- **random.randint**(3,10,5) => Returns 5 random integers between 3 and 10 | Uniform distribution
- **random.shuffle(X)** => Shuffles X. If you use random.seed, every time it shuffles similarly.
  - If X is a matrix => Shuffles rows; not elements.
- **random.normal(loc= , scale= , size=)** => Generates 'size' number of random numbers with normal distribution of:
  - Mean = loc | This can be array - to provide multiple means of multiple curves
  - SD = scale | This can be array - to provide multiple SDs of multiple curves
  - Also for standard normal distribution i.e between 0-1; use directly **random.randn()**
- **random.binomial(n=, p= , size=) =>** Generates 'size' number of random numbers with binomial distribution of:
  - n trials | This can be array - to provide for multiple distributions
  - p = probability of required event | This can be array - to provide for multiple distributions
-

Data Science Interview Questions



Basic Data Science Questions:

- Difference between balanced and unbalanced datasets.
- Sampling bias.
- What is difference between SQL and MongoDB?
- Your favorite applications in Data Science! Mine were social/public sector projects. People will want to hire someone who is passionate about his/her skill and is not there just because every one else is learning it.
- Categorical data & Numeric Data

    - Numeric.
        - Countable
        - Measurable => consists an interval
    - Categorical data is of two types:
        - Ordinal => Indicating hierarchy between the categories. Eg: Movie Ratings, Class Grades
        - Nominal => No hierarchy. Eg: Aadhar numbers,
- 3 ways of scaling data:
    - Normalisation: Data becomes very sensitive to outliers.
    - Min-Max: Min to 0; Max to 1. Data becomes very sensitive to outliers.
    - Standardisation: convert to Z-Scores. This is good, but few ML algos need input within certain range like 0-1 which is not possible here

Excel:

- Learn VLOOKUP and HLOOKUP
- Difference between INDEX MATCH and VLOOKUP
- Pivot tables
- Data cleaning => Learn string formatting, concatenation etc. => This applies to Python as well.

SQL:

- Good understanding of basic syntax => I was asked questions on ALTER, INSERT INTO, WHERE, CREATE,
- Learn Partitioning and Window Functions like RANK() etc
- Learn and practice using **GROUPBY** and **HAVING** commands.