**Data Preprocessing.**

For various reasons, many real-world datasets contain missing values, often encoded as blanks, NaNs or other placeholders. Such datasets however are incompatible with scikit-learn estimators which assume that all values in an array are numerical, and that all have and hold meaning. A basic strategy to use incomplete datasets is to discard entire rows and/or columns containing missing values. However, this comes at the price of losing data which may be valuable (even though incomplete). A better strategy is to impute the missing values, i.e., to infer them from the known part of the data.

One type of imputation algorithm is univariate, which imputes values in the i-th feature dimension using only non-missing values in that feature dimension (e.g. impute.SimpleImputer). By contrast, multivariate imputation algorithms use the entire set of available feature dimensions to estimate the missing values (e.g. impute.IterativeImputer).

**Univariate feature imputation**

The [**SimpleImputer**](https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html#sklearn.impute.SimpleImputer) class provides basic strategies for imputing missing values. Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located. This class also allows for different missing values encodings.

# SimpleImputer class is imported from sklearn.

# It is used for replacing missing data with desired values the dataset.

from sklearn.impute import SimpleImputer

# Strategy can be changed to mean as well. But if the data is skewed then median is prefered

imp = SimpleImputer(missing\_values = np.nan , strategy = 'median')

imp.fit(x[:,1:3])

x[:,1:3] = imp.transform(x[:,1:3])

**Encoding Categorical Data**

**Feature Scaling**

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.

**Example:** If an algorithm is not using the feature scaling method, then it can consider the value 3000 meters to be greater than 5 km but that’s actually not true and, in this case, the algorithm will give wrong predictions. So, we use Feature Scaling to bring all values to the same magnitudes and thus, tackle this issue.

Text, letter

Description automatically generated

Normalization is good to use when the distribution of data does not follow a Gaussian distribution. It can be useful in algorithms that do not assume any distribution of the data like K-Nearest Neighbors.

In Neural Networks algorithm that require data on a 0–1 scale, normalization is an essential pre-processing step. Another popular example of data normalization is image processing, where pixel intensities have to be normalized to fit within a certain range (i.e., 0 to 255 for the RGB color range).

Standardization can be helpful in cases where the data follows a Gaussian distribution. Though this does not have to be necessarily true. Since standardization does not have a bounding range, so, even if there are outliers in the data, they will not be affected by standardization.

In clustering analyses, standardization comes in handy to compare similarities between features based on certain distance measures. Another prominent example is the Principal Component Analysis, where we usually prefer standardization over Min-Max scaling since we are interested in the components that maximize the variance.

There are some points which can be considered while deciding whether we need Standardization or Normalization

* Standardization may be used when data represent Gaussian Distribution, while Normalization is great with Non-Gaussian Distribution
* Impact of Outliers is very high in Normalization

To conclude, you can always start by fitting your model to raw, normalized, and standardized data and compare the performance for the best results.

In order to apply Normalization or Standardization, we can use the prebuilt functions in scikit-learn or can create our own custom function.

# [**Why standardization of the testing set must be performed with the mean and standard deviation of the training set?**](https://stats.stackexchange.com/questions/202287/why-standardization-of-the-testing-set-has-to-be-performed-with-the-mean-and-sd)

When you center and scale a variable in the training data using the mean and sd of that variable calculated on the training data, you are essentially creating a brand-new variable. Then you are doing, say, a regression on that brand new variable.

To use that new variable to predict for the validation and/or test datasets, you have to create the same variable in those data sets. Subtracting a different number and dividing by a different number does not create the same variable.

Data leakage mainly occurs when some information from the training data is revealed to the validation data. In order to prevent the same, the point to pay attention to is to fit the scaler on the train data and then use it to transform the test data.

Normalization across instances should be done after splitting the data between training and test set, using only the data from the training set.

This is because the test set plays the role of fresh unseen data, so it's not supposed to be accessible at the training stage. Using any information coming from the test set before or during training is a potential bias in the evaluation of the performance.

Do Normalization after splitting into train and test/validation. The reason is to avoid any **data leakage**.

Data leakage is when information from outside the training dataset is used to create the model. This additional information can allow the model to learn or know something that it otherwise would not know and in turn invalidate the estimated performance of the mode being constructed.

**Dummy Variable in Regression Models:**

In statistics, especially in regression models, we deal with various kinds of data. The data may be quantitative (numerical) or qualitative (categorical). The numerical data can be easily handled in regression models, but we can’t use categorical data directly, it needs to be transformed in some way.

For transforming categorical attributes to numerical attributes, we can use the label encoding procedure (label encoding assigns a unique integer to each category of data). But this procedure is not alone that suitable, hence, ***one hot encoding*** is used in regression models following label encoding. This enables us to create new attributes according to the number of classes present in the categorical attribute i.e., if there are *n* number of categories in categorical attribute, *n* new attributes will be created. These attributes created are called ***Dummy Variables***. Hence, dummy variables are “proxy” variables for categorical data in regression models.   
These dummy variables will be created with one-hot*encoding* and each attribute will have a value of either 0 or 1, representing the presence or absence of that attribute.

**Dummy Variable Trap:**

The Dummy variable trap is a scenario where there are attributes that are highly correlated (Multicollinear) and one variable predicts the value of others. When we use*one-hot encoding* for handling the categorical data, then one dummy variable (attribute) can be predicted with the help of other dummy variables. Hence, one dummy variable is highly correlated with other dummy variables. Using all dummy variables for regression models leads to a ***dummy variable trap***. So, the regression models should be designed to exclude one dummy variable.

**For Example –**   
Let’s consider the case of gender having two values *male*(0 or 1) and *female*(1 or 0). Including both the dummy variable can cause redundancy because if a person is not male in such case that person is a female, hence, we don’t need to use both the variables in regression models. This will protect us from the dummy variable trap.

**Regression**

# **What is Regression?**

# Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent and independent variable. The above definition is a bookish definition, in simple terms the regression can be defined as, “Using the relationship between variables to find the best fit line or the regression equation that can be used to make predictions”.

Chart, scatter chart

Description automatically generated

# **Linear Regression**

In statistics, linear regression is a linear approach to modelling the relationship between a dependent variable and one or more independent variables. Let **X** be the independent variable and **Y** be the dependent variable. We will define a linear relationship between these two variables as follows:

Chart, scatter chart

Description automatically generated

This is the equation for a line where **m** is the slope of the line and **c** is the y intercept. We will use this equation to train our model with a given dataset and predict the value of **Y** for any given value of **X**. Challenge is to determine the value of **m** and **c**, such that the line corresponding to those values is the best fitting line or gives the minimum error.

**ŷ = b0 + b1x**

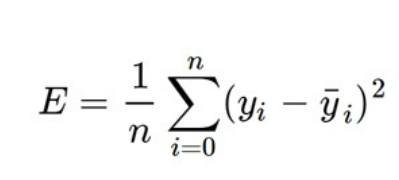
**b1 = Σ [ (xi - x)(yi - y) ] / Σ [ (xi - x)2]**

**b0 = y - b1 \* x**

# **Loss Function**

The loss is the error in our predicted value of **m** and **c**. Our goal is to minimize this error to obtain the most accurate value of **m** and **c**.  
We will use the Mean Squared Error function to calculate the loss. There are three steps in this function:

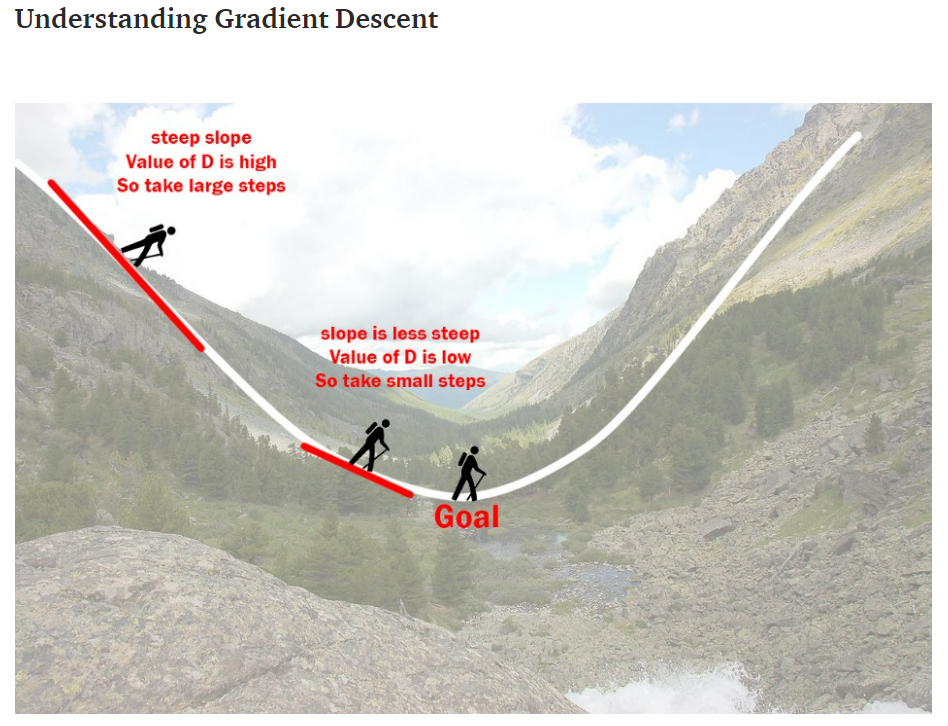
1. Find the difference between the actual y and predicted y value (y = mx + c), for a given x.
2. Square this difference.
3. Find the mean of the squares for every value in X.



Here yᵢ is the actual value and ȳᵢ is the predicted value. So, we square the error and find the mean. hence the name Mean Squared Error. Now that we have defined the loss function, let’s get into the interesting part — minimizing it and finding **m** and **c.**

# **The Gradient Descent Algorithm**

Gradient descent is an iterative optimization algorithm to find the minimum of a function. Here that function is our Loss Function. It’s used to minimize the loss function.



Imagine a valley and a person with no sense of direction who wants to get to the bottom of the valley. He goes down the slope and takes large steps when the slope is steep and small steps when the slope is less steep. He decides his next position based on his current position and stops when he gets to the bottom of the valley which was his goal.

Let’s try applying gradient descent to **m** and **c** and approach it step by step:

**1.**Initially let m = 0 and c = 0. Let L be our learning rate. This controls how much the value of **m** changes with each step. L could be a small value like 0.0001 for good accuracy.

**2.**Calculate the partial derivative of the loss function with respect to m, and plug in the current values of x, y, m and c in it to obtain the derivative value **D**.

Text, letter

Description automatically generated

Dₘ is the value of the partial derivative with respect to **m**. Similarly let’s find the partial derivative with respect to **c**, Dc:

Text

Description automatically generated with medium confidence

**3.**Now we update the current value of **m** and **c** using the following equation:

Text

Description automatically generated with medium confidence

**4.**We repeat this process until our loss function is a very small value or ideally 0 (which means 0 error or 100% accuracy). The value of **m** and **c** that we are left with now will be the optimum values.

Now going back to our analogy, **m** can be considered the current position of the person. **D** is equivalent to the steepness of the slope and **L** can be the speed with which he moves. Now the new value of **m** that we calculate using the above equation will be his next position, and **L×D** will be the size of the steps he will take. When the slope is steeper (**D** is more) he takes longer steps and when it is less steep (**D** is less), he takes smaller steps. Finally, he arrives at the bottom of the valley which corresponds to our loss = 0.  
Now with the optimum value of **m** and **c** our model is ready to make predictions!

Gradient descent is one of the simplest and widely used algorithms in machine learning, mainly because it can be applied to any function to optimize it.

Multiple Linear Regression

Regression models are used to describe relationships between variables by fitting a line to the observed data. Regression allows you to estimate how a [dependent variable](https://www.scribbr.com/methodology/independent-and-dependent-variables/) changes as the independent variable(s) change.

Multiple linear regression is used to estimate the relationship between two or more independent variables and one dependent variable. You can use multiple linear regression when you want to know:

1.How strong the relationship is between two or more independent variables and one dependent variable (e.g., how rainfall, temperature, and amount of fertilizer added affect crop growth).

2.The value of the dependent variable at a certain value of the independent variables (e.g., the expected yield of a crop at certain levels of rainfall, temperature, and fertilizer addition).

Example You are a public health researcher interested in social factors that influence heart disease. You survey 500 towns and gather data on the percentage of people in each town who smoke, the percentage of people in each town who bike to work, and the percentage of people in each town who have heart disease.

Because you have two independent variables and one dependent variable, and all your variables are quantitative, you can use multiple linear regression to analyze the relationship between them.

Multiple linear regression makes all of the same assumptions as[simple linear regression](https://www.scribbr.com/statistics/simple-linear-regression/):

**Homogeneity of variance (homoscedasticity):** the size of the error in our prediction doesn’t change significantly across the values of the independent variable.

**Independence of observations:** the observations in the dataset were collected using statistically valid methods, and there are no hidden relationships among variables.

In multiple linear regression, it is possible that some of the independent variables are correlated with one another, so it is important to check these before developing the regression model. If two independent variables are too highly correlated (r2 > ~0.6), then only one of them should be used in the regression model.

**Normality:** The data follows a [normal distribution](https://www.scribbr.com/statistics/normal-distribution/).

**Linearity:** the line of best fit through the data points is a straight line, rather than a curve or some sort of grouping factor.

## How to perform a multiple linear regression

### Multiple linear regression formula

The formula for a multiple linear regression is:

Multiple linear regression formula

* **y** = the predicted value of the dependent variable
* **B0**= the y-intercept (value of y when all other parameters are set to 0)
* **B1X1**= the regression coefficient (B1) of the first independent variable (**X1**) (a.k.a. the effect that increasing the value of the independent variable has on the predicted **y** value)
* **…** = do the same for however many independent variables you are testing
* **BnXn** = the regression coefficient of the last independent variable
* **e** = model error (a.k.a. how much variation there is in our estimate of **y**)

To find the best-fit line for each independent variable, multiple linear regression calculates three things:

* The regression coefficients that lead to the smallest overall model error.
* The t-statistic of the overall model.
* The associated [p-value](https://www.scribbr.com/statistics/p-value/) (how likely it is that the t-statistic would have occurred by chance if the null hypothesis of no relationship between the independent and dependent variables was true).

It then calculates the t-statistic and p-value for each regression coefficient in the model.

**Backward Elimination**

Multiple Linear Regression is a type of regression where the model depends on several independent variables (instead of only on one independent variable as seen in the case of Simple Linear Regression). Multiple Linear Regression has several techniques to build an effective model namely:

* All-in
* Backward Elimination
* Forward Selection
* Bidirectional Elimination

We will implement multiple linear regression using the backward elimination technique.  
Backward Elimination consists of the following steps:

* Select a significance level to stay in the model (e.g. SL = 0.05)
* Fit the model with all possible predictors
* Consider the predictor with the highest P-value. If P>SL, go to point d.
* Remove the predictor
* Fit the model without this variable and repeat the step c until the condition becomes false.

Let us suppose that we have a dataset containing a set of expenditure information for different companies. We would like to know the profit made by each company to determine which company can give the best results if collaborated with them. We build the regression model using a step-by-step approach.

# import the necessary libraries

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

# import the dataset

df = pd.read\_csv('50\_Startups.csv')

# first five entries of the dataset

df.head()

# split the dataframe into dependent and independent variables.

x = df[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]

y = df['Profit']

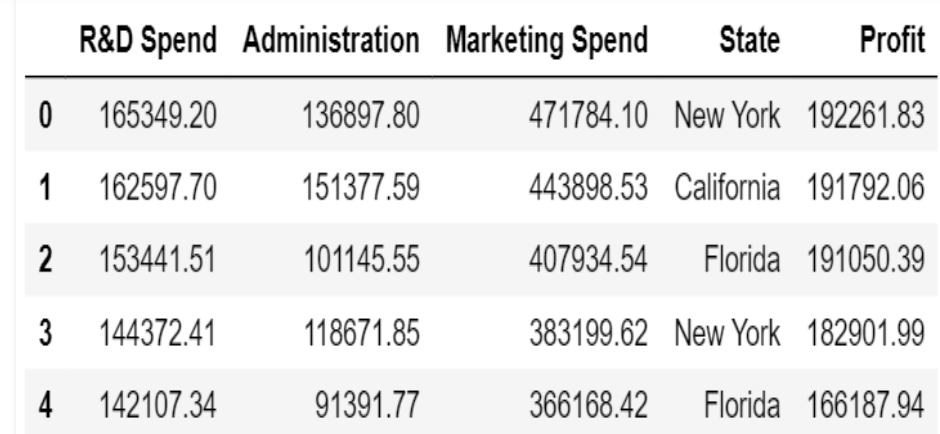
x.head()

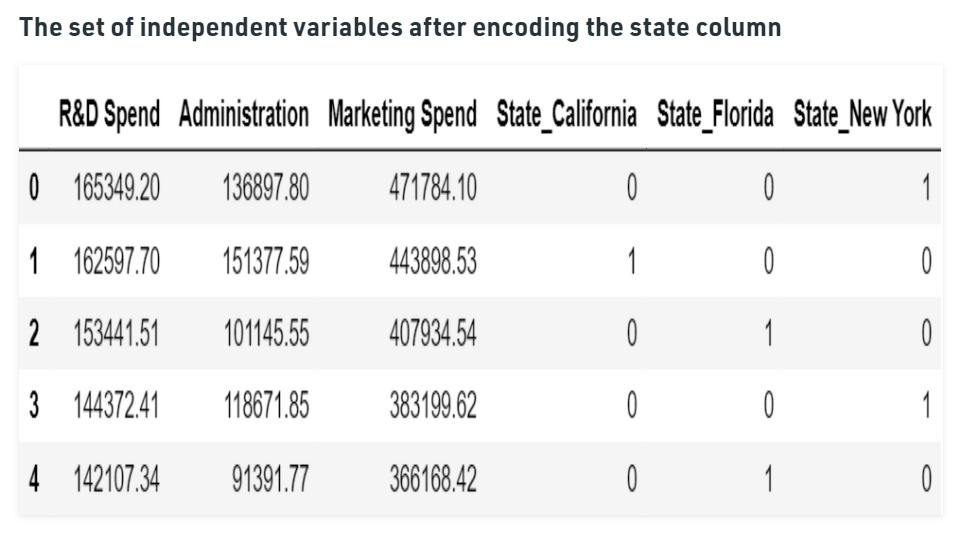
y.head()

# since the state is a string datatype column we need to encode it.

x = pd.get\_dummies(x)

x.head()





|  |
| --- |
| x\_train, x\_test, y\_train, y\_test = train\_test\_split(          x, y, test\_size = 0.3, random\_state = 0)  from sklearn.linear\_model import LinearRegression  lm = LinearRegression()  lm.fit(x\_train, y\_train)  pred = lm.predict(x\_test) |

Table

Description automatically generated with medium confidence

We can see that our predictions our close enough to the test set but how do we find the most important factor contributing to the profit.  
Here is a solution for that.  
We know that the equation of a multiple linear regression line is given by y=b1+b2\*x+b3\*x’+b4\*x”+…….  
where b1, b2, b3, … are the coefficients and x, x’, x” are all independent variables.  
Since we don’t have any ‘x’ for the first coefficient we assume it can be written as a product of b and 1 and hence we append a column of ones. There are libraries that take care of it but since we are using the stats model library we need to explicitly add the column.

import statsmodels.regression.linear\_model as sm

# add a column of ones as integer data type

x = np.append(arr = np.ones((50, 1)).astype(int),

              values = x, axis = 1)

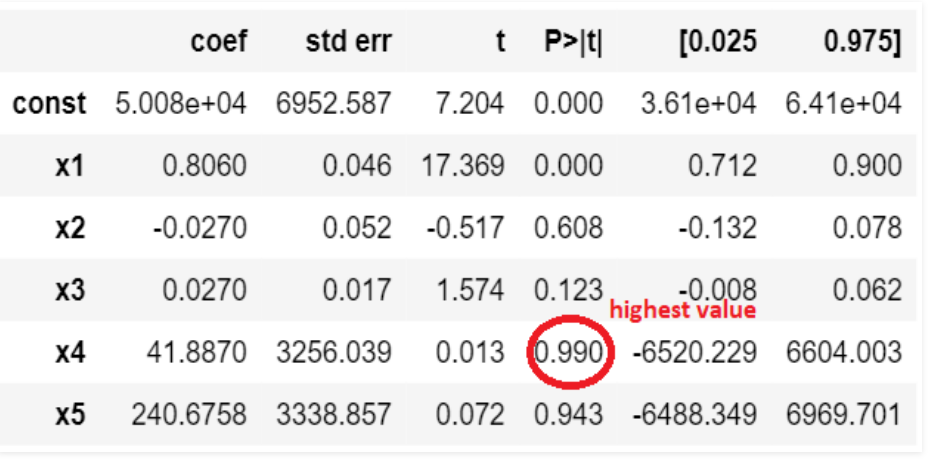
# choose a Significance level usually 0.05, if p>0.05

#  for the highest values parameter, remove that value

x\_opt = x[:, [0, 1, 2, 3, 4, 5]]

ols = sm.OLS(endog = y, exog = x\_opt).fit()

ols.summary()



**This figure shows the highest valued parameter**

And now we follow the steps of the **backward elimination and start eliminating unnecessary parameters.**

# remove the 4th column as it has the highest value

x\_opt = x[:, [0, 1, 2, 3, 5]]

ols = sm.OLS(endog = y, exog = x\_opt).fit()

ols.summary()

# remove the 5th column as it has the highest value

x\_opt = x[:, [0, 1, 2, 3]]

ols = sm.OLS(endog = y, exog = x\_opt).fit()

ols.summary()

# remove the 3rd column as it has the highest value

x\_opt = x[:, [0, 1, 2]]

ols = sm.OLS(endog = y, exog = x\_opt).fit()

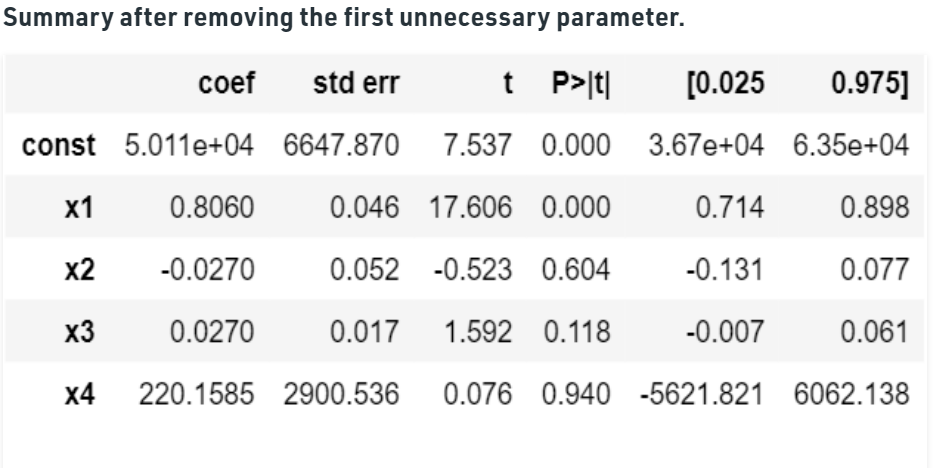
ols.summary()

# remove the 2nd column as it has the highest value

x\_opt = x[:, [0, 1]]

ols = sm.OLS(endog = y, exog = x\_opt).fit()

ols.summary()



So if we continue the process we see that we are left with only one column at the end and that is the R&D spent. We can conclude that the company which has maximum expenditure on the R&D makes the highest profit.

Text

Description automatically generated with medium confidence

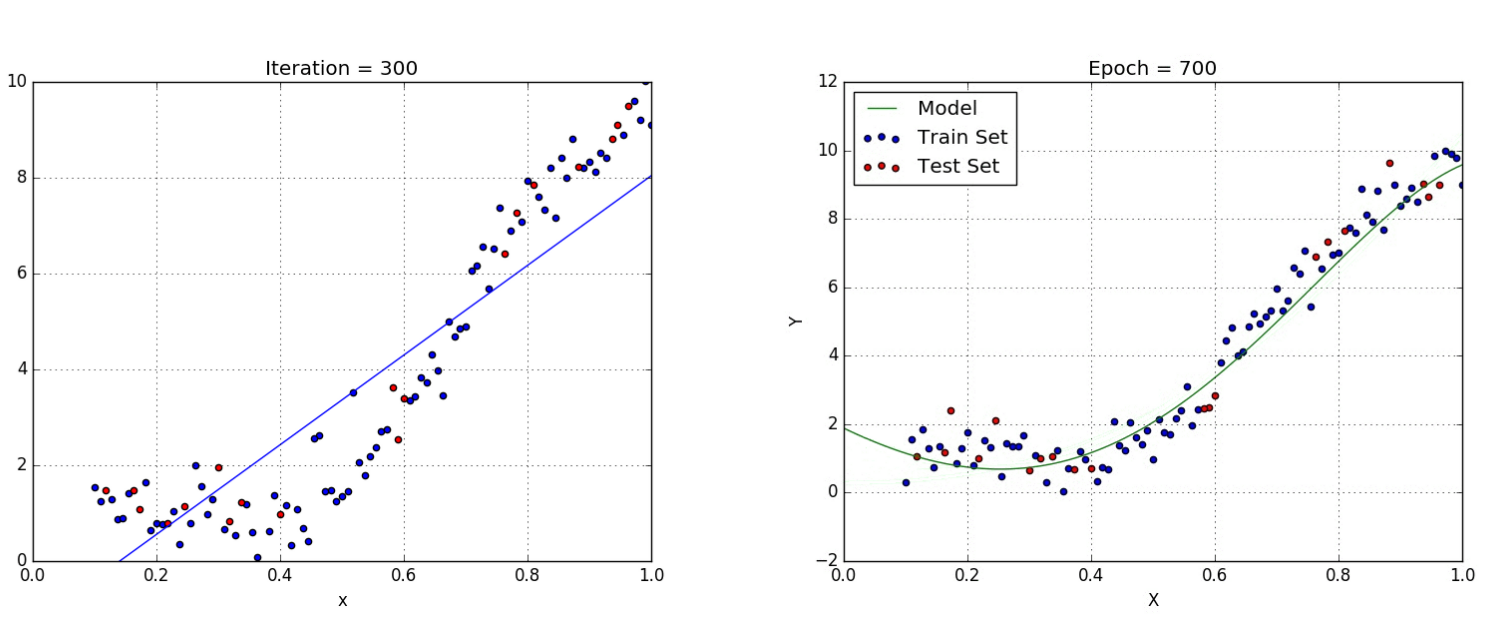
With this, we have solved the problem statement of finding the company for collaboration. Now let us have a brief look at the parameters of the OLS summary.

* **R square** – It talks about the goodness of the fit. It ranges between 0 and 1. The closer the value to 1, the better it is. It explains the extent of variation of the dependent variables in the model. However, it is biased in a way that it never decreases (even on adding variables).
* **Adj Rsquare** – This parameter has a penalizing factor(the no. of regressors) and it always decreases or stays identical to the previous value as the number of independent variables increases. If its value keeps increasing on removing the unnecessary parameters go ahead with the model or stop and revert.
* **F statistic** – It is used to compare two variances and is always greater than 0. It is formulated as v12/v22. In regression, it is the ratio of the explained to the unexplained variance of the model.
* **AIC and BIC** – AIC stands for Akaike’s information criterion and BIC stands for Bayesian information criterion Both these parameters depend on the likelihood function L.
* **Skew** – Informs about the data symmetry about the mean.
* **Kurtosis** – It measures the shape of the distribution i.e. The amount of data close to the mean than far away from the mean.
* **Omnibus** – **D’Angostino’s test**. It provides a combined statistical test for the presence of skewness and kurtosis.

**Log-likelihood** – It is the log of the likelihood function.

# **Polynomial Linear Regression**

In the last section, we saw two variables in your data set were correlated but what happens if we know that our data is correlated, but the relationship doesn’t look linear? So hence depending on what the data looks like, we can do a polynomial regression on the data to fit a polynomial equation to it.



Hence If we try to use a simple linear regression in the above graph then the linear regression line won’t fit very well. It is very difficult to fit a linear regression line in the above graph with a low value of error. Hence we can try to use the polynomial regression to fit a polynomial line so that we can achieve a minimum error or minimum cost function. The equation of the polynomial regression for the above graph data would be:

y = *θo + θ*₁*x*₁*+ θ*₂*x*₁²

This is the general equation of a polynomial regression is:

*Y*=θo + θ₁*X*+ θ₂*X*² + … + θₘXᵐ + **residual error**

**Advantages of using Polynomial Regression:**

Polynomial provides the best approximation of the relationship between the dependent and independent variable.

A Broad range of function can be fit under it.

Polynomial basically fits a wide range of curvature.

**Disadvantages of using Polynomial Regression**

The presence of one or two outliers in the data can seriously affect the results of the nonlinear analysis.

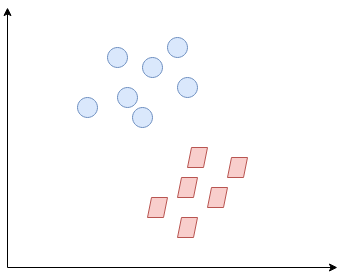
These are too sensitive to the outliers.

In addition, there are unfortunately fewer model validation tools for the detection of outliers in nonlinear regression than there are for linear regression.

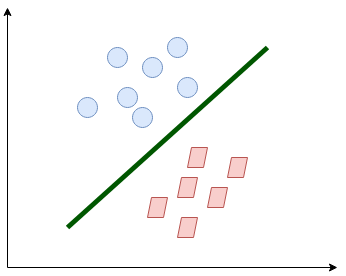
**Support Vector Regression**

## What is a Support Vector Machine (SVM)?

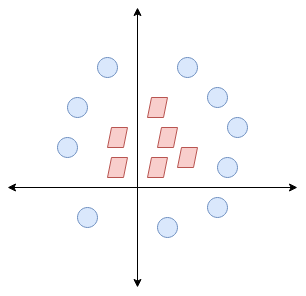
So what exactly is Support Vector Machine (SVM)? We’ll start by understanding SVM in simple terms. Let’s say we have a plot of two label classes as shown in the figure below:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR2.png)

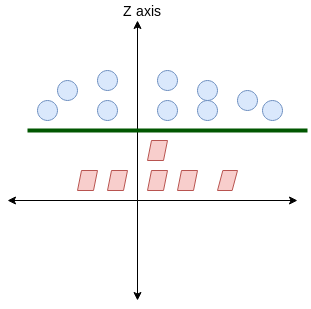
Can you decide what the separating line will be? You might have come up with this:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR3.png)

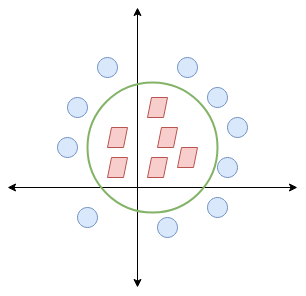
The line fairly separates the classes. This is what SVM essentially does – **simple class separation.**Now, what is the data was like this:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR4.png)

Here, we don’t have a simple line separating these two classes. So we’ll extend our dimension and introduce a new dimension along the z-axis. We can now separate these two classes:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR5.png)

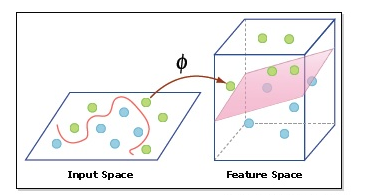
When we transform this line back to the original plane, it maps to the circular boundary as I’ve shown here:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR6.png)

**This is exactly what SVM does! It tries to find a line/hyperplane (in multidimensional space) that separates these two classes. Then it classifies the new point depending on whether it lies on the positive or negative side of the hyperplane depending on the classes to predict.**

Here are a few important parameters of SVM that you should be aware of before proceeding further:

* **Kernel:** A kernel helps us find a hyperplane in the higher dimensional space without increasing the computational cost. Usually, the computational cost will increase if the dimension of the data increases. This increase in dimension is required when we are unable to find a separating hyperplane in a given dimension and are required to move in a higher dimension:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/hyperplane.png)

* **Hyperplane:**This is basically a separating line between two data classes in SVM. But in Support Vector Regression, this is the line that will be used to predict the continuous output
* **Decision Boundary**: A decision boundary can be thought of as a demarcation line (for simplification) on one side of which lie positive examples and on the other side lie the negative examples. On this very line, the examples may be classified as either positive or negative. This same concept of SVM will be applied in Support Vector Regression as well

## Introduction to Support Vector Regression (SVR)

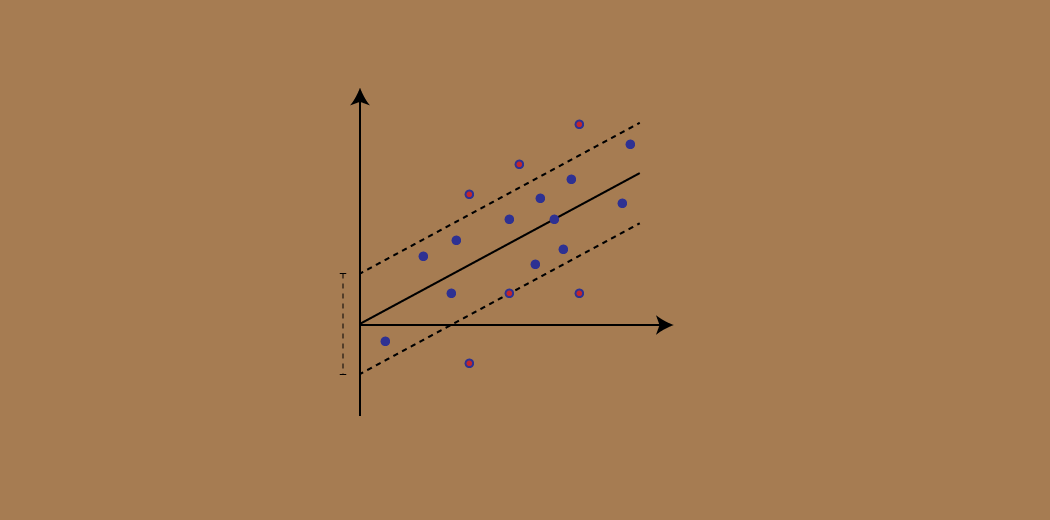
Support Vector Regression (SVR) uses the same principle as SVM, but for regression problems.

The problem of regression is to find a function that approximates mapping from an input domain to real numbers based on a training sample. So, let’s now dive deep and understand how SVR works.

## Unlocking a New World with the Support Vector Regression Algorithm

Support Vector Machines (SVM) are popularly and widely used for classification problems in machine learning. I’ve often relied on this not just in machine learning projects but when I want a quick result in a hackathon.

But SVM for regression analysis? I hadn’t even considered the possibility for a while! And even now when I bring up “Support Vector Regression” in front of machine learning beginners, I often get a bemused expression. I understand – most courses and experts don’t even mention Support Vector Regression (SVR) as a machine learning algorithm.

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/Support-Vector-Regression.gif)

But SVR has its uses as you’ll see in this tutorial. We will first quickly understand what SVM is, before diving into the world of Support Vector Regression and how to implement it in Python!

Note: You can learn about Support Vector Machines and Regression problems in course format here (it’s free!):

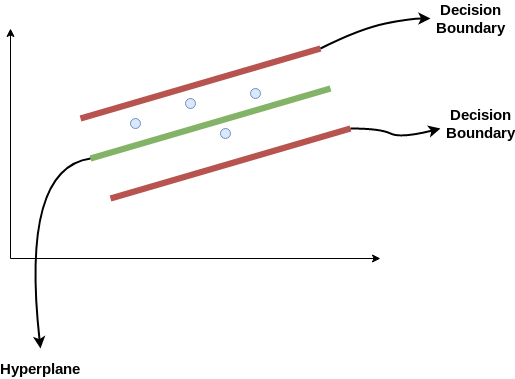
* [*Support Vector Machine (SVM) in Python and R*](https://courses.analyticsvidhya.com/courses/support-vector-machine-svm-in-python-and-r?utm_source=blog&utm_medium=understaing-support-vector-machine-example-code?utm_source=blog&utm_medium=support-vector-regression-tutorial-for-machine-learning)
* [*Fundamentals of Regression Analysis*](https://courses.analyticsvidhya.com/courses/Fundamentals-of-Regression-Analysis?utm_source=blog&utm_medium=support-vector-regression-tutorial-for-machine-learning)

## Introduction to Support Vector Regression (SVR)

Support Vector Regression (SVR) uses the same principle as SVM, but for regression problems. Let’s spend a few minutes understanding the idea behind SVR.

### **The Idea Behind Support Vector Regression**

The problem of regression is to find a function that approximates mapping from an input domain to real numbers based on a training sample. So, let’s now dive deep and understand how SVR works actually.

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/SVR1.png)

Consider these two red lines as the decision boundary and the green line as the hyperplane. **Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line.** Our best fit line is the hyperplane that has a maximum number of points.

The first thing that we’ll understand is what is the decision boundary (the danger red line above!). Consider these lines as being at any distance, say ‘a’, from the hyperplane. So, these are the lines that we draw at distance ‘+a’ and ‘-a’ from the hyperplane. This ‘a’ in the text is basically referred to as epsilon.

Assuming that the equation of the hyperplane is as follows:

Y = wx+b (equation of hyperplane)

Then the equations of decision boundary become:

wx+b= +a

wx+b= -a

Thus, any hyperplane that satisfies our SVR should satisfy:

**-a < Y- wx+b < +a**

Our main aim here is to decide a decision boundary at ‘a’ distance from the original hyperplane such that data points closest to the hyperplane or the support vectors are within that boundary line.

Hence, we are going to take only those points that are within the decision boundary and have the least error rate or are within the Margin of Tolerance. This gives us a better fitting model.

**Decision Tree Regression**

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

A picture containing diagram

Description automatically generated

**Decision Tree Regression**

The core algorithm for building decision trees called ID3 by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with Standard Deviation Reduction.

Standard Deviation

A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). We use standard deviation to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous its standard deviation is zero.

a) Standard deviation for one attribute:

Graphical user interface, text, application

Description automatically generated

* Standard Deviation (S) is for tree building (branching).
* Coefficient of Deviation (CV) is used to decide when to stop branching. We can use Count (n) as well.
* Average (Avg) is the value in the leaf nodes.

b) Standard deviation for two attributes (target and predictor):

Table

Description automatically generated

Standard Deviation Reduction

The standard deviation reduction is based on the decrease in standard deviation after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest standard deviation reduction (i.e., the most homogeneous branches).

Step 1: The standard deviation of the target is calculated.

Standard deviation (Hours Played) = 9.32

Step 2: The dataset is then split on the different attributes. The standard deviation for each branch is calculated. The resulting standard deviation is subtracted from the standard deviation before the split. The result is the standard deviation reduction.

Table

Description automatically generated

Step 3: The attribute with the largest standard deviation reduction is chosen for the decision node.

Table

Description automatically generated

Step 4a: The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches, until all data is processed.

Table

Description automatically generated

In practice, we need some termination criteria. For example, when coefficient of deviation (CV) for a branch becomes smaller than a certain threshold (e.g., 10%) and/or when too few instances (n) remain in the branch (e.g., 3).

Step 4b: "Overcast" subset does not need any further splitting because its CV (8%) is less than the threshold (10%). The related leaf node gets the average of the "Overcast" subset.

Table

Description automatically generated with low confidence

Step 4c: However, the "Sunny" branch has an CV (28%) more than the threshold (10%) which needs further splitting. We select "Windy" as the best node after "Outlook" because it has the largest SDR.

Table

Description automatically generated

Because the number of data points for both branches (FALSE and TRUE) is equal or less than 3 we stop further branching and assign the average of each branch to the related leaf node

A picture containing diagram

Description automatically generated

***Step 4d***: Moreover, the "rainy" branch has an CV (22%) which is more than the threshold (10%). This branch needs further splitting. We select "Windy" as the best best node because it has the largest SDR.

Table

Description automatically generated

Because the number of data points for all three branches (Cool, Hot and Mild) is equal or less than 3 we stop further branching and assign the average of each branch to the related leaf node.

When the number of instances is more than one at a *leaf node,* we calculate the *average* as the final value for the target.

Diagram

Description automatically generated

# **What is classification?**

Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y).

For example, spam detection in email service providers can be identified as a classification problem. This is s binary classification since there are only 2 classes as spam and not spam. A classifier utilizes some training data to understand how given input variables relate to the class. In this case, known spam and non-spam emails have to be used as the training data. When the classifier is trained accurately, it can be used to detect an unknown email.

Classification belongs to the category of supervised learning where the targets also provided with the input data. There are many applications in classification in many domains such as in credit approval, medical diagnosis, target marketing etc.

There are two types of learners in classification as lazy learners and eager learners.

**Lazy learners**

Lazy learners simply store the training data and wait until a testing data appear. When it does, classification is conducted based on the most related data in the stored training data. Compared to eager learners, lazy learners have less training time but more time in predicting.

Ex. k-nearest neighbor, Case-based reasoning

**2. Eager learners**

Eager learners construct a classification model based on the given training data before receiving data for classification. It must be able to commit to a single hypothesis that covers the entire instance space. Due to the model construction, eager learners take a long time for train and less time to predict.

Ex. Decision Tree, Naive Bayes, Artificial Neural Networks

**Naïve Bayes Classifier**

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes’ Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

To start with, let us consider a dataset.

Consider a fictional dataset that describes the weather conditions for playing a game of golf. Given the weather conditions, each tuple classifies the conditions as fit(“Yes”) or unfit(“No”) for playing golf.

Here is a tabular representation of our dataset.

| Outlook | Temperature | Humidity | Windy | Play Golf |
| --- | --- | --- | --- | --- |
| Rainy | Hot | High | False | No |
| Rainy | Hot | High | True | No |
| Overcast | Hot | High | False | Yes |
| Sunny | Mild | High | False | Yes |
| Sunny | Cool | Normal | False | Yes |
| Sunny | Cool | Normal | True | No |
| Overcast | Cool | Normal | True | Yes |
| Rainy | Mild | High | False | No |
| Rainy | Cool | Normal | False | Yes |
| Sunny | Mild | Normal | False | Yes |
| Rainy | Mild | Normal | True | Yes |
| Overcast | Mild | High | True | Yes |
| Overcast | Hot | Normal | False | Yes |
| Sunny | Mild | High | True | No |

The dataset is divided into two parts, namely, **feature matrix** and the **response vector**.

* Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value of **dependent features**. In above dataset, features are ‘Outlook’, ‘Temperature’, ‘Humidity’ and ‘Windy’.
* Response vector contains the value of **class variable**(prediction or output) for each row of feature matrix. In above dataset, the class variable name is ‘Play golf’.

**Assumption:**

The fundamental Naive Bayes assumption is that each feature makes an:

* independent
* equal

contribution to the outcome.

With relation to our dataset, this concept can be understood as:

* We assume that no pair of features are dependent. For example, the temperature being ‘Hot’ has nothing to do with the humidity or the outlook being ‘Rainy’ has no effect on the winds. Hence, the features are assumed to be **independent**.
* Secondly, each feature is given the same weight(or importance). For example, knowing only temperature and humidity alone can’t predict the outcome accurately. None of the attributes is irrelevant and assumed to be contributing **equally** to the outcome.

**Note:** The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Now, before moving to the formula for Naive Bayes, it is important to know about Bayes’ theorem.

**Bayes’ Theorem**

Bayes’ Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes’ theorem is stated mathematically as the following equation:

Graphical user interface, text

Description automatically generated with medium confidence

where A and B are events and P(B) ? 0.

* Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.
* P(A) is the **priori** of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
* P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

Now, with regards to our dataset, we can apply Bayes’ theorem in following way:



where, y is class variable and X is a dependent feature vector (of size n) where:

Text

Description automatically generated

Just to clear, an example of a feature vector and corresponding class variable can be: (refer 1st row of dataset)

X = (Rainy, Hot, High, False)

y = No

So basically, P(y|X) here means, the probability of “Not playing golf” given that the weather conditions are “Rainy outlook”, “Temperature is hot”, “high humidity” and “no wind”.

**Naive assumption**

Now, it’s time to put a naive assumption to the Bayes’ theorem, which is, **independence** among the features. So now, we split **evidence** into the independent parts.

Now, if any two events A and B are independent, then,

Graphical user interface, text, application, email

Description automatically generated

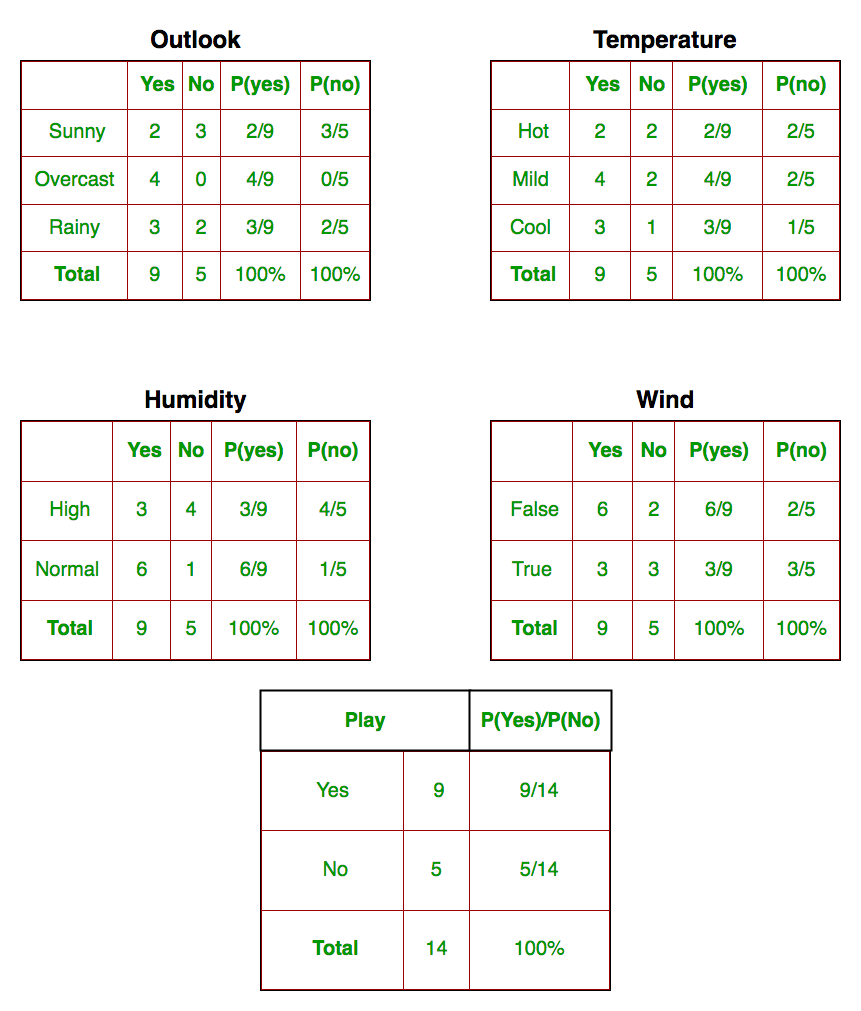
So, finally, we are left with the task of calculating P(y) and P(xi | y).

Please note that P(y) is also called **class probability** and P(xi | y) is called **conditional probability**.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi | y).

Let us try to apply the above formula manually on our weather dataset. For this, we need to do some precomputations on our dataset.

We need to find P(xi | yj) for each xi in X and yj in y. All these calculations have been demonstrated in the tables below:



So, in the figure above, we have calculated P(xi | yj) for each xi in X and yj in y manually in the tables 1-4. For example, probability of playing golf given that the temperature is cool, i.e P(temp. = cool | play golf = Yes) = 3/9.

Also, we need to find class probabilities (P(y)) which has been calculated in the table 5. For example, P(play golf = Yes) = 9/14.

So now, we are done with our pre-computations and the classifier is ready!

Let us test it on a new set of features (let us call it today):

today = (Sunny, Hot, Normal, False)

Table

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

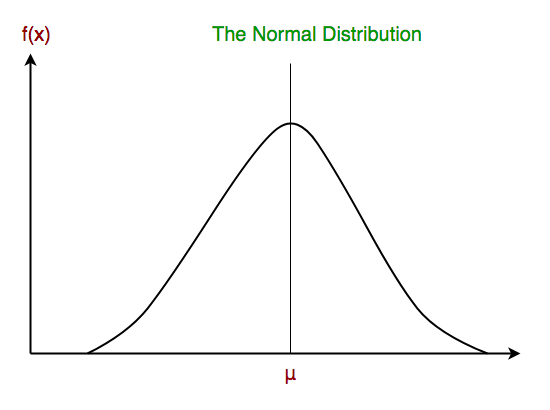
So, prediction that golf would be played is ‘Yes’.

The method that we discussed above is applicable for discrete data. In case of continuous data, we need to make some assumptions regarding the distribution of values of each feature. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi | y).

Now, we discuss one of such classifiers here.

**Gaussian Naive Bayes classifier**

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**. A Gaussian distribution is also called [Normal distribution](https://en.wikipedia.org/wiki/Normal_distribution). When plotted, it gives a bell-shaped curve which is symmetric about the mean of the feature values as shown below:



The likelihood of the features is assumed to be Gaussian; hence, conditional probability is given by:

Text, whiteboard

Description automatically generated

Other popular Naive Bayes classifiers are:

* **Multinomial Naive Bayes**: Feature vectors represent the frequencies with which certain events have been generated by a **multinomial distribution**. This is the event model typically used for document classification.
* **Bernoulli Naive Bayes**: In the multivariate Bernoulli event model, features are independent booleans (binary variables) describing inputs. Like the multinomial model, this model is popular for document classification tasks, where binary term occurrence(i.e. a word occurs in a document or not) features are used rather than term frequencies(i.e. frequency of a word in the document).

**Logistic Regression**

Logistic Regression is used when the dependent variable(target) is categorical.

For example,

* To predict whether an email is spam (1) or (0)
* Whether the tumor is malignant (1) or not (0)

Consider a scenario where we need to classify whether an email is spam or not. If we use linear regression for this problem, there is a need for setting up a threshold based on which classification can be done. Say if the actual class is malignant, predicted continuous value 0.4 and the threshold value is 0.5, the data point will be classified as not malignant which can lead to serious consequence in real time. From this example, it can be inferred that linear regression is not suitable for classification problem. Linear regression is unbounded, and this brings logistic regression into picture. Their value strictly ranges from 0 to 1.

**Simple Logistic Regression**

**Model**

Output = 0 or 1

Hypothesis => Z = WX + B

hΘ(x) = sigmoid (Z)

**Sigmoid Function**

Chart, diagram

Description automatically generated

Figure 2: Sigmoid Activation Function

If ‘Z’ goes to infinity, Y(predicted) will become 1 and if ‘Z’ goes to negative infinity, Y(predicted) will become 0.

**Analysis of the hypothesis**

The output from the hypothesis is the estimated probability. This is used to infer how confident can predicted value be actual value when given an input X. Consider the below example,

X = [x0 x1] = [1 IP-Address]

Based on the x1 value, let’s say we obtained the estimated probability to be 0.8. This tells that there is 80% chance that an email will be spam.

Mathematically this can be written as,

Text

Description automatically generated

Figure 3: Mathematical Representation

This justifies the name ‘logistic regression’. Data is fit into linear regression model, which then be acted upon by a logistic function predicting the target categorical dependent variable.

**Types of Logistic Regression**

1. Binary Logistic Regression

The categorical response has only two 2 possible outcomes. Example: Spam or Not

2. Multinomial Logistic Regression

Three or more categories without ordering. Example: Predicting which food is preferred more (Veg, Non-Veg, Vegan)

3. Ordinal Logistic Regression

Three or more categories with ordering. Example: Movie rating from 1 to 5

**Decision Boundary**

To predict which class a data belongs, a threshold can be set. Based upon this threshold, the obtained estimated probability is classified into classes.

Say, if predicted\_value ≥ 0.5, then classify email as spam else as not spam.

Decision boundary can be linear or non-linear. Polynomial order can be increased to get complex decision boundary.

**Cost Function**

Text

Description automatically generated with medium confidence

Figure 4: Cost Function of Logistic Regression

Why cost function which has been used for linear cannot be used for logistic?

Linear regression uses mean squared error as its cost function. If this is used for logistic regression, then it will be a non-convex function of parameters (theta). Gradient descent will converge into global minimum only if the function is convex.

Chart, line chart

Description automatically generated

**Cost function explanation**

Diagram

Description automatically generated

Figure 6: Cost Function part 1

Diagram

Description automatically generated

Figure 7: Cost Function part 2

**Simplified cost function**

Graphical user interface, text

Description automatically generated

Figure 8: Simplified Cost Function

**Why this cost function?**

Text, letter

Description automatically generated

Figure 9: Maximum Likelihood Explanation part-1

Text, letter

Description automatically generated

Figure 10: Maximum Likelihood Explanation part-2

This negative function is because when we train, we need to maximize the probability by minimizing loss function. Decreasing the cost will increase the maximum likelihood assuming that samples are drawn from an identically independent distribution.

**Deriving the formula for Gradient Descent Algorithm**

A picture containing text, receipt

Description automatically generated

Figure 11: Gradient Descent Algorithm part 1

Diagram, letter

Description automatically generated

**Decision Tree**

* Decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems.
* Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
* We can represent any Boolean function on discrete attributes using the decision tree.

Diagram

Description automatically generated

**Below are some assumptions that we made while using decision tree:**

* At the beginning, we consider the whole training set as the root.
* Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
* On the basis of attribute values records are distributed recursively.
* We use statistical methods for ordering attributes as root or the internal node.

Graphical user interface, diagram

Description automatically generated

As you can see from the above image that Decision Tree works on the Sum of Product form which is also known as *Disjunctive Normal Form*. In the above image, we are predicting the use of computer in the daily life of the people.

In Decision Tree the major challenge is to identification of the attribute for the root node in each level. This process is known as attribute selection.

**Information Gain:**  
When we use a node in a decision tree to partition the training instances into smaller subsets the entropy changes. Information gain is a measure of this change in entropy.  
***Definition***: Suppose S is a set of instances, A is an attribute, Sv is the subset of S with A = v, and Values (A) is the set of all possible values of A, then

**Entropy:**  
Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy more the information content.  
***Definition***: Suppose S is a set of instances, A is an attribute, Sv is the subset of S with A = v, and Values (A) is the set of all possible values of A, then  
  
Example:

For the set X = {a,a,a,b,b,b,b,b}

Total intances: 8

Instances of b: 5

Instances of a: 3

= -[0.375 \* (-1.415) + 0.625 \* (-0.678)]

=-(-0.53-0.424)

= 0.954

**Building Decision Tree using Information Gain**  
**The essentials:**

* Start with all training instances associated with the root node
* Use info gain to choose which attribute to label each node with
* *Note:* No root-to-leaf path should contain the same discrete attribute twice
* Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree.

**The border cases:**

* If all positive or all negative training instances remain, label that node “yes” or “no” accordingly
* If no attributes remain, label with a majority vote of training instances left at that node
* If no instances remain, label with a majority vote of the parent’s training instances

**Example:**  
Now, lets draw a Decision Tree for the following data using Information gain.

**Training set: 3 features and 2 classes**

| X | Y | Z | C |
| --- | --- | --- | --- |
| 1 | 1 | 1 | I |
| 1 | 1 | 0 | I |
| 0 | 0 | 1 | II |
| 1 | 0 | 0 | II |

Here, we have 3 features and 2 output classes.  
To build a decision tree using Information gain. We will take each of the feature and calculate the information for each feature.  
Chart, bubble chart

Description automatically generated

**Split on feature X**

Diagram

Description automatically generated with low confidence

**Split on feature Y**

Bubble chart

Description automatically generated with medium confidence

**Split on feature Z**

From the above images we can see that the information gain is maximum when we make a split on feature Y. So, for the root node best suited feature is feature Y. Now we can see that while splitting the dataset by feature Y, the child contains pure subset of the target variable. So we don’t need to further split the dataset.

The final tree for the above dataset would be look like this:  
A picture containing text, clock, vector graphics

Description automatically generated

Windowing in ID3

Windowing is applied in ID3 as a way of dealing with large sets of training instances. Without

windowing, such an algorithm can be really slow, as it needs to do entropy calculations over huge

amounts of data. With windowing, training is done on a relatively small sample of the data, and

then checked against the full set of training data.

Here is the windowing algorithm:

1. Select a sample S of the training instances at random. The proportion chosen would need to

be small enough that ID3 could run fast, but large enough to be representative of the

whole set of examples.

2. Run the ID3 algorithm on the set of training instances to obtain a decision tree.

3. Check the decision tree on the full data set, to obtain a set E of training instances that are

misclassified by the current tree.

4. If E is empty, stop.

5. Let the new set of training instances S1 be the union of S‘ and E.

6. Go to step 2 and run the ID3 to induce a tree from S’

Pruning in ID3

Pruning is the final step after growing tree. Purpose: avoid overfitting. The idea is to produce a

smaller (simpler) tree with improved generalization properties.

• Split data into training and validation sets.

• Try removing each possible node and evaluate impact on validation set.

• Remove node that improves accuracy the most on validation set.

• Repeat until no additional improvement possible.

Handling continuous attributes in ID3

Discretize.

CART

CART stands for “Classification and Regression Trees”. It is similar to ID3 with a few differences.

The most important difference is in the way pruning is computed. The idea of CART is to build

many “small” trees and combine them in order to form “larger” trees.

**Advantages to using decision trees**:

1. Easy to interpret and make for straightforward visualizations.

2. The internal workings are capable of being observed and thus make it possible to reproduce work.

3. Can handle both numerical and categorical data.

4. Perform well on large datasets

5. Are extremely fast

**Disadvantages of decision trees**:

1. Building decision trees require algorithms capable of determining an optimal choice at each node. One popular algorithm is the Hunt’s algorithm. This is a greedy model, meaning it makes the most optimal decision at each step, but does not consider the global optimum. What does this mean? At each step the algorithm chooses the best result. However, choosing the best result at a given step does not ensure you will be headed down the route that will lead to the optimal decision when you make it to the final node of the tree, called the leaf node.

2. Decision trees are prone to overfitting, especially when a tree is particularly deep. This is due to the amount of specificity we look at leading to smaller sample of events that meet the previous assumptions. This small sample could lead to unsound conclusions. An example of this could be predicting if the Boston Celtics will beat the Miami Heat in tonight’s basketball game. The first level of the tree could ask if the Celtics are playing home or away. The second level might ask if the Celtics have a higher win percentage than their opponent, in this case the Heat. The third level asks if the Celtic’s leading scorer is playing? The fourth level asks if the Celtic’s second leading scorer is playing. The fifth level asks if the Celtics are traveling back to the east coast from 3 or more consecutive road games on the west coast. While all of these questions may be relevant, there may only be two previous games where the conditions of tonights game were met. Using only two games as the basis for our classification would not be adequate for an informed decision. One way to combat this issue is by setting a max depth. This will limit our risk of overfitting; but as always, this will be at the expense of error due to bias. Thus if we set a max depth of three, we would only ask if the game is home or away, do the Celtics have a higher winning percentage than their opponent, and is their leading scorer playing. This is a simpler model with less variance sample to sample but ultimately will not be a strong predictive model.

Ideally, we would like to minimize both error due to bias and error due to variance. Enter random forests. Random forests mitigate this problem well. A random forest is simply a collection of decision trees whose results are aggregated into one final result. Their ability to limit overfitting without substantially increasing error due to bias is why they are such powerful models.

One-way Random Forests reduce variance is by training on different samples of the data. A second way is by using a random subset of features. This means if we have 30 features, random forests will only use a certain number of those features in each model, say five. Unfortunately, we have omitted 25 features that could be useful. But as stated, a random forest is a collection of decision trees. Thus, in each tree we can utilize five random features. If we use many trees in our forest, eventually many or all of our features will have been included. This inclusion of many features will help limit our error due to bias and error due to variance. If features weren’t chosen randomly, base trees in our forest could become highly correlated. This is because a few features could be particularly predictive and thus, the same features would be chosen in many of the base trees. If many of these trees included the same features, we would not be combating error due to variance.

With that said, random forests are a strong modeling technique and much more robust than a single decision tree. They aggregate many decision trees to limit overfitting as well as error due to bias and therefore yield useful results.

**Ensemble Methods (Random Forest)**

Ensemble Methods, what are they? **Ensemble methods** is a machine learning technique that combines several base models in order to produce one optimal predictive model. To better understand this definition let’s take a step back into the goal of machine learning and model building. This is going to make more sense as I dive into specific examples and why Ensemble methods are used.

I will largely utilize Decision Trees to outline the definition and practicality of Ensemble Methods (however it is important to note that Ensemble Methods do not only pertain to Decision Trees).

Diagram

Description automatically generated

A Decision Tree determines the predictive value based on series of questions and conditions. For instance, this simple Decision Tree determining on whether an individual should play outside or not. The tree takes several weather factors into account and given each factor either makes a decision or asks another question. In this example, every time it is overcast, we will play outside. However, if it is raining, we must ask if it is windy or not? If windy, we will not play. But given no wind, tie those shoelaces tight because were going outside to play.

Diagram

Description automatically generated

Decision Trees can also solve quantitative problems as well with the same format. In the Tree to the left, we want to know whether or not to invest in a commercial real estate property. Is it an office building? A Warehouse? An Apartment building? Good economic conditions? Poor Economic Conditions? How much will an investment return? These questions are answered and solved using this decision tree.

When making Decision Trees, there are several factors we must take into consideration: On what features do we make our decisions on? What is the threshold for classifying each question into a yes or no answer? In the first Decision Tree, what if we wanted to ask ourselves if we had friends to play with or not. If we have friends, we will play every time. If not, we might continue to ask ourselves questions about the weather. By adding an additional question, we hope to greater define the Yes and No classes.

This is where Ensemble Methods come in handy! Rather than just relying on one Decision Tree and hoping we made the right decision at each split, Ensemble Methods allow us to take a sample of Decision Trees into account, calculate which features to use or questions to ask at each split, and make a final predictor based on the aggregated results of the sampled Decision Trees.

# **Types of Ensemble Methods**

1. **Bagg**ing, or **B**ootstrap **Agg**regating. **Bagg**ing gets its name because it combines **B**ootstrapping and **Agg**regation to form one ensemble model. Given a sample of data, multiple bootstrapped subsamples are pulled. A Decision Tree is formed on each of the bootstrapped subsamples. After each subsample Decision Tree has been formed, an algorithm is used to aggregate over the Decision Trees to form the most efficient predictor. The image below will help explain:

Diagram

Description automatically generated

Given a Dataset, bootstrapped subsamples are pulled. A Decision Tree is formed on each bootstrapped sample. The results of each tree are aggregated to yield the strongest, most accurate predictor.

2. **Random Forest** Models. Random Forest Models can be thought of as **Bagg**ing, with a slight tweak. When deciding where to split and how to make decisions, Bagged Decision Trees have the full disposal of features to choose from. Therefore, although the bootstrapped samples may be slightly different, the data is largely going to break off at the same features throughout each model. In contrary, Random Forest models decide where to split based on a random selection of features. Rather than splitting at similar features at each node throughout, Random Forest models implement a level of differentiation because each tree will split based on different features. This level of differentiation provides a greater ensemble to aggregate over, ergo producing a more accurate predictor. Refer to the image for a better understanding.

Diagram

Description automatically generated

Similar to Bagging, bootstrapped subsamples are pulled from a larger dataset. A decision tree is formed on each subsample. HOWEVER, the decision tree is split on different features (in this diagram the features are represented by shapes).

### Bootstrap aggregating (bagging)

Bootstrap aggregating, often abbreviated as *bagging*, involves having each model in the ensemble vote with equal weight. In order to promote model variance, bagging trains each model in the ensemble using a randomly drawn subset of the training set. As an example, the [random forest](https://en.wikipedia.org/wiki/Random_forest) algorithm combines random decision trees with bagging to achieve very high classification accuracy.[[16]](https://en.wikipedia.org/wiki/Ensemble_learning#cite_note-16)

In bagging the samples are generated in such a way that the samples are different from each other however replacement is allowed. Replacement means that an instance can occur in multiple samples multiple times or it can not appear in some samples at all. These samples are then given to multiple learners and then the results from each learner are combined in the form of voting.

### Boosting

Boosting involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models mis-classified. In some cases, boosting has been shown to yield better accuracy than bagging, but it also tends to be more likely to over-fit the training data. By far, the most common implementation of boosting is [Adaboost](https://en.wikipedia.org/wiki/Adaboost" \o "Adaboost), although some newer algorithms are reported to achieve better results.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

In Boosting, an equal weight (uniform probability distribution) is given to the sample training data (say D1) at the very starting round. This data (D1) is then given to a base learner (say L1). The mis-classified instances by L1 are assigned a weight higher than the correctly classified instances, but keeping in mind that the total probability distribution will be equal to 1. This boosted data (say D2) is then given to second base learner (say L2) and so on. The results are then combined in the form of voting.

**Support Vector Machine**

Support Vector Machine, abbreviated as SVM can be used for both regression and classification tasks. But, it is widely used in classification objectives.

What is Support Vector Machine?

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

Line chart

Description automatically generated with low confidenceChart

Description automatically generatedPossible hyperplanes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes and Support Vectors

Chart, scatter chart

Description automatically generated

Hyperplanes in 2D and 3D feature space

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.

Chart, scatter chart

Description automatically generated

Support Vectors

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

Large Margin Intuition

In logistic regression, we take the output of the linear function and squash the value within the range of [0,1] using the sigmoid function. If the squashed value is greater than a threshold value(0.5) we assign it a label 1, else we assign it a label 0. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values([-1,1]) which acts as margin.

Cost Function and Gradient Updates

In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.

Text, letter

Description automatically generated

Text

Description automatically generated

Hinge loss function (function on left can be represented as a function on the right)

The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions looks as below.

Diagram, schematic

Description automatically generated

Loss function for SVM

Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.

Text

Description automatically generated

Gradients

When there is no misclassification, i.e our model correctly predicts the class of our data point, we only have to update the gradient from the regularization parameter.

A picture containing text

Description automatically generated

Gradient Update — No misclassification

When there is a misclassification, i.e our model make a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.

Text

Description automatically generated with low confidence

# **Linear Inseparability**

Before we move on to the concepts of Soft Margin and Kernel trick, let us establish the need of them. Suppose we have some data, and it can be depicted as following in the 2D space:

Chart, scatter chart

Description automatically generated

Figure 1: Data representation where the two classes are not linearly separable

From the figure, it is evident that there’s no specific linear decision boundary that can perfectly separate the data, i.e. the data is linearly inseparable. We can have a similar situation in higher-dimensional representations as well. This can be attributed to the fact that usually, the features we derive from the data don’t contain sufficient information so that we can clearly separate the two classes. This is usually the case in many real-world applications. Fortunately, researchers have already come up with techniques that can handle situations like these. Let’s see what they are and how they work.

# **Soft Margin Formulation**

This idea is based on a simple premise: allow SVM to make a certain number of mistakes and keep margin as wide as possible so that other points can still be classified correctly. This can be done simply by modifying the objective of SVM.

## Motivation

Let us briefly go over the motivation for having this kind of formulation.

* As mentioned earlier, almost all real-world applications have data that is linearly inseparable.
* In rare cases where the data is linearly separable, we might not want to choose a decision boundary that perfectly separates the data to avoid overfitting. For example, consider the following diagram:

Chart, scatter chart

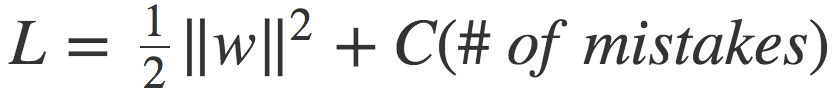
Description automatically generated

Figure 2: Which decision boundary is better? Red or Green?

Here the red decision boundary perfectly separates all the training points. However, is it really a good idea of having a decision boundary with such less margin? Do you think such kind of decision boundary will generalize well on unseen data? The answer is: No. The green decision boundary has a wider margin that would allow it to generalize well on unseen data. In that sense, soft margin formulation would also help in avoiding the overfitting problem.

## How it Works (mathematically)?

Let us see how we can modify our objective to achieve the desired behavior. In this new setting, we would aim to minimize the following objective:



equation 1

This differs from the original objective in the second term. Here, **C** is a hyperparameter that decides the trade-off between maximizing the margin and minimizing the mistakes. When **C** is small, classification mistakes are given less importance and focus is more on maximizing the margin, whereas when **C** is large, the focus is more on avoiding misclassification at the expense of keeping the margin small.

At this point, we should note, however, that not all mistakes are equal. Data points that are far away on the wrong side of the decision boundary should incur more penalty as compared to the ones that are closer. Let’s see how this could be incorporated with the help of the following diagram.

Chart, scatter chart

Description automatically generated

Figure 3: The penalty incurred by data points for being on the wrong side of the decision boundary

The idea is: for every data point **x\_i**, we introduce a slack variable **ξ\_i**. The value of **ξ\_i**is the distance of **x\_i** from the corresponding class’s margin if **x\_i** is on the wrong side of the margin, otherwise zero. Thus the points that are far away from the margin on the wrong side would get more penalty.

With this idea, each data point **x\_i**needs to satisfy the following constraint:

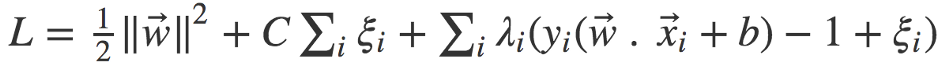
A picture containing text

Description automatically generated

equation 2

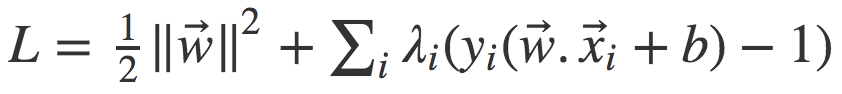
Here, the left-hand side of the inequality could be thought of like the confidence of classification. Confidence score ≥ 1 suggests that classifier has classified the point correctly. However, if confidence score ≤ 1, it means that classifier did not classify the point correctly and incurring a linear penalty of **ξ\_i.**

Given these constraints, our objective is to minimize the following function:



equation 3

where we have used the concepts of [Lagrange Multiplier](https://en.wikipedia.org/wiki/Lagrange_multiplier) for optimizing loss function under constraints. Let us compare this with SVM’s objective that handles the linearly separable cases (as given below).



We see that only **ξ\_i**terms are extra in the modified objective and everything else is the same.

*Point to note: In the final solution,****λ\_i****s corresponding to points that are closest to the margin and on the wrong side of the margin (i.e. having non-zero****ξ\_i****) would be non-zero as they play a key role in positioning of the decision boundary, essentially making them the support vectors.*

# **Kernel Trick**

Now let us explore the second solution of using “Kernel Trick” to tackle the problem of linear inseparability. But first, we should learn what Kernel functions are.

## Kernel Functions

Kernel functions are generalized functions that take two vectors (of any dimension) as input and output a score that denotes how similar the input vectors are. A simple Kernel function you already know is the dot product function: if the dot product is small, we conclude that vectors are different and if the dot product is large, we conclude that vectors are more similar. If you are interested in knowing about other types of Kernel functions, [this](https://en.wikipedia.org/wiki/Kernel_method#Popular_kernels) would be a good source.

## The “Trick”

Let us look at the objective function for the linearly separable case:



equation 5

*This is a modified form of the objective in equation 4. Here, we have substituted the optimal value of****w****and****b****. These optimal values can be calculated by differentiating equation 4 with respect to these parameters and equating it to 0.*

We can observe from equation 5 that objective depends on the dot product of input vector pairs (**x\_i . x\_j**), which is nothing but a Kernel function. Now here’s a good thing: we don’t have to be restricted to a simple Kernel function like dot product. We can use any [fancy Kernel function](https://en.wikipedia.org/wiki/Kernel_method#Popular_kernels) in place of dot product that has the capability of measuring similarity in higher dimensions (where it could be more accurate; more on this later), without increasing the computational costs much. This is essentially known as the Kernel Trick.

## How it Works (mathematically)?

A Kernel function can be written mathematically as follows:



equation 6

Here **x** and **y** are input vectors, **ϕ** is a transformation function and < , > denotes dot product operation. In the case of dot product function, **ϕ**just maps the input vector to itself.

Kernel functions essentially take the dot product of transformed input vectors.

Now let us consider the case depicted in figure 4 below. We see that there is no linear decision boundary in 2d space that could perfectly separate the data points. A circular (or quadratic) decision boundary might do the job, however, linear classifiers are not capable of coming up with these types of decision boundaries.

Diagram

Description automatically generated

Figure 4: Points in 2D space are separable by a circular decision boundary.

In figure 4, each point P is represented by the features of form (x,y) in 2D space. Looking at the desirable decision boundary, we can define a transformation function **ϕ** for a point P as **ϕ**(P) = (x^2, y^2, √2xy) (why we came up with such a transformation would be clear in just a moment). Let’s see what the Kernel function looks like for this type of transformation for two points P\_1 and P\_2.

Diagram, text, schematic

Description automatically generated

equation 7

If we observe the final form of the Kernel function, it’s nothing but [a circle](https://www.mathsisfun.com/algebra/circle-equations.html)! This means we have changed our notion of similarity: instead of measuring similarity by how close the points are (using the dot product), we are measuring similarity based on whether points are within a circle. In this sense, defining such a transformation allowed us to have a non-linear decision boundary in 2D space (it is still linear in the original 3D space). It could be a lot to keep track of, so following is a brief summary of the decisions we have taken:

**1** - Each point P is represented by (**x**,**y**) coordinates in 2D space.**2** - We project the points to 3D space by transforming their coordinates to (**x^2**, **y^2**, **√2xy**)**3** - Points which have high value of **x**.**y** would move upwards along the z-axis (in this case, mostly the red circles). [This video](https://www.youtube.com/watch?v=3liCbRZPrZA) provides a good visualization of the same.**4** - We find a hyperplane in 3D space that would perfectly separate the classes.**5** - The form of Kernel function indicates that this hyperplane would form a circle in 2D space, thus giving us a non-linear decision boundary.

And the main takeaway is:

By embedding the data in a higher-dimensional feature space, we can keep using a linear classifier!

A caveat here is that these transformations could increase the feature space drastically and that increases the computational costs. Is there any way we can reap the aforementioned benefits while not increasing the computational costs much? Turns out there is!

Let us try to rewrite the Kernel function in equation 7:

Text, letter

Description automatically generated

equation 8

Whoa! So the value of Kernel function (thus, the similarity between points in 3D space) is just the square of the dot product between points in 2D space. Pretty great, right?! But how did this happen?

The reason for this is that we chose our transformation function **ϕ**wisely. And as long as we continue to do that, we can circumvent the transformation step and calculate the value of Kernel function directly from the similarity between points in 2D space. This, in turn, would also curb the computational costs. We have many popular [Kernel functions](https://en.wikipedia.org/wiki/Kernel_method#Popular_kernels) which have this nice property and can be used out of the box (we don’t need to search for perfect **ϕ**).

**Performance Measurement**

**Performance Measurement** is an essential task for any machine learning project, it is very important to check how good or bad our model is. We use R squared (R²) and Root mean squared error (RMSE) when it comes to regression models. In case of classification models, we can rely on an AUC-ROC curve or CAP curve, when we need to evaluate or illustrate the performance of a multi-class classification issue.

**What is AUC-ROC curve?**AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve, also known as AUROC (Area Under the Receiver Operating Characteristics) is a performance measurement for classification problems at various threshold levels. It is one of the most significant evaluation measures for assessing the performance of binary classification problems.  
ROC is a probability curve that plots the TPR (True Positive Rate) against FPR (False Positive Rate). AUC is the measure of separability, it shows how much our model is capable to distinguish between classes.  
The AUC indicates how well the model distinguishes between positive and negative classes. The greater the AUC, the better.

Chart, pie chart

Description automatically generated

AUC-ROC Curve

**What is CAP curve?**A CAP (Cumulative Accuracy Profile) curve is a performance measurement for classification problems. It is used to evaluate a model by comparing the current curve to both the ‘perfect’ or ‘ideal’ curve and a ‘randomized’ curve.  
A decent or good model will have a CAP that is in the middle of the perfect and random curves. The closer a model is to the perfect CAP, the better is.

Diagram

Description automatically generated

CAP Curve

There are two methods to analyze the performance using CAP curve:

* Area Under Curve: We calculate the Accuracy Rate(AR) by calculating area under the perfect model and the random model (aP), and calculating area under the prediction model and random model (aR).  
  Accuracy Rate (AR) = aR / aP  
  Higher the AR, that is closer to the 1, better is the model.

Diagram

Description automatically generated

Area under the perfect model and random model (aP)

Diagram

Description automatically generated

Area under the good model and random model (aR)

* Plot: We draw a vertical line at 50% from x-axis till it intersects the ‘good model’ line, from that intersection point we draw a horizontal line till y-axis. now this point which cuts y-axis is the percentage of how many positive outcomes you are going to identify if you take 50% of the population.

Diagram

Description automatically generated

Just by looking at this plot, you can assess the performance of the model based on X% value.

1. X < 60% → Rubbish
2. 60% < X < 70% → Poor/Average
3. 70% < X <80% → Good
4. 80% < X < 90% → Very Good
5. 90% < X < 100% → Too Good ( In this case you should be very careful with the chances of overfitting )

Now I will show you how you can use AUCROC curve and CAP curve to evaluate your classification model using python.

**Dataset**I am using the [Social\_Network\_Ads](https://www.kaggle.com/rakeshrau/social-network-ads?select=Social_Network_Ads.csv" \t "_blank) dataset, in which I have used only three features, ‘Age’, ‘EstimatedSalary’ and ‘Purchased’. The output labels are ‘0’ and ‘1’ which represent whether a person a purchased the product or not.  
0 → not purchased  
1 → purchased  
Our goal is to predict if any person will buy the product or not.

Chart, scatter chart

Description automatically generated

Complete Dataset

The ‘Red’ points represent people who have not purchased the product and ‘Green’ points represents people who have purchased the product.

**Classification**I have split the dataset into two set, 75% training data and 25% testing data. I used the **Logistic Regression**to train and test the model. The model achieved a accuracy score of **89%**

Chart, scatter chart

Description automatically generated

Classification on test set

**Performance Evaluation:**

* AUROC :  
  Import roc\_curve and auc from sklearn.metrics to create the ROC curve and also to calculate the Area Under Curve.  
  First, calculate the probabilities of prediction using predict\_proba. It will return a numpy array with two columns, the first column consists of probabilities of class 0 and the second column consists probabilities of class 1. As we have to measure how well our model distinguish between positive and negative class, so I have used probability of class 1 to plot the ROC curve.

The roc\_curve will generate ROC curve and returns fpr, tpr and threshold. We need fpr and tpr to calculate the area under the curve of this model, which now we have as returned by roc\_curve. So, we can use the value of fpr and tpr as input for auc function to calculate the area under the curve.  
Now we plot the ROC curve and analyze the performance of our model.

Chart, line chart

Description automatically generated

AUC → 95%

The area under the curve is 0.95 → 95%, which is incredible and indicates that our model is doing well.

* CAP:  
  First, calculate the total data points in the test data (100). Then calculate the number of data points of class 1 in the test data (32) and also calculate the number of data points of class 0 in the test data (68).

Now we start plotting our CAP curve. To begin, we create a random model based on the assumption that the correct detection of class 1 will increase linearly.  
Next we plot the perfect model. A perfect model is one that detects all class 1 data points in the same number of trials as the number of data points in the class. The perfect model requires exactly 32 trials to detect 32 class 1 data points.  
Now, finally we plot the results from the Logistic Regression. As in AUROC curve, we have to get the probability of class 1 and merge these values with y\_test using zip function.  
To calculate the y\_ values we use np.cumsum(). np.cumsum() produces an array of values by adding all of the array’s previous values to the current value. For instance, consider the array [1, 1, 1, 1, 1]. [1, 2, 3, 4, 5] would be the outcome of np.cumsum() function. In addition, we must append 0 to the array for the at the start point (0,0). The x\_values will be in the range of 0 to total + 1.

Chart, line chart

Description automatically generated

CAP curve

Now, as we have two methods to analyze the performance using CAP method:

* Area under curve: First we have to calculate all areas using auc function and then use these values to calculate the Accuracy Rate. The rate is at 0.90, which is quite near to one, indicating that our model is really effective.
* Plot: First, we have to find the index of 50% of total test data. Then we have to plot a vertical line till it intersect our trained model line and from that intersecting point, draw a horizontal line till it intersect y-axis. Then calculate the X% value by dividing the observed class 1 values by the total number of class 1 data points and multiplying by 100. We get 96.875%

Chart, line chart

Description automatically generated

CAP curve using plot method

CAP curve gave us 96.875% which implies that our model is too good. It shows us that our model is too good to classify positive and negative classes.

**Conclusion**The article shares brief understanding about performance measures and methods you can use to analyze the performance of your classification model. You can use any of the above mentioned methods to measure the performance of your model.