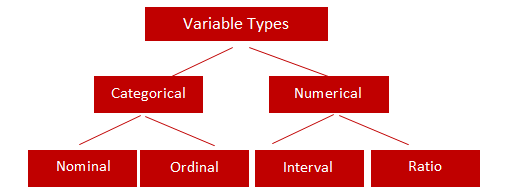
Variable types:

[](http://3.bp.blogspot.com/-L7KUapFGUXs/U001UGoJSFI/AAAAAAAACVw/8vH0VNUcuJ0/s1600/data+type.png)

Qualitative data are categorical variable

Nominal:-variable that has 2 or more type without any ordering.

Gender-Male/female

Marital status:Unmarried,married,divorcee

Ordinal variable:- having order

Interval:- Temp 10c to 40c

Measure of Central tendency:

Describes whole set of data with a single value that represents center of its distribution.

1.Mean (5,6,6,8,9,9,9,9,10,10 is (5+6+6+8+9+9+9+9+10+10)/10 = 8.1)

Affected by extreme values

Very large or small value can distort the answer

2.Median

It is the middle value. It splits the data in half. Half of the data are above the median; half of the data are below the median.

It is NOT affected by extreme values. Very large or very small numbers does not affect it

3.Mode

It is the value that occurs most frequently in a dataset.

When to use mean, median and mode:-

Mean – When your data is not skewed i.e normally distributed. In other words, there are no extreme values present in the data set (Outliers).

Median – When your data is skewed or you are dealing with ordinal (ordered categories) data (e.g. like scale 1. Strongly dislike 2. Dislike 3.Neutral 4. Like 5. Strongly like)

Mode - When dealing with nominal (unordered categories) data

Ex:

In real life, suppose a company is considering expanding into an area and is studying the size of containers that competitors are offering. They would be more interested in the mode because they want to know what size tends to sell most often.

Measure of Dispersion:

It refers to the spread or dispersion of scores. There are four main measures of variability

Range  
It is simply the largest observation minus the smallest observation.

Advantage:    
It is easy to calculate.

Disadvantage:   
It is very sensitive to outliers and does not use all the observations in a data set.

Standard Deviation  
 It is a measure of spread of data about the mean.

Advantage :

It gives a better picture of your data than just the mean alone.

Disadvantage :

1. It doesn't give a clear picture about the whole range of the data.  
2. It can give a skewed picture if data contain outliers.

Skewness  
It is a measure of symmetry. A distribution is symmetric if it looks the same to the left and right of the center point.

Kurtosis  
It is a measure of whether the data are peaked or flat relative to the rest of the data. Higher values indicate a higher, sharper peak; lower values indicate a lower, less distinct peak.

Examples:

**Example 1:** Suppose you are asked to calculate the average asset value of top stock funds and check whether there is any variability in the assets of these stock funds. You would answer this question with a measure of central tendency and variability.

**Example 2:**Suppose you are asked to provide a figure that best describes the annual salary offered to students in ABC College. You would answer this question with a measure of central tendency and variability.

Machine Learning:

computers the ability to learn without being explicitly programmed.

study and construction of algorithms that can learn from and make prediction on data

Related to

1.computation statistics

2.Mathematical optimization

3.data mining

4.predictive analytics

A computer program is said to learn from experience E

with respect to some class of tasks T and performance measure P

if its performance at tasks in T, as measured by P, improves with experience E.

Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible.

Types:

1.supervised learning - learn a general rule that maps inputs to outputs

2.Unsupervised Learning - No labels are given to the learning algorithm, leaving it on its own to find structure in its input

3.Semisupervised learning

4.Reinforcement learning - A computer program interacts with a dynamic environment in which it must perform a certain goal

Another Categorization:

1.Classification – inputs are divided into 2 or more classes (spam filtering,Cancer malignant or Non- malignant). Classes are known already

-supervised way

2.Regression-output are continuous rather than discrete (Housing prices)

3.Clustering-set of inputs divided into groups, Unsupervised learning

Groups are not known before head.

4.Density estimation – distribution of input is some spaces

5.Dimensionality reduction -simplifies input by mapping them into lower dimensional spaces.

Reduces the feature vectors.

Supervised Learning:

learn a general rule that maps inputs to outputs.

h(x1,x2,x3,x4) = a0+a1x1+a2x3^2+a3x3x4+a4x3x4+..

in majority of supervised learning the goal is to develop finely tuned predictor h(x) called as hypothesis.

This function takes input in four dimensions and has a variety of polynomial terms. Deriving a normal equation for this function is a significant challenge. Many modern machine learning problems

take thousands or even millions of dimensions of data to build predictions using hundreds of coefficients.

Predicting how an organism’s genome will be expressed, or what the climate will be like in fifty years,

are examples of such complex problems.

Gradient Descent - Minimizing “Wrongness”

The wrongness measure is known as the cost function.

1.Generalized Linear model: -

\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p

w = (w_1,
..., w_p) as coef\_ and w_0 as intercept\_.

**>>> from** **sklearn** **import** linear\_model

**>>>** reg = linear\_model.LinearRegression()

**>>>** reg.fit ([[0, 0], [1, 1], [2, 2]], [0, 1, 2])

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=1, normalize=False)

**>>>** reg.coef\_

array([ 0.5, 0.5])

2. Logistic regression: -

Can take value either 0 to 1 or between 0 to 1.

Eg: Pass/fail,win/lose

If multiple output between 0 to 1, then multinomial regression

T = a0+a1x

F(x) = 1/1+e~-(a0+a1x)

Polynomial regression:-

\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2

This preprocessor transforms an input data matrix into a new data matrix of a given degree. It can be used as follows:

**>>> from** **sklearn.preprocessing** **import** PolynomialFeatures

**>>> import** **numpy** **as** **np**

**>>>** X = np.arange(6).reshape(3, 2)

**>>>** X

array([[0, 1],

[2, 3],

[4, 5]])

**>>>** poly = PolynomialFeatures(degree=2)

**>>>** poly.fit\_transform(X)

array([[ 1., 0., 1., 0., 0., 1.],

[ 1., 2., 3., 4., 6., 9.],

[ 1., 4., 5., 16., 20., 25.]])

Data Exploration:-

Understand the problem

Univariable study

Multivariate study

Basic cleaning. - We'll clean the dataset and handle the missing data, outliers and categorical variables.

Used library:-

Numpy,scipy,Matplotlib,Pandas,Scikit-learn,statsmodel,seaborn,

Requests- for accessing the web. It works similar to the the standard python library urllib2

OS- for operating system and file operation

RE -regular expression

BeautifulSoup – for webscrapping.

For exploratory data analysis :-

Load the dataset using pandas library

Df.read\_csv/read\_excel/read\_table/read\_clipboard

Read data from clip board is useful converting tables from webpages.

Df.read\_csv(‘filename’,sep = ‘\t’)

Convert character date to Date python way.

Df.head(10)

Df.describe()

Df[‘field name’].value\_count() – it will count the

Describe function provides count , mean, std.min.max etc.

Distribution analysis:-

Boxplot and histogram directly can be ploted on pandas.Matplot lib is not required.

Draw histogram

df['ApplicantIncome'].hist(bins=50)

draw box plot

df.boxplot(column='ApplicantIncome')

Categorical variable analysis:-

Pivot table

Label encoder and one hot encoder

Avoid missing values

Data munging:- after cleaning the data we found some missing value . so we need to fill that .

Preprocessing of DATA:

Standardization = x -Mean(x)/std(x)

Normalization = x -min(x)/max(x) – min(x)

Standardization of data means zero mean and unit variance.

**from** **sklearn** **import** preprocessing

**>>> import** **numpy** **as** **np**

**>>>** X\_train = np.array([[ 1., -1., 2.],

**...**  [ 2., 0., 0.],

**...**  [ 0., 1., -1.]])

X\_scaled = preprocessing.scale(X\_train)

**>>>** X\_scaled

array([[ 0. ..., -1.22..., 1.33...],

[ 1.22..., 0. ..., -0.26...],

[-1.22..., 1.22..., -1.06...]])

Zero mean and unit variance

**>>>** X\_scaled.mean(axis=0)

array([ 0., 0., 0.])

**>>>** X\_scaled.std(axis=0)

array([ 1., 1., 1.])

**Standard scaler:-**

scaler = preprocessing.StandardScaler().fit(X\_train)

scaler.transform(X\_train)

array([[ 0. ..., -1.22..., 1.33...],

[ 1.22..., 0. ..., -0.26...],

[-1.22..., 1.22..., -1.06...]])

For x\_test you do not have to fit and transform. Only you need to do transform.

**>>>** X\_test = [[-1., 1., 0.]]

**>>>** scaler.transform(X\_test)

array([[-2.44..., 1.22..., -0.26...]])

scaling feature to a range:-

[MinMaxScaler](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler) or [MaxAbsScaler](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html#sklearn.preprocessing.MaxAbsScaler)

Here is an example to scale a toy data matrix to the [0, 1] range:

>>>

**>>>** X\_train = np.array([[ 1., -1., 2.],

**...**  [ 2., 0., 0.],

**...**  [ 0., 1., -1.]])

**...**

**>>>** min\_max\_scaler = preprocessing.MinMaxScaler()

**>>>** X\_train\_minmax = min\_max\_scaler.fit\_transform(X\_train)

**>>>** X\_train\_minmax

array([[ 0.5 , 0. , 1. ],

[ 1. , 0.5 , 0.33333333],

[ 0. , 1. , 0. ]])

**>>>** X\_test = np.array([[ -3., -1., 4.]])

**>>>** X\_test\_minmax = min\_max\_scaler.transform(X\_test)

**>>>** X\_test\_minmax

array([[-1.5 , 0. , 1.66666667]])

Normalization:-

The function [**normalize**](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html#sklearn.preprocessing.normalize) provides a quick and easy way to perform this operation on a single array-like dataset, either using the l1 or l2 norms:

>>>

**>>>** X = [[ 1., -1., 2.],

**...**  [ 2., 0., 0.],

**...**  [ 0., 1., -1.]]

**>>>** X\_normalized = preprocessing.normalize(X, norm='l2')

**>>>** X\_normalized

array([[ 0.40..., -0.40..., 0.81...],

[ 1. ..., 0. ..., 0. ...],

[ 0. ..., 0.70..., -0.70...]])

Feature Binarization: -

Convert numerical feature to Boolean value;

**>>>** X = [[ 1., -1., 2.],

**...**  [ 2., 0., 0.],

**...**  [ 0., 1., -1.]]

**>>>** binarizer = preprocessing.Binarizer().fit(X) *# fit does nothing*

**>>>** binarizer

Binarizer(copy=True, threshold=0.0)

**>>>** binarizer.transform(X)

array([[ 1., 0., 1.],

[ 1., 0., 0.],

[ 0., 1., 0.]])

Encoding categorical feature:-

Label encoder: convert into numerical feature.

To create dummy variable use one hot encoder.It is required to create dummy variable and exclude one column for modeling.

Sometimes if you scale the dummy variable you lose the data interpretation.

Onehot encoder: convert one hotencoder to many 0-1 columns.

Handle Missing value: -

The [Imputer](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Imputer.html#sklearn.preprocessing.Imputer) class provides basic strategies for imputing missing values, either using the mean, the median or the most frequent value of the row or column in which the missing values are located. This class also allows for different missing values encodings.

**strategy** : string, optional (default=”mean”)

The imputation strategy.

* If “mean”, then replace missing values using the mean along the axis.
* If “median”, then replace missing values using the median along the axis.
* If “most\_frequent”, then replace missing using the most frequent value along the axis.

**axis** : integer, optional (default=0)

The axis along which to impute.

* If axis=0, then impute along columns.
* If axis=1, then impute along rows.
* **missing\_values** : integer or “NaN”, optional (default=”NaN”)
* The placeholder for the missing values. All occurrences of missing\_values will be imputed. For missing values encoded as np.nan, use the string value “NaN”.

Generating Polynomial feature:-

Often it’s useful to add complexity to the model by considering nonlinear features of the input data

**>>>** poly = PolynomialFeatures(2)

**>>>** poly.fit\_transform(X)

array([[ 1., 0., 1., 0., 0., 1.],

[ 1., 2., 3., 4., 6., 9.],

[ 1., 4., 5., 16., 20., 25.]])

The features of X have been transformed from (X_1, X_2) to (1, X_1, X_2, X_1^2, X_1X_2, X_2^2).

Custom transformer:-

Often, you will want to convert an existing Python function into a transformer to assist in data cleaning or processing.

**>>> import** **numpy** **as** **np**

**>>> from** **sklearn.preprocessing** **import** FunctionTransformer

**>>>** transformer = FunctionTransformer(np.log1p)

**>>>** X = np.array([[0, 1], [2, 3]])

**>>>** transformer.transform(X)

**What is the need of fit and transform in ML:**

**the "fit()" function will find the mean and standard deviation of a variable set. The "transform()" function will apply and conversions (like "fit") to a data set**

**Regression:-**

Linear regression:

Y = b0+b1x

Y = dependant variable

X= Independent variable

Sum (Y\_a – Yp)^2 should be minmimum

|Ya – Yp| is less accurate than square of differences.

Independent variable is a matrix and dependent variable is vector.

If we do not assign a random state, python will not assign one for use, and we will get a random set every time. This is why it's important to set a random state.  So assign random\_state = 0.

Mean squared error and r2 in regression:-

MSE should be higher on test data and less on training data.Ideal case it should be near to Zero

R2 has value between 0-1. -ve value points the model is poorly fitted. 0 also poorly fitted.

**>>> from** **sklearn** **import** linear\_model

**>>>** reg = linear\_model.LinearRegression()

reg.fit ([[0, 0], [1, 1], [2, 2]], [0, 1, 2])

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=1, normalize=False)

N\_job = -1 means all CPU are used.

Normalize = whether you want to normalize the input

Ridge regression: -

**>>> from** **sklearn** **import** linear\_model

**>>>** reg = linear\_model.Ridge (alpha = .5)

**>>>** reg.fit ([[0, 0], [0, 0], [1, 1]], [0, .1, 1])

Ridge(alpha=0.5, copy\_X=True, fit\_intercept=True, max\_iter=None,

normalize=False, random\_state=None, solver='auto', tol=0.001)

pass alpha between 0 to 1.

Alpha is the measure of regularization.

 Regularization improves the conditioning of the problem and reduces the variance of the estimates

Larger values specify stronger regularization

controls the amount of shrinkage.

Solver - {‘auto’, ‘svd’, ‘cholesky’, ‘lsqr’, ‘sparse\_cg’, ‘sag’, ‘saga’}

Lasso:-

**>>> from** **sklearn** **import** linear\_model

**>>>** reg = linear\_model.Lasso(alpha = 0.1)

**>>>** reg.fit([[0, 0], [1, 1]], [0, 1])

Lasso(alpha=0.1, copy\_X=True, fit\_intercept=True, max\_iter=1000,

normalize=False, positive=False, precompute=False, random\_state=None,

selection='cyclic', tol=0.0001, warm\_start=False)

**>>>** reg.predict([[1, 1]])

array([ 0.8])

Generally, when you have many small/medium sized effects you should go with ridge. If you have only a few variables with a medium/large effect, go with lasso

Elastic net:-

Linear regression with combined L1 and L2 priors as regularizer.

Important parametes:- alpha nd l1\_ratio.

with 0 <= l1\_ratio <= 1. For l1\_ratio = 0 the penalty is an L2 penalty. For l1\_ratio = 1 it is an L1 penalty. For 0 < l1\_ratio < 1, the penalty is a combination of L1 and L2.

Logistic Regression:- p(x) = 1/1+e^-y , varies between 0 to 1. But in classification we take as 0 or 1.

Choose a threshold value like 0.55. if predic\_probability greater than .55 then choose 1

If less than 0.55 then choose 0.

Logistic regression, despite its name, is a linear model for classification rather than regression

logreg = [linear\_model.LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)(C=1e5)

**penalty** : ‘l1’ or ‘l2’, default: ‘l2’

used to specify type of normalization

**C** : float, default: 1.0

Inverse of regularization strength;

**solver** : {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’},

**For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ and**

‘saga’ are faster for large ones.

**For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’**

**Polynomial regression:-**

**>>> from** **sklearn.preprocessing** **import** PolynomialFeatures

**>>>** X

array([[0, 1],

[2, 3],

[4, 5]])

**>>>** poly = PolynomialFeatures(degree=2)

**>>>** poly.fit\_transform(X)

array([[ 1., 0., 1., 0., 0., 1.],

[ 1., 2., 3., 4., 6., 9.],

[ 1., 4., 5., 16., 20., 25.]])

If we want to fit a paraboloid to the data instead of a plane, we can combine the features in second-order polynomials, so that the model looks like this:

**>>> from** **sklearn.pipeline** **import** Pipeline

**>>> import** **numpy** **as** **np**

**>>>** model = Pipeline([('poly', PolynomialFeatures(degree=3)),

**...**  ('linear', LinearRegression(fit\_intercept=**False**))])

SVM:-

Support vector machine

Used in classification, regression and outlier detection

Effective in high dimensional spaces.

Still effective in cases where number of dimensions is greater than the number of samples.

Use different kernel functions.

SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

**>>> from** **sklearn** **import** svm

**>>>** X = [[0, 0], [1, 1]]

**>>>** y = [0, 1]

**>>>** clf = svm.SVC()

**>>>** clf.fit(X, y)

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

SVC

Nusvc

LinearSVC

In SVM we create support vectors to maximize the data separation.

In multidimensional medium support vectors will be hyperplane which will create the decision bourdry.

**C** : float, optional (default=1.0) :Penalty parameter C of the error term.

**Kernel** : string, optional (default=’rbf’) : must be one of ‘linear’, ‘poly’, ‘rbf’

**degree** : int, optional (default=3)

Degree of the polynomial kernel function (‘poly’). Ignored by all other kernels.

**gamma** : float, optional (default=’auto’)

Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. If gamma is ‘auto’ then 1/n\_features will be used instead.

**decision\_function\_shape** : ‘ovo’, ‘ovr’, default=’ovr’

Whether to return a one-vs-rest (‘ovr’) decision function of shape (n\_samples, n\_classes) as all other classifiers, or the original one-vs-one (‘ovo’) decision function of libsvm which has shape (n\_samples, n\_classes \* (n\_classes - 1) / 2).

You can see the support vectors by below code.

clf.support\_vectors\_

**Setting C**: C is 1 by default and it’s a reasonable default choice. If you have a lot of noisy observations you should decrease it. It corresponds to regularize more the estimation.

Custom kernel can also be used in svr/svm.

Stochastic gradient descent:-

**>>> import** **numpy** **as** **np**

**>>> from** **sklearn** **import** linear\_model

**>>>** X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])

**>>>** Y = np.array([1, 1, 2, 2])

**>>>** clf = linear\_model.SGDClassifier()

**>>>** clf.fit(X, Y)

**...**

SGDClassifier(alpha=0.0001, average=False, class\_weight=None, epsilon=0.1,

eta0=0.0, fit\_intercept=True, l1\_ratio=0.15,

learning\_rate='optimal', loss='hinge', max\_iter=None, n\_iter=None,

n\_jobs=1, penalty='l2', power\_t=0.5, random\_state=None,

shuffle=True, tol=None, verbose=0, warm\_start=False)

>>>

**>>>** print(clf.predict([[-0.8, -1]]))

[1]

Naïve Bayes:-

They require a small amount of training data to estimate the necessary parameters

Can be used in document classification and spam filtering.

Suitable when many data points with few features.  
 - Uses Bayes theorem  
 - Presence of particular feature unrelated to presence of other feature  
 - Features are independent ,so it is called as Naïve

GaussianNB;

MultinomialNB:

BernouliNB:

Class sklearn.Naive\_bayes.MultinimialNB(alpha = 1)

Alpha = Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

Bernoulli: The binomial model is useful if your feature vectors are binary (i.e. zeros and ones)  
- One application would be text classification with ‘bag of words’ model.

 - where the 1s & 0s are “word occurs in the document” and “word does not occur in the document” respectively.

 - P(c|x) = P(x|c)\*P(c)/P(x)

 - P(x|c) = likelihood  
 - P(c) = class of prior probability  
 - P(x) = predictor prior probability

Naive Bayes models can be used to tackle large scale classification problems for which the full training set might not fit in memory. To handle this case, [**MultinomialNB**](http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB), [**BernoulliNB**](http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB), and [**GaussianNB**](http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB) expose a **partial\_fit** method that can be used incrementally

Decision Tree:-

 Non parametric algorithm in scikit learn.  
 - Requires little data preparation  
 - however that this module does not support missing values.  
 - The cost of predicting data is logarithmic in the number of data points used to train the tree  
 - Able to handle both numerical and categorical data  
 - Able to handle multi-output problems  
 - Uses a white box model. Given situation is observable and interpretable in a model   
 - they map non-linear relationships quite well.  
 - can be used in classification and regression  
 - scikit-learn uses an optimized version of the CART algorithm

**>>> from** **sklearn** **import** tree

**>>>** X = [[0, 0], [1, 1]]

**>>>** Y = [0, 1]

**>>>** clf = tree.DecisionTreeClassifier()

**>>>** clf = clf.fit(X, Y)

How to do the splitting and adding of nodes.

Gini Index:

Gini index says, if we select two items from a population at random then they must be of same class  
   and probability for this is 1 if population is pure.

CART (Classification and Regression Tree) uses Gini method to create binary splits.

Chi -squared.

 - Chi-square = ((Actual – Expected)^2 / Expected)^1/2

Entropy: degree of disorganization in a system known as Entropy

Information gain = 1 - entropy.

**criterion** : string, optional (default=”gini”)

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

**max\_depth** : int or None, optional (default=None)

**min\_samples\_split** : int, float, optional (default=2)

**min\_samples\_leaf** : int, float, optional (default=1)

**max\_leaf\_nodes** : int or None, optional (default=None)

Compared with entropy, gini impurity should be computed slightly faster because of the absence of *log* computation and this could be a reason why gini impurity is the default method in some machine learning packages.

Ensemble Model:-

 ensemble means Group  
 - Multiple models group together and bring forth the model that is more accurate.  
 - Ensemble methods are techniques that create multiple models and then combine   
   them to produce improved results.

 - Bagging: making models in parallel.  
 - Boosting: Making models in series or sequential.  
 - In each successive model, the weights are adjusted based on learning of previous model.

Random Forest:-

**from** **sklearn.ensemble** **import** RandomForestClassifier

clf = RandomForestClassifier(n\_estimators=10)

Extremely randomized tree:

clf = ExtraTreesClassifier(n\_estimators=10, max\_depth=**None**,

**...**  min\_samples\_split=2, random\_state=0)

Adaboost:-

**from** **sklearn.ensemble** **import** AdaBoostClassifier

Gradient Boosting:

Parameters:

Loss: {‘deviance’, ‘exponential’}, optional (default=’deviance’)

Learning rate : learning rate shrinks the contribution of each tree by learning\_rate. There is a trade-off between learning\_rate and n\_estimators.

N\_esitmator: The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.

Max\_depth: maximum depth of the individual regression estimators(min =3)

Criteriaon: freedman\_mse

Min\_sample\_split:

Min\_sample\_leaf:

Min\_weight\_fraction\_leaf:

Max\_leaf\_node

**from** **sklearn.ensemble** **import** GradientBoostingClassifier

XGBOOST:-

Voting classifier:-

**VotingClassifier** is to combine conceptually different machine learning classifiers and use a majority vote or the average predicted probabilities (soft vote) to predict the class labels.

clf1 = LogisticRegression(random\_state=1)

**>>>** clf2 = RandomForestClassifier(random\_state=1)

**>>>** clf3 = GaussianNB()

**>>>** eclf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)], voting='hard')

**>>> for** clf, label **in** zip([clf1, clf2, clf3, eclf], ['Logistic Regression', 'Random Forest', 'naive Bayes', 'Ensemble']):

**...**  scores = cross\_val\_score(clf, X, y, cv=5, scoring='accuracy')

Multiclass: - divide set of fruit to oranges,apples etc.classification with more than 2 class.

Multi label: -A set of label can be assigned to a group.

Orange can be fruit,not vegitable, veg category people eat it. This fruit->orange + fruit+veg people.

Neural Network model:-

Input layer-> hidden layer – >activation function->output layer

Weights adjusting

Activation function :- threshold function f(x) = 1 or 0

Sigmoid function f(x) = 1+1/1+e^-x here value will be 0 or 1. But in addition it will tell the probability of 0 or 1. As it is smooth curve.

It is smooth. Useful in output layer

Rectifier function f(x) = max(x,0)

Hyperbolic tangent function

Hidden layer apply RELU and in output use sigmoid function

How neural network learn:

Initialize any weight, then compare the output value with actual value, then again adjust the weight.

Repeated the process until the error is minimized. Error is called cost function

Through back propagation the neural network learn by adjusting the weight.

**Gradient descent: -**

How the weight are adjusted, SGD will answer that.

Cost function expected to be convex

Take all the row as batch, run NN, look as cost function, adjust the weights.

**Stochastic gradient descent:**

Finding a global minimum

Take each row one by one, run the NN look at cost function, and adjust the weight.

Faster. Don’t have to Load up all the data like BGD(batch gradient descent)

Forward propagation:

Backpropagation:

GPU and CPU,

GPU is more powerful, floating point calculation, many cores

Keras:-

Import keras

Import keras.models import sequential

From keras.layers import Dense

Classifier = sequential()

Classifier.add(Dense(output\_dim = 6,init = ‘uniform’,activation = ‘relu’,input = 11))

As a convention take output layer as input/2 nodes.

Classifier.compile( optimizer = ‘adam’,loss = ‘binary\_cross entropy’,metrics = [‘accuracy’])

Loss : categorical\_cross\_entropy-> if more than 2 class.

Classifer.fit(x\_train,y\_train,batch\_size = 10,nb\_epoch = 100)

**Multi-layer Perceptron:**

**MLPClassifier**

**MLPRegressor**

**For multiclass classification use softmax model**

**Parameters:**

**Hidden\_layer\_size:default 100**

**Activation function:** {‘identity’, ‘logistic’, ‘tanh’, ‘relu’}

**Activation function for hidden layer**

**Identity – for linear f(x) = x**

**Logistic- for logistic sigmoid function f(x) = 1/1+e^-x**

**Tanh – for hyperbolic function**

**Relu -rectifier linear unit function.**

**Solver: for weight optimization,default is adam**

**Lbfgs -**  For small datasets,‘lbfgs’ can converge faster and perform better

Second order partial derivative of function

**Sgd -stochastic gradient descent**

**Adam -** adam’ works pretty well on relatively large datasets

**Alpha**: default 0.0001, l2 regularization parameter

avoiding overfitting by penalizing weights with large magnitudes

**Learning\_rate:**

**learning\_rate\_init** : double, optional, default 0.001

Multi-layer Perceptron is sensitive to feature scaling, so it is highly recommended to scale your data

**Convolution NN:-**

**Black and white image in 2d pixel**

**Color image is 3d BGR**

**1Convolution Layer:-**

**Input image -> feature vector example(3\*3) = Feature map**

**Slide the feature vector and element wise multiplication.**

**Many convolution layers are created.**

**2Relu Layer**

**We want to increase non linearity, because images are non linear.**

**3Pooling: maxpooling/min pooling/average pooling**

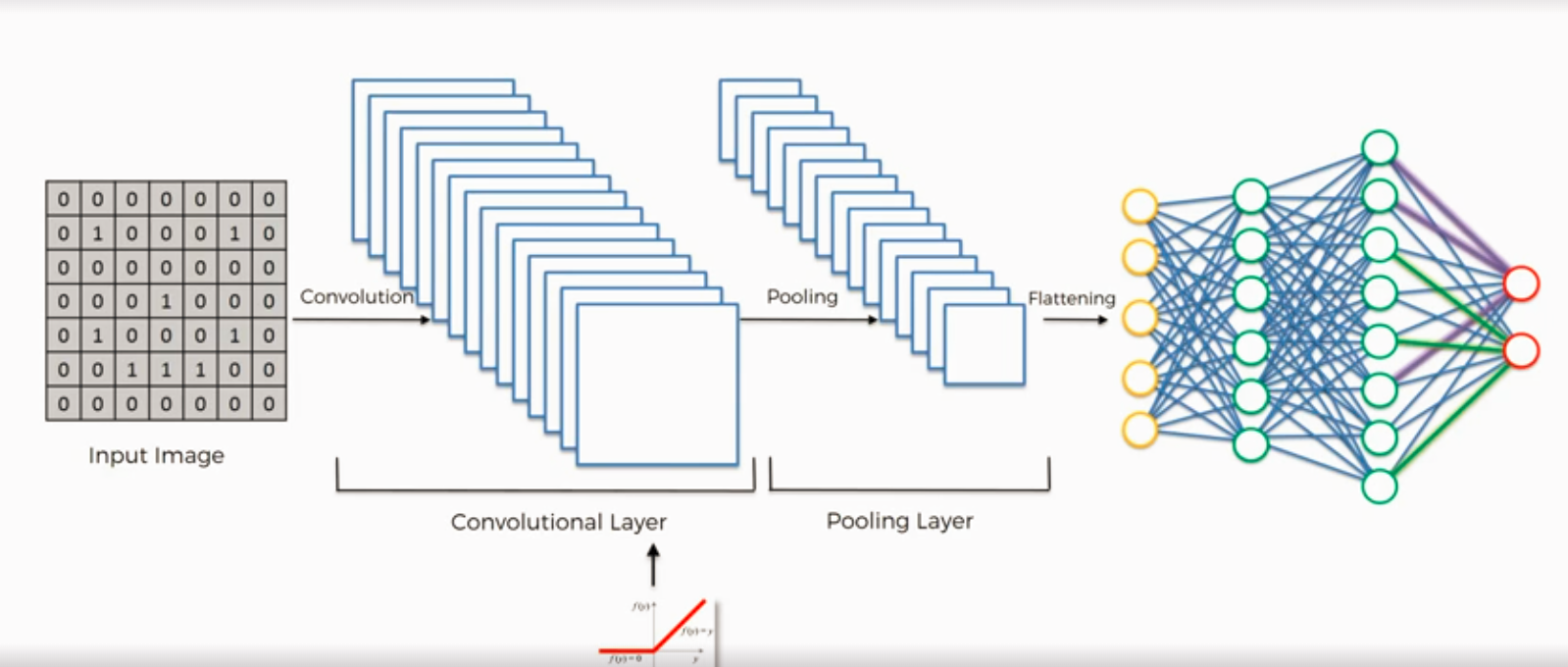
**After pooling also, we have preserve the feature by reducing the information/parameters**

**4Flattening:**

**Flat the pooled feature map.**

**5Fullyconnected layer**

**Create the hidden layer in ANN from the flattening as input.**



**SoftMax and Cross Entropy: -**

**Binary – sigmoid activation function**

**Before applying the softmax the value will be anything and if we add it hasn’t to be 1.**

**So we apply softmax to calculate the percentage and add up to 1.**

**Dimensionality Reduction: -**

**2 types of dimensionalty reduction technique**

**1.Feature selection:- Backward elimination, Forward selection, Bidirectional Elimination, Score comparison,**

**2.Feature extraction**

**PCA**

**LDA**

**Kernel PCA**

**QDA**

**PCA :- from m independent variable of our dataset,PCA extracts p<= m new independent variable that explain the most of the variance dataset regardless of the dependent variable.**

**So DV is not considered in PCA makes it a unsupervised model.**

**Restore the variance with less features. Can be visualize in a graph.**

**Before PCA you may scale your data.**

**From sklearn.decomposition import PCA**

**Pca = PCA(n\_components = None)**

**X\_train = pca.fit\_transform(X\_train)**

**X\_test = pca.transform(X\_test)**

**Explained\_variance = pca.explained\_variance\_ratio\_**

**It will show column wise variance. You can choose how much percentage you want to restore with no of columns.**

**Then set n\_components accordingly .(2,3,4 etc)**

**Linear discriminate analysis** :- and quadratic discriminate analysis.

Considers the dependent variable while extracting the data.

So it is supervised learning.

From n independent variable of our dataset LDA extracts p<=n new independent variable without changing the properties of the data or independent variable.

So that it can separate most of the classes of dependent variable.

Basically, you should select enough components to explain enough of the variance of the data to be fit .

It is a Dimensionality reduction technique

**from** **sklearn.discriminant\_analysis** **import** LinearDiscriminantAnalysis as LDA

lda = LDA(n\_componenets = 2)

x\_train = lda.fit\_transform(X\_train,y\_train)

x\_test = lda.transform(x\_test)

here we have choosed compoents = 2.

The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix.

**Solver: svd – singular value decomposition**

**Does not compute the covariance matrix, therefore this solver is recommended for data with a large number of features.**

**‘eigen’: Eigenvalue decomposition, can be combined with shrinkage**

**lsqr’: Least squares solution, can be combined with shrinkage.**

**n\_components** : int, optional

no of component for dimensionality reduction.

**Kernel PCA:-**

**Non- linearly separable**

**Use kernel trick.(linear,poly,rbf,sigmoid,cosine,precomputed)**

**Rbf is gaussian kernel.**

**R squares:-**

**SS-res = SUM (Yi – Yi^)**

**SS-tot = SUM(Yi-Yi^)**

**Rsqaured = 1 – SS-res/SS-tot**

**Adjusted r-squared:-**

**Kernel:-**

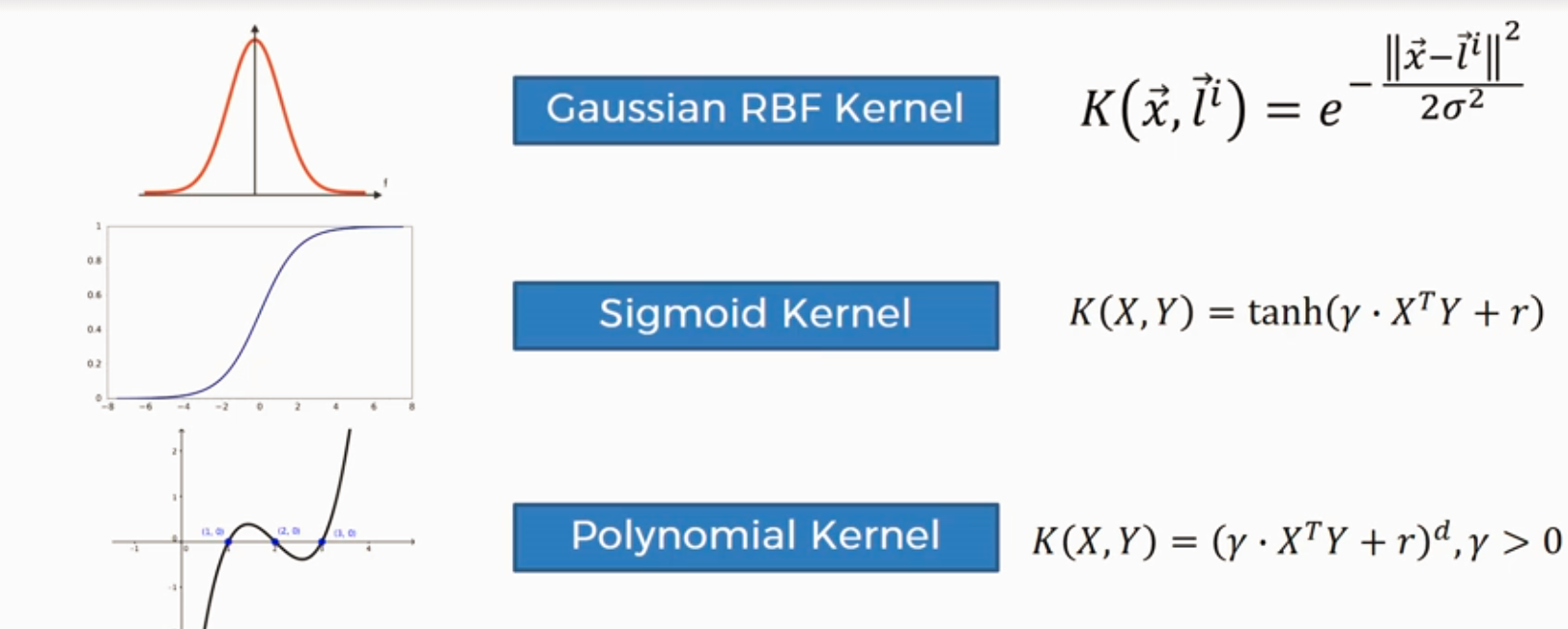
Assume that your input data is not a linearly separable dataset in the feature space.

A kernel helps to transform this data into a high-dimensional space where this is actually linearly separable.

**Kernel is a similarity function,takes input ans spits out how similar they are.**

**Every kernel function can be expressed as dot product in a infinite dimensional feature space.**

**Many machine learning algorithms can be expressed entirely in terms of dot products.**



**Request library:-**

**It deals with HTTP contents.**

Import requests

*r = request.get(‘https://github.com/timeline.json’)*

**NLP:-**

**NLTK library.**

**Copora: body of text**

**Lexicon: words and their meanings**

**Token:-**  Each "entity" that is a part of whatever was split up based on rules.

Each word is a token when sentence is lokenized

Each sentence is a token when a paragraph is tokenized.

**Tokenizing: -** Splitting sentences and words from the body of text.

**Remove stop words:-**

**from nltk.corpus import stopwords**

**stopwords.words('english')**

**Tokenizing: - sentence tokenizing and word tokenizing**

**From nltk.tokenize imprt sent\_tokenize.word\_tokenize**

**Example\_test = “ ”**

**Time series forecasting:-**

**Confusion matrix:**

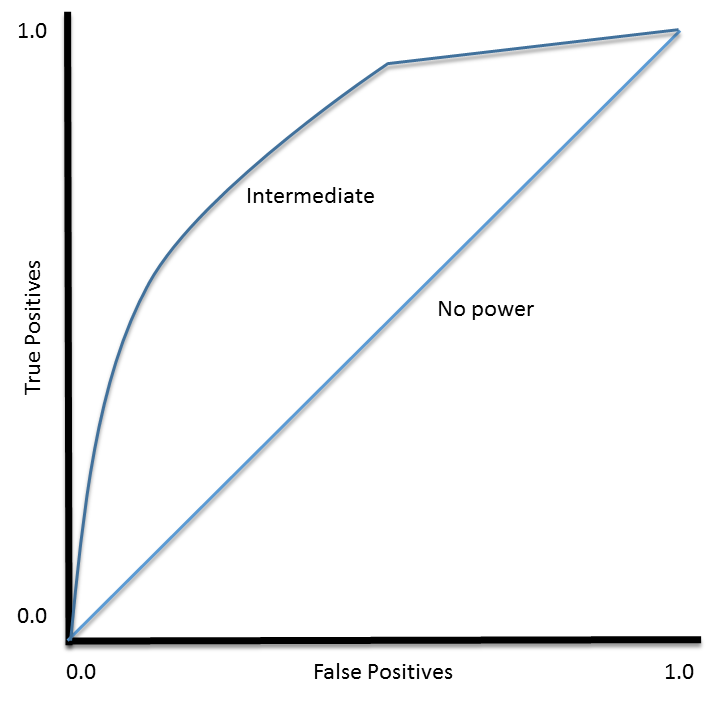
**correct prediction = True Negative + True Positive**

**incorrect prediction = False Positive + False Negative**



**ROC :- Receiver operating characteristics**

TP (sensitivity) can then be plotted against FP (1 – specificity) for each threshold used. The resulting graph is called a Receiver Operating Characteristic (ROC) curve

\

ROC curve can be used to select a **threshold** for a classifier which maximizes the true positives, while minimizing the false positives.

**Regularization:-**

Process of adding Norm to our cost function is called regularization

Regularization is the process of adding a tuning parameter to a model to induce smoothness in order to prevent [**overfitting**](https://en.wikipedia.org/wiki/Overfitting)**/underfitting.**

Regularization becomes necessary when the model begins to ovefit / underfit. This technique introduces a cost term for bringing in more features with the objective function. Hence, it tries to push the coefficients for many variables to zero or add penalty and hence reduce cost term. This helps to reduce model complexity so that the model can become better at predicting (generalizing).

L2 regularization is more efficient than L1.

L2 Norm – Ridge regresson

L1 Norm – Lasso regression

**You are given a train data set having 1000 columns and 1 million rows. The data set is based on a classification problem. Your manager has asked you to reduce the dimension of this data so that model computation time can be reduced. Your machine has memory constraints. What would you do?**

**Use generator in python for data operation**

**Remove correlated variables**

**Use PCA for dimensionality reduction**

**Use n\_jobs = -1 for parallel processing.**

**Q3.** **You are given a data set. The data set has missing values which spread along 1 standard deviation from the median. What percentage of data would remain unaffected? Why?**

**Answer:** This question has enough hints for you to start thinking! Since, the data is spread across median, let’s assume it’s a normal distribution. We know, in a normal distribution, ~68% of the data lies in 1 standard deviation from mean (or mode, median), which leaves ~32% of the data unaffected. Therefore, ~32% of the data would remain unaffected by missing values.

Lasso vs Ridge Regression:-

 in presence of few variables with medium / large sized effect, use lasso regression. In presence of many variables with small / medium sized effect, use ridge regression.

Data preprocessing tool:-

Trifacta and Paxata are becoming increasingly popular to perform the data cleansing jobs

Another alternative is pandas 'get\_dummies' function.

Check below code.

# Prepare feature and dependent variable along with categorical encoding  
X=pd.get\_dummies(data.loc[:, ['feature1', 'feature2']])  
y=pd.get\_dummies(data.loc[:, 'target'], drop\_first=True)

Train - test split of data should be there before Scaling-

The reason is that you should train your model on the training data, without using any information regarding the test data. If you apply PCA on the whole data (including the test data) before training the model, then you in fact use some information from the test data. Thus, you cannot really judge the behaviour of your model using the test data, because it is not an unseen data anymore.

**Normalization vs Standardization: -**

Normalization rescales the values into a range of [0,1]. This might be useful in some cases where all parameters need to have the same positive scale. However, the outliers from the data set are lost.

Xchanged=X−Xmin/Xmax−Xmin

Standardization rescales data to have a mean (μ) of 0 and standard deviation (σ) of 1 (unit variance).

Xchanged=X−μ/σ

For most applications standardization is recommended because more resistance to outlier.

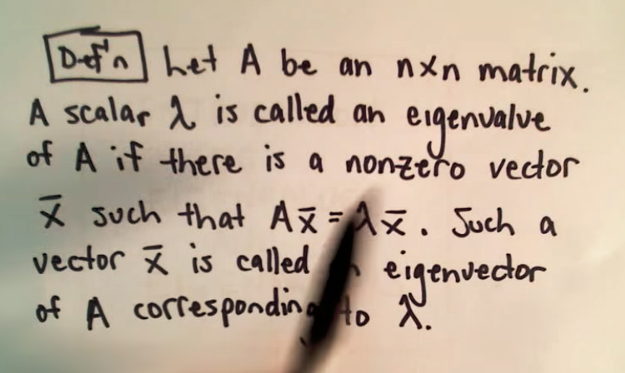
When we do standardization, most of the variance of the feature is between -1 and 1 and the mean is at 0.

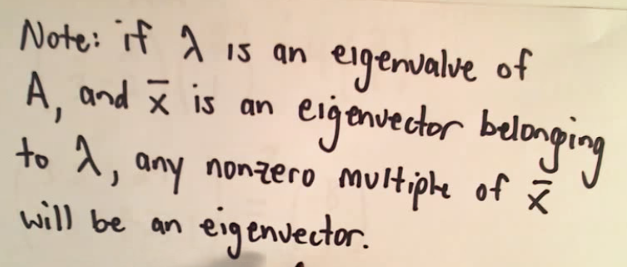
If you normalize the value the then the range should be between [0,1]

If you scale the data then it can be anything.

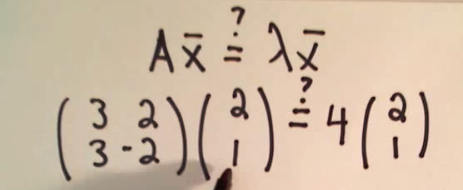
For Regression you may perform the feature scaling on both dependent(y) and independent(X) variable. For classifications FS is not required on Dependent(Y) variable.

 Eign vector: -





**Example: -**



**To avoid dummy variable trap for multiple categorical variable, exclude one column for each categorical variable.**

**P Value: - P value is a statistical measure that helps scientists determine whether or not their hypotheses are correct.**  **It is a probability of getting an unusual result. We set a significant level let say 0.05 for an outcome.**

**Then we calculate the P Value. P – Value should be less to fit the model correctly.**

**High P values: your data are likely with a true null.**

**Low P values: your data are unlikely with a true null.**

**Best way to put Date into numerical feature: -**

**you can take direct no. of days by using**

Sys.Date()- as.Date("2012-03-12")# calculate the days Time difference of 847 days

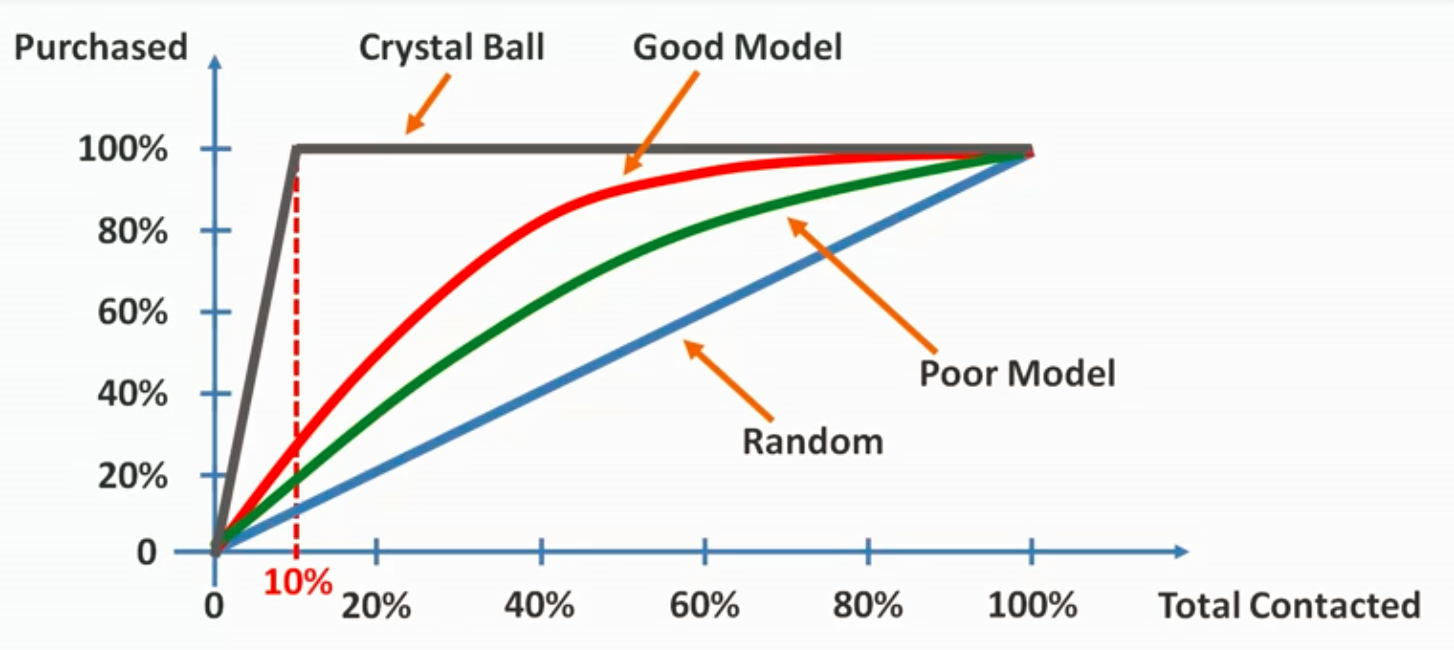
or in week format.

Null hypothesis: Variable has no effect and its coefficient in reality vanish.

When p-value is less than SL(Significant Level) we reject the null hypothesis and this implies that the variable does have some effect in prediction.

**Cumulative Accuracy profile: -CAP**

The CAP can be used to evaluate a classification model by comparing the curve to the perfect CAP in which the maximum number of positive outcomes is achieved directly and to the random CAP in which the positive outcomes are distributed equally. A good model will have a CAP between the perfect CAP and the random CAP with a better model tending to the perfect CAP.



It is different from ROC where in ROC curve is created by plotting the [true positive rate](https://en.wikipedia.org/wiki/True_positive_rate) (TPR) against the [false positive rate](https://en.wikipedia.org/wiki/False_positive_rate) (FPR) at various threshold settings.

Deep Learning needs GPU: why?:-

Inputs and weights need matrix multiplication with feed forward and backward neural network.

Performing Matrix multiplication simultaneously in CPU is a time consuming as it should happen one by one. Through GPU we can do it at a time. So it is faster.