What is Machine Learning?

Two definitions of Machine Learning are offered.

Arthur Samuel described it as:

"the field of study that gives computers the ability to learn without being explicitly programmed." This is an older, informal definition.

Tom Mitchell provides a more modern definition:

"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."

Example: playing checkers.

E = the experience of playing many games of checkers

T = the task of playing checkers.

P = the probability that the program will win the next game.

In general, any machine learning problem can be assigned to one of two broad classifications:

Supervised learning and Unsupervised learning.

<https://github.com/patilankita79/ML_DataPreprocessing/blob/master/ML_DataPreprocessing/Data.csv>

Machine Learning is the hottest thing of this decade. Everybody wants to get on the bandwagon and start deploying machine learning models in their businesses. At the heart of this intricate process is data. Your machine learning tools are as good as the quality of your data. Sophisticated algorithms will not make up for poor data. Just like how precious stones found while digging go through several steps of cleaning process, data needs to also go through a few before it is ready for further use.

In this article I will try to simplify the exercise of data preprocessing, or in other words, the rituals programmers usually follow before it is ready to be used for machine learning models into 6 simple steps.

#### Step 1: Import Libraries

First step is usually importing the libraries that will be needed in the program. A library is essentially a collection of modules that can be called and used. A lot of the things in the programming world do not need to be written explicitly ever time they are required. There are functions for them, which can simply be invoked. This is a [list](https://medium.com/activewizards-machine-learning-company/top-15-python-libraries-for-data-science-in-in-2017-ab61b4f9b4a7)for most popular Python libraries for Data Science. Here’s a snippet of me importing the pandas library and assigning a shortcut “pd”.

import pandas as pd

#### Step 2: Import the Dataset

A lot of datasets come in CSV formats. We will need to locate the directory of the CSV file at first (it’s more efficient to keep the dataset in the same directory as your program) and read it using a method called read\_csv which can be found in the library called pandas.

import pandas as pd

dataset = pd.read\_csv('Medium.csv')

After inspecting our dataset carefully, we are going to create a matrix of features in our dataset (X) and create a dependent vector (Y) with their respective observations. To read the columns, we will use iloc of pandas (used to fix the indexes for selection) which takes two parameters — [row selection, column selection].

X = dataset.iloc[:, :-1].values

: as a parameter selects all. So the above piece of code selects all the rows. For columns we have :-1, which means all the columns except the last one. You can read more about the usage of iloc [here](https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/).

#### Step 3: Taking care of Missing Data in Dataset

Sometimes you may find some data are missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously you could remove the entire line of data but what if you are unknowingly removing crucial information? Of course we would not want to do that. One of the most common idea to handle the problem is to take a mean of all the values of the same column and have it to replace the missing data.

The library that we are going to use for the task is called [Scikit Learn](http://scikit-learn.org/stable/index.html" \t "_blank)preprocessing. It contains a class called Imputer which will help us take care of the missing data.

from sklearn.preprocessing import Imputer

A lot of the times the next step, as you will also see later on in the article, is to create an object of the same class to call the functions that are in that class. We will call our object imputer. The Imputer class can take a few parameters —   
i. missing\_values — We can either give it an integer or “NaN” for it to find the missing values.   
ii. strategy — we will find the average so we will set it to mean. We can also set it to median or most\_frequent (for mode) as necessary.   
iii. axis — we can either assign it 0 or 1, 0 to impute along columns and 1 to impute along rows.

imputer = Imputer(missing\_values = "NaN", strategy = "mean", axis = 0)

Now we will fit the imputer object to our data. Fit is basically training, or in other words, imposing the model to our data.

imputer = imputer.fit(X[:,1:3])

The code above will fit the imputer object to our matrix of features X. Since we used :, it will select all rows and 1:3 will select the second and the third column (why? because in python index starts from 0 so 1 would mean the second column and the upper-bound is excluded. If we wanted to include the third column instead, we would have written 1:4).

Now we will just replace the missing values with the mean of the column by the method transform.

X[:, 1:3] = imputer.transform(X[:, 1:3])

#### Step 4: Encoding categorical data

Sometimes our data is in qualitative form, that is we have texts as our data. We can find categories in text form. Now it gets complicated for machines to understand texts and process them, rather than numbers, since the models are based on mathematical equations and calculations. Therefore, we have to encode the categorical data.



This is an example of categorical data. In the first column, the data is in text form. We can see that there are five categories — Very, Somewhat, Not very, Not at all, Not sure — and hence the name categorical data.

So the way we do it, we will import the scikit library that we previously used. There’s a class in the library called LabelEncoder which we will use for the task.

from sklearn.preprocessing import LabelEncoder

As I have mentioned before, the next step is usually to create an object of that class. We will call our object labelencoder\_X.

labelencoder\_X = LabelEncoder()

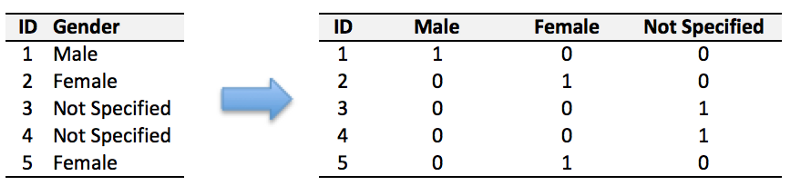
To do our task, there’s a method in the LabelEncoder class called fit\_transformwhich is what we will use. Once again, just like how we did it before, we will pass two parameters of X — row selection and column selection.

X[:,0] = labelencoder\_X.fit\_transform(X[:,0])

The above code will select all the rows (because :) of the first column (because 0) and fit the LabelEncoder to it and transform the values. The values will then immediately be encoded to 0,1,2,3… accordingly.

The text has been replaced by numbers as we wanted. But if there are more than two categories, we may have created a new problem in the way. As we keep assigning different integers to different categories, it may create a confusion. If one category is assigned 0 and another category is assigned 2, and since 2 is greater than 0, are we trying imply that the category assigned as 2 is greater? Of course we don’t! So this strategy might as well defeat its own purpose.

So instead of having one column with n number of categories, we will use n number of columns with only 1s and 0s to represent whether the category occurs or not.



Example of a Dummy encoding

To accomplish the task, we will import yet another library called OneHotEncoder.

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

Next we will create an object of that class, as usual, and assign it to onehotencoder. OneHotEncoder takes an important parameter called categorical\_features which takes the value of the index of the column of categories.

onehotencoder = OneHotEncoder(categorical\_features =[0])

The code above will select the first column to OneHotEncode the categories.

Just as we used fit\_transform for LabelEncoder, we will use it for OneHotEncoder as well but also have to additionally include toarray().

X = onehotencoder.fit\_transform(X).toarray()

If you check your dataset now, all your categories will have been encoded to 0s and 1s.

#### Step 5: Splitting the Dataset into Training set and Test Set

Now we need to split our dataset into two sets — a Training set and a Test set. We will train our machine learning models on our training set, i.e our machine learning models will try to understand any correlations in our training set and then we will test the models on our test set to check how accurately it can predict. A general rule of the thumb is to allocate 80% of the dataset to training set and the remaining 20% to test set. For this task, we will import test\_train\_split from model\_selection library of scikit.

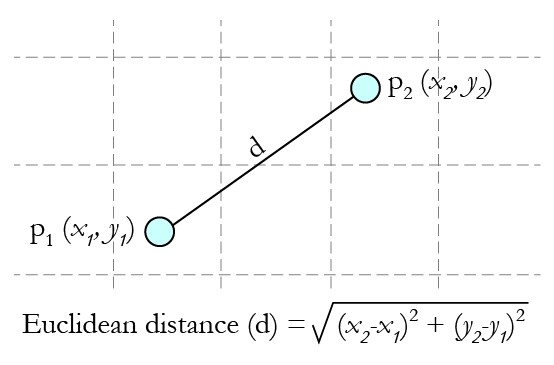
from sklearn.model\_selection import train\_test\_split

Now to build our training and test sets, we will create 4 sets— X\_train (training part of the matrix of features), X\_test (test part of the matrix of features), Y\_train (training part of the dependent variables associated with the X train sets, and therefore also the same indices) , Y\_test (test part of the dependent variables associated with the X test sets, and therefore also the same indices). We will assign to them the test\_train\_split, which takes the parameters — arrays (X and Y), test\_size (if we give it the value 0.5, meaning 50%, it would split the dataset into half. Since an ideal choice is to allocate 20% of the dataset to test set, it is usually assigned as 0.2. 0.25 would mean 25%, just saying).

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y, test\_size=0.2)

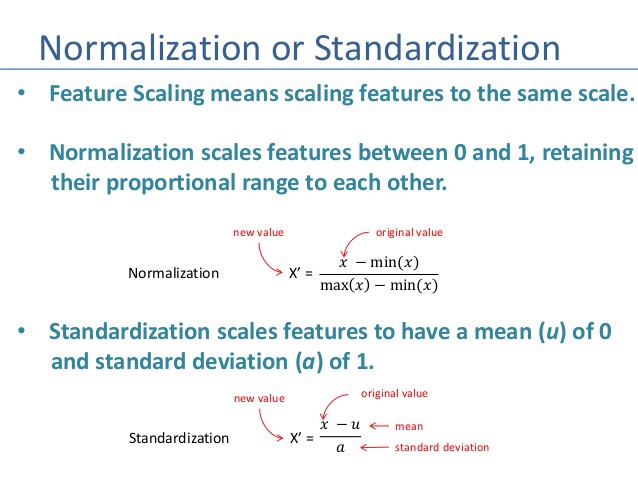
#### Step 6: Feature Scaling

The final step of data preprocessing is to apply the very important feature scaling.



The formula and graphical representation of Euclidean distance

**But what is it?**   
Itis a method used to standardize the range of independent variables or features of data.   
**But why is it necessary?** A lot of machine learning models are based on Euclidean distance. If, for example, the values in one column (x) is much higher than the value in another column (y), (x2-x1) squared will give a far greater value than (y2-y1) squared. So clearly, one square difference dominates over the other square difference. In the machine learning equations, the square difference with the lower value in comparison to the far greater value will almost be treated as if it does not exist. We do not want that to happen. That is why it is necessary to transform all our variables into the same scale. There are several ways of scaling the data. One way is called Standardization which may be used. For every observation of the selected column, our program will apply the formula of standardization and fit it to a scale.



To accomplish the job, we will import the class StandardScaler from the sckit preprocessing library and as usual create an object of that class.

from sklearn.preprocessing import StandardScaler  
sc\_X = StandardScaler()

Now we will fit and transform our X\_train set (It is important to note that when applying the Standard Scalar object on our training and test sets, we can simply transform our test set but for our training set we have to at first fit it and then transform the set). That will transform all the data to a same standardized scale.

X\_train = sc\_X.fit\_transform(X\_train)  
X\_test = sc\_X.transform(X\_test)

These are the general 6 steps of preprocessing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

# ML\_DataPreprocessing

* Data preprocessing is the crucial step in making a ML model.
* If there is no data preprocessing step on the data, your ML model won't work properly

|  |  |
| --- | --- |
| **Languages Used:** | Python, R |
| **IDE Used:** | Spyder, RStudio |

### Pre-processing steps to do to prepare any dataset on which we will build ML model

1. **Get the dataset**   
   The first step is always to get the dataset and try to understand dataset. Try to figure out what all are independent variables and what are dependent variables.In ML, some independent variables are used to predict a depedent variable.
   * The name of the dataset I have taken is 'Data.csv'.
   * Dataset contains four columns -> Country, Age, Salary, Purchased. Dataset has total 10 observations.
   * Dataset contains information of customers of some company and first three columns are information of a customer like country, age, salary and fourth column tells if the customer has purchased the product of the company or not.
   * In Data.csv, first three columns i.e. Country, Age, Salary are independent variables and fourth column Purchased is an indepedent variable
2. **Importing the libraries**
   * For Python, import the following three essential libraries
     + numpy -> contains mathematical tools
     + matplotlib -> to plot nice charts
     + pandas -> to import the dataset and manage the dataset
   * For R, we don't need to import any libraries
3. **Import the dataset**
   * For Python, you have to create the matrix of features and dependent variable vector
   * Hence for Data.csv, create the matrix of three independent variables and then create the dependent variable vector
4. **Dealing with missing data**
   * Your dataset may contain missing data. In Data.csv, we can see that there is one missing data in the age column for Spain and one missing data in salary column for Germany
   * One way to deal with missing data is to remove the lines of the observaton where there is some missing data but that can be dangerous because the data set may contain crucial information. So it is quite dangerous to remove observation
   * Another way to handle missing data is to take the mean of the columns. So in age column where we have missing entry for Spain, we will replace this missing data by the mean of all the values in the column age
   * In Python, Imputer class from Scikitlearn preprocessing library allows us to take care of missing data
   * In R, we can write ifelse condition to check missing data and take mean of that column if the condition holds true
5. **Encode categorical data**
   * Your dataset may contain quantitative and qualitative variables. Quantitative variables contain numeric values whereas qualitative variables contain the categories or levels within the data
   * ML models are based on mathematical equations, so it would cause some problem if we keep the text and use categorical variables in the equation because we only want numbers in the equations. That's why we need to encode categorical variables
   * In 'Data.csv', Country and Purchased are categorical variables because they simply contain categories
     + Country variable contains three categories - France, Spain, Germany
     + Purchased variable contains two categories - yes, no
   * In Python, categorical data can be encoded using LabelEncoder class for scikit learn preprocessing. Hence, python will give levels to these categories and the order of those levels is not important. But we have to prevent ML equations from thinking one level is greater than other or vice versa. To prevent this, we use dummy variables. So, for example, for Country column, instead of having one column, we will have 3 columns. This can be achieved with the help of OneHotEncoder class
   * We don't need to use OneHotEncoder for Purchased variable which is a dependent variable.Since it is a dependent variable ML will know that it is a category and there is no order between the categories of Purchased variable.
   * In R, we use factor function to transform a categorical variable to numeric variable
6. **Split the dataset into training data and test data**
   * ML is about a machine that is going to learn from the data to make predictions
   * We need to split the dataset into training set and test set. Using training set, we build the machine model and using test set, we test the performance of this machine learning model
   * We are building our ML model on training set by establishing some correlation between independent variables and dependent variables and once the ML model understands the correlation between independent variables and dependent variables we will test if the ML model can apply the correlations you understood based on training set on test set
   * In nutshell, we have to make two different datasets. The training set on which the machine learning model learns and test set on which we test if the ML model learned correctly the correlations
   * In Python, cross-validation class from scikit library is used to split the dataset into training and test set
   * In R, caTools library is used to split the dataset into training and test set
7. **Feature Scaling**
   * In the dataset, the variables age and salary are not on the same scale because the Age is ranging from 27 to 50 and Salary is ranging 40k to 90k
   * We need to have these variables in the same scale otherwise their distance(or Euclidean distance) would be dominated by the variable with high range
   * Standardization and normalization are two ways of doing feature scaling
   * In Python, we use StandardScaler class from scikit preprocessing library
   * In R, we use scale function

|  |  |  |
| --- | --- | --- |
| # -\*- coding: utf-8 -\*- | | |
|  | | | """ | |
|  | | | Created on Fri Jan 12 12:38:34 2018 | |
|  | | |  | |
|  | | | @author: patil | |
|  | | | """ | |
|  | | | # Data Preprocessing | |
|  | | |  | |
|  | | | # Importing the libraries | |
|  | | | import numpy as np | |
|  | | | import matplotlib.pyplot as plt | |
|  | | | import pandas as pd | |
|  | | |  | |
|  | | | # Importing the datasets | |
|  | | | dataset = pd.read\_csv('Data.csv') | |
|  | | |  | |
|  | | | # Creating the matrix of features. In the imported dataset Country, Age, Salary are independent variables | |
|  | | | X = dataset.iloc[:, :-1].values | |
|  | | |  | |
|  | | | # Creating the dependent variable | |
|  | | | y = dataset.iloc[:, 3].values | |
|  | | | ... | |
|  | | | # Taking care of missing data | |
|  | | | from sklearn.preprocessing import Imputer | |
|  | | |  | |
|  | | | # Creating the object of Imputer class | |
|  | | | imputer = Imputer(missing\_values = 'NaN', strategy = 'mean', axis = 0) | |
|  | | |  | |
|  | | | # fit imputer object to data X (Matrix of feature X) | |
|  | | | imputer = imputer.fit(X[:, 1:3]) | |
|  | | |  | |
|  | | | # Replace the missing data of column by mean | |
|  | | | X[:, 1:3] = imputer.transform(X[:, 1:3]) | |
|  | | |  | |
|  | | | # Encoding categorical data | |
|  | | | # Creating dummy variables using OneHotEncoder class | |
|  | | | from sklearn.preprocessing import LabelEncoder, OneHotEncoder | |
|  | | |  | |
|  | | | # Creating the object of LabelEncoder class | |
|  | | | labelencoder\_X = LabelEncoder() | |
|  | | |  | |
|  | | | # fit labelencoder\_X object to first coulmn Country of matrix X | |
|  | | | X[:, 0] = labelencoder\_X.fit\_transform(X[:, 0]) | |
|  | | |  | |
|  | | | # Creating dummy variables | |
|  | | | # Creating the object of OneHotEncoder class | |
|  | | | onehotencoder = OneHotEncoder(categorical\_features = [0]) | |
|  | | |  | |
|  | | | # fit onehotencoder object to first column - Country of matrix X | |
|  | | | X = onehotencoder.fit\_transform(X).toarray() | |
|  | | |  | |
|  | | | # Creating the object of LabelEncoder class | |
|  | | | labelencoder\_y = LabelEncoder() | |
|  | | |  | |
|  | | | # fit labelencoder\_y object to last coulmn Purchased, we will get encoded vector | |
|  | | | y = labelencoder\_y.fit\_transform(y) | |
|  | | |  | |
|  | | | # Splitting the dataset into training set and test set | |
|  | | | from sklearn.cross\_validation import train\_test\_split | |
|  | | |  | |
|  | | | # Choosing 20% data as test data, so we will have 80% data in training set | |
|  | | | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0) | |
|  | | |  | |
|  | | | # Feature Scaling | |
|  | | | from sklearn.preprocessing import StandardScaler | |
|  | | |  | |
|  | | | # Creating the object of StandardScaler | |
|  | | | sc\_X = StandardScaler() | |
|  | | |  | |
|  | | | # fit and transform training set | |
|  | | | X\_train = sc\_X.fit\_transform(X\_train) | |
|  | | |  | |
|  | | | # transform test set | |
|  | | | X\_test = sc\_X.transform(X\_test) | |
| **Country** | **Age** | | **Salary** | **Purchased** |
|  | | France | 44 | 72000 | No |
|  | | Spain | 27 | 48000 | Yes |
|  | | Germany | 30 | 54000 | No |
|  | | Spain | 38 | 61000 | No |
|  | | Germany | 40 |  | Yes |
|  | | France | 35 | 58000 | Yes |
|  | | Spain |  | 52000 | No |
|  | | France | 48 | 79000 | Yes |
|  | | Germany | 50 | 83000 | No |
|  | | France | 37 | 67000 | Yes |

# Introduction to k-Nearest Neighbors: Simplified (with implementation in Python)

[**TAVISH SRIVASTAVA**](https://www.analyticsvidhya.com/blog/author/tavish1/)**, MARCH 26, 2018**

**Note: This article was originally published on Oct 10, 2014 and updated on Mar 27th, 2018**

## Introduction

In four years of my career into analytics I have built more than 80% of classification models and just 15-20% regression models. These ratios can be more or less generalized throughout the industry. The reason of a bias towards classification models is that most analytical problem involves making a decision. For instance will a customer attrite or not, should we target customer X for digital campaigns, whether customer has a high potential or not etc. These analysis are more insightful and directly links to an implementation roadmap. In this article, we will talk about another widely used classification technique called K-nearest neighbors (KNN) . Our focus will be primarily on how does the algorithm work and how does the input parameter effect the output/prediction.

## Table of Contents

* When do we use KNN algorithm?
* How does the KNN algorithm work?
* How do we choose the factor K?
* Breaking it Down – Pseudo Code of KNN
* Implementation in Python from scratch
* Comparing our model with scikit-learn

## When do we use KNN algorithm?

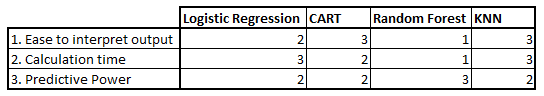
KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output

2. Calculation time

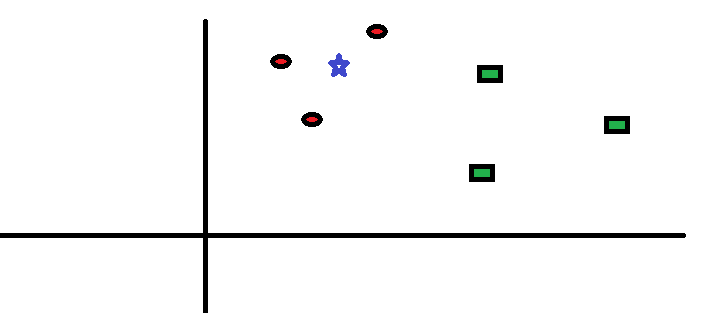
3. Predictive Power

Let us take a few examples to  place KNN in the scale :

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/Model-comparison.png)KNN algorithm fairs across all parameters of considerations. It is commonly used for its easy of interpretation and low calculation time.

## How does the KNN algorithm work?

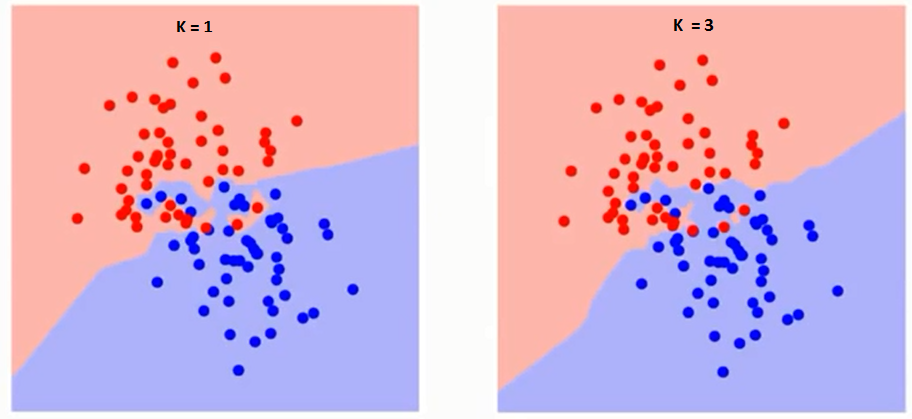
Let’s take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS) :

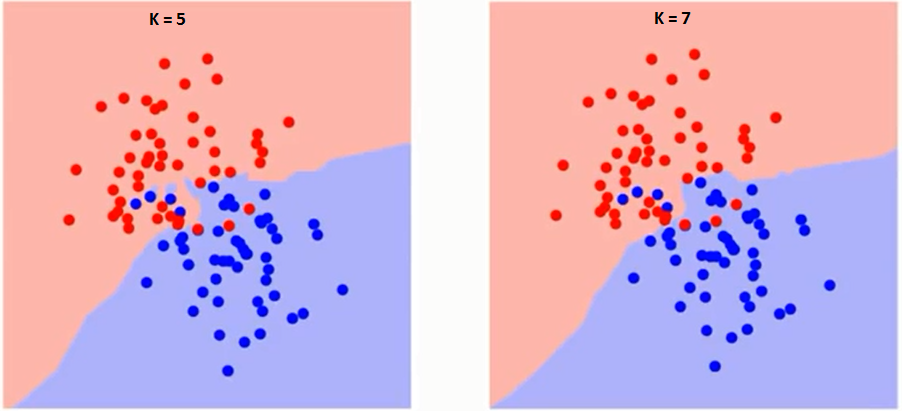
[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/scenario1.png)You intend to find out the class of the blue star (BS) . BS can either be RC or GS and nothing else. The “K” is KNN algorithm is the nearest neighbors we wish to take vote from. Let’s say K = 3. Hence, we will now make a circle with BS as center just as big as to enclose only three datapoints on the plane. Refer to following diagram for more details:

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/scenario2.png)The three closest points to BS is all RC. Hence, with good confidence level we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm. Next we will understand what are the factors to be considered to conclude the best K.

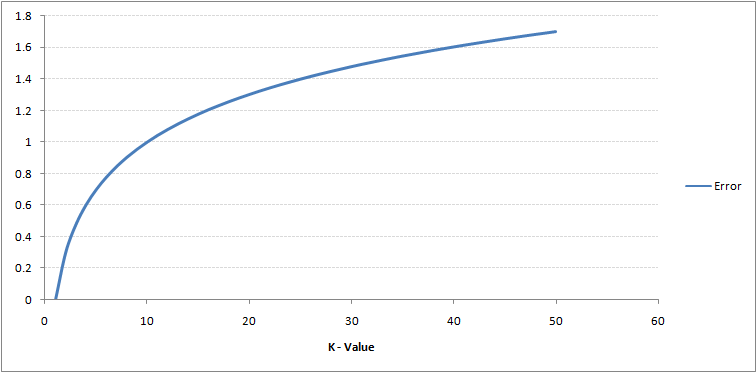
## How do we choose the factor K?

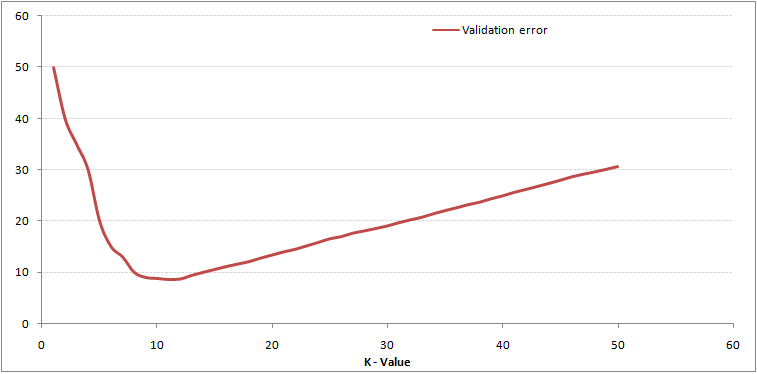
First let us try to understand what exactly does K influence in the algorithm. If we see the last example, given that all the 6 training observation remain constant, with a given K value we can make boundaries of each class. These boundaries will segregate RC from GS. The same way, let’s try to see the effect of value “K” on the class boundaries. Following are the different boundaries separating the two classes with different values of K.

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/K-judgement.png)

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/K-judgement2.png)

If you watch carefully, you can see that the boundary becomes smoother with increasing value of K. With K increasing to infinity it finally becomes all blue or all red depending on the total majority.  The training error rate and the validation error rate are two parameters we need to access on different K-value. Following is the curve for the training error rate with varying value of K :

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/training-error.png)As you can see, the error rate at K=1 is always zero for the training sample. This is because the closest point to any training data point is itself.Hence the prediction is always accurate with K=1. If validation error curve would have been similar, our choice of K would have been 1. Following is the validation error curve with varying value of K:

[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/training-error_11.png)This makes the story more clear. At K=1, we were overfitting the boundaries. Hence, error rate initially decreases and reaches a minima. After the minima point, it then increase with increasing K. To get the optimal value of K, you can segregate the training and validation from the initial dataset. Now plot the validation error curve to get the optimal value of K. This value of K should be used for all predictions.

## Breaking it Down – Pseudo Code of KNN

We can implement a KNN model by following the below steps:

1. Load the data
2. Initialise the value of k
3. For getting the predicted class, iterate from 1 to total number of training data points
   1. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it’s the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
   2. Sort the calculated distances in ascending order based on distance values
   3. Get top k rows from the sorted array
   4. Get the most frequent class of these rows
   5. Return the predicted class

## Implementation in Python from scratch

We will be using the popular Iris dataset for building our KNN model. You can download it from [here](https://gist.githubusercontent.com/gurchetan1000/ec90a0a8004927e57c24b20a6f8c8d35/raw/fcd83b35021a4c1d7f1f1d5dc83c07c8ffc0d3e2/iris.csv).

# Importing libraries

import pandas as pd

import numpy as np

import math

import operator

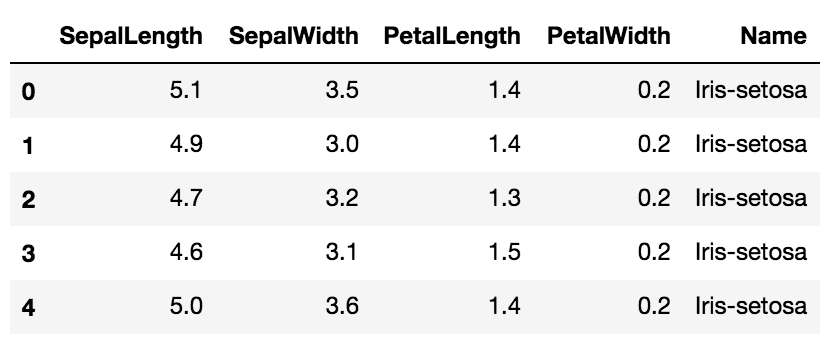
#### Start of STEP 1

# Importing data

data = pd.read\_csv("iris.csv")

#### End of STEP 1

data.head()



# Defining a function which calculates euclidean distance between two data points

def euclideanDistance(data1, data2, length):

distance = 0

for x in range(length):

distance += np.square(data1[x] - data2[x])

return np.sqrt(distance)

# Defining our KNN model

def knn(trainingSet, testInstance, k):

distances = {}

sort = {}

length = testInstance.shape[1]

#### Start of STEP 3

# Calculating euclidean distance between each row of training data and test data

for x in range(len(trainingSet)):

#### Start of STEP 3.1

dist = euclideanDistance(testInstance, trainingSet.iloc[x], length)

distances[x] = dist[0]

#### End of STEP 3.1

#### Start of STEP 3.2

# Sorting them on the basis of distance

sorted\_d = sorted(distances.items(), key=operator.itemgetter(1))

#### End of STEP 3.2

neighbors = []

#### Start of STEP 3.3

# Extracting top k neighbors

for x in range(k):

neighbors.append(sorted\_d[x][0])

#### End of STEP 3.3

classVotes = {}

#### Start of STEP 3.4

# Calculating the most freq class in the neighbors

for x in range(len(neighbors)):

response = trainingSet.iloc[neighbors[x]][-1]

if response in classVotes:

classVotes[response] += 1

else:

classVotes[response] = 1

#### End of STEP 3.4

#### Start of STEP 3.5

sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)

return(sortedVotes[0][0], neighbors)

#### End of STEP 3.5

# Creating a dummy testset

testSet = [[7.2, 3.6, 5.1, 2.5]]

test = pd.DataFrame(testSet)

#### Start of STEP 2

# Setting number of neighbors = 1

k = 1

#### End of STEP 2

# Running KNN model

result,neigh = knn(data, test, k)

# Predicted class

print(result)

-> Iris-virginica

# Nearest neighbor

print(neigh)

-> [141]

Now we will try to alter the k values, and see how the prediction changes.

# Setting number of neighbors = 3

k = 3

# Running KNN model

result,neigh = knn(data, test, k)

# Predicted class

print(result) -> Iris-virginica

# 3 nearest neighbors

print(neigh)

-> [141, 139, 120]

# Setting number of neighbors = 5

k = 5

# Running KNN model

result,neigh = knn(data, test, k)

# Predicted class

print(result) -> Iris-virginica

# 5 nearest neighbors

print(neigh)

-> [141, 139, 120, 145, 144]

## Comparing our model with scikit-learn

from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n\_neighbors=3)

neigh.fit(data.iloc[:,0:4], data['Name'])

# Predicted class

print(neigh.predict(test))

-> ['Iris-virginica']

# 3 nearest neighbors

print(neigh.kneighbors(test)[1])

-> [[141 139 120]]

We can see that both the models predicted the same class (‘Iris-virginica’) and the same nearest neighbors ( [141 139 120] ). Hence we can conclude that our model runs as expected.

## End Notes

KNN algorithm is one of the simplest classification algorithm. Even with such simplicity, it can give highly competitive results. KNN algorithm can also be used for regression problems. The only difference from the discussed methodology will be using averages of nearest neighbors rather than voting from nearest neighbors. KNN can be coded in a single line on R. I am yet to explore how can we use KNN algorithm on SAS.

Did you find the article useful? Have you used any other machine learning tool recently? Do you plan to use KNN in any of your business problems? If yes, share with us how you plan to go about it.

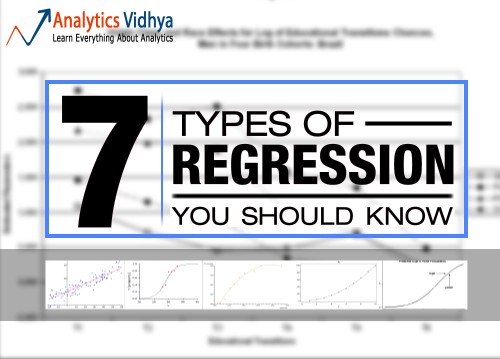
# 7 Types of Regression Techniques you should know!

[**SUNIL RAY**](https://www.analyticsvidhya.com/blog/author/sunil-ray/)**, AUGUST 14, 2015**

## Introduction

Linear and Logistic regressions are usually the first algorithms people learn in predictive modeling. Due to their popularity, a lot of analysts even end up thinking that they are the only form of regressions. The ones who are slightly more involved think that they are the most important amongst all forms of regression analysis.

The truth is that there are innumerable forms of regressions, which can be performed. Each form has its own importance and a specific condition where they are best suited to apply. In this article, I have explained the most commonly used 7 forms of regressions in a simple manner. Through this article, I also hope that people develop an idea of the breadth of regressions, instead of just applying linear / logistic regression to every problem they come across and hoping that they would just fit!

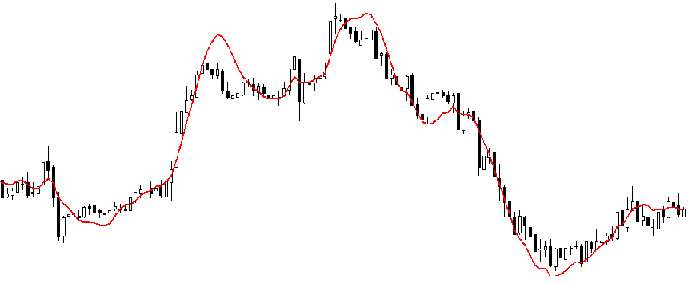
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/regression.jpg)

## Table of Contents

1. What is Regression Analysis?
2. Why do we use Regression Analysis?
3. What are the types of Regressions?
   * Linear Regression
   * Logistic Regression
   * Polynomial Regression
   * Stepwise Regression
   * Ridge Regression
   * Lasso Regression
   * ElasticNet Regression
4. How to select the right Regression Model?

## What is Regression Analysis?

Regression analysis is a form of predictive modelling technique which investigates the relationship between a **dependent**(target) and **independent variable (s)** (predictor). This technique is used for forecasting, time series modelling and finding the [causal effect relationship](https://www.analyticsvidhya.com/blog/2015/06/establish-causality-events/) between the variables. For example, relationship between rash driving and number of road accidents by a driver is best studied through regression.

Regression analysis is an important tool for modelling and analyzing data. Here, we fit a curve / line to the data points, in such a manner that the differences between the distances of data points from the curve or line is minimized.  I’ll explain this in more details in coming sections.[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Regression_Line.png)

## Why do we use Regression Analysis?

As mentioned above, regression analysis estimates the relationship between two or more variables. Let’s understand this with an easy example:

Let’s say, you want to estimate growth in sales of a company based on current economic conditions. You have the recent company data which indicates that the growth in sales is around two and a half times the growth in the economy. Using this insight, we can predict future sales of the company based on current & past information.

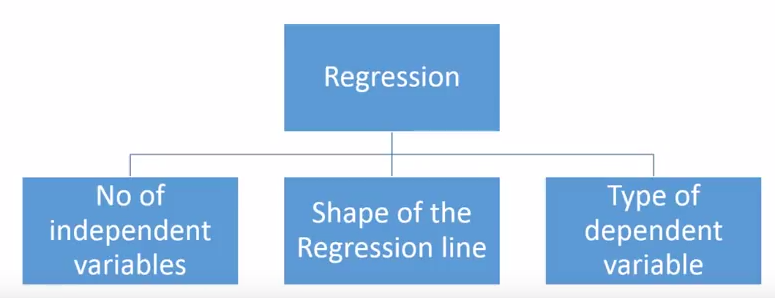
There are multiple benefits of using regression analysis. They are as follows:

1. It indicates the **significant relationships** between dependent variable and independent variable.
2. It indicates the **strength of impact** of multiple independent variables on a dependent variable.

Regression analysis also allows us to compare the effects of variables measured on different scales, such as the effect of price changes and the number of promotional activities. These benefits help market researchers / data analysts / data scientists to eliminate and evaluate the best set of variables to be used for building predictive models.

## How many types of regression techniques do we have?

There are various kinds of regression techniques available to make predictions. These techniques are mostly driven by three metrics (number of independent variables, type of dependent variables and shape of regression line). We’ll discuss them in detail in the following sections.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Regression_Type.png)

For the creative ones, you can even cook up new regressions, if you feel the need to use a combination of the parameters above, which people haven’t used before. But before you start that, let us understand the most commonly used regressions:

## 1. Linear Regression

It is one of the most widely known modeling technique. Linear regression is usually among the first few topics which people pick while learning predictive modeling. In this technique, the dependent variable is continuous, independent variable(s) can be [continuous or discrete](https://en.wikipedia.org/wiki/Continuous_and_discrete_variables), and nature of regression line is linear.

Linear Regression establishes a relationship between **dependent variable (Y)** and one or more **independent variables (X)** using a **best fit straight line** (also known as regression line).

It is represented by an equation **Y=a+b\*X + e**, where a is intercept, b is slope of the line and e is error term. This equation can be used to predict the value of target variable based on given predictor variable(s).

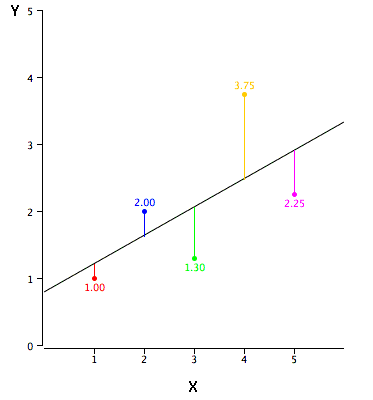
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Linear_Regression1.png)

The difference between simple linear regression and multiple linear regression is that, multiple linear regression has (>1) independent variables, whereas simple linear regression has only 1 independent variable.  Now, the question is “How do we obtain best fit line?”.

#### How to obtain best fit line (Value of a and b)?

This task can be easily accomplished by Least Square Method. It is the most common method used for fitting a regression line. It calculates the best-fit line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line. Because the deviations are first squared, when added, there is no cancelling out between positive and negative values.

[Least_Square](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Least_Square.png)

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/reg_error.gif)

We can evaluate the model performance using the metric **R-square**. To know more details about these metrics, you can read: Model Performance metrics [Part 1](https://www.analyticsvidhya.com/blog/2015/01/model-performance-metrics-classification/), [Part 2](https://www.analyticsvidhya.com/blog/2015/01/model-perform-part-2/) .

#### Important Points:

* There must be **linear relationship** between independent and dependent variables
* Multiple regression suffers from **multicollinearity, autocorrelation, heteroskedasticity**.
* Linear Regression is very sensitive to **Outliers**. It can terribly affect the regression line and eventually the forecasted values.
* Multicollinearity can increase the variance of the coefficient estimates and make the estimates very sensitive to minor changes in the model. The result is that the coefficient estimates are unstable
* In case of multiple independent variables, we can go with **forward selection**, **backward elimination** and **step wise approach** for selection of most significant independent variables.

## 2. Logistic Regression

Logistic regression is used to find the probability of event=Success and event=Failure. We should use logistic regression when the dependent variable is binary (0/ 1, True/ False, Yes/ No) in nature. Here the value of Y ranges from 0 to 1 and it can represented by following equation.

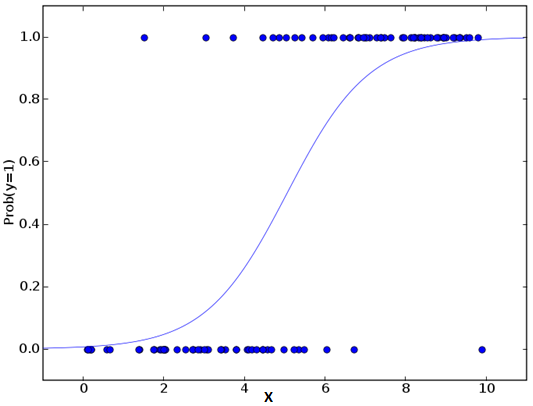
odds= p/ (1-p) = probability of event occurrence / probability of not event occurrence

ln(odds) = ln(p/(1-p))

logit(p) = ln(p/(1-p)) = b0+b1X1+b2X2+b3X3....+bkXk

Above, p is the probability of presence of the characteristic of interest. A question that you should ask here is “why have we used log in the equation?”.

Since we are working here with a binomial distribution (dependent variable), we need to choose a link function which is best suited for this distribution. And, it is [**logit**](https://en.wikipedia.org/wiki/Logistic_function) function. In the equation above, the parameters are chosen to maximize the likelihood of observing the sample values rather than minimizing the sum of squared errors (like in ordinary regression).

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Logistic_Regression.png)

#### Important Points:

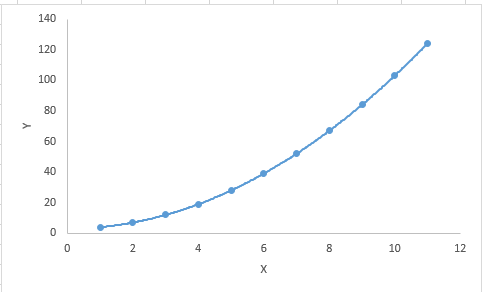
* It is widely used for **classification problems**
* Logistic regression doesn’t require linear relationship between dependent and independent variables.  It can handle various types of relationships because it applies a non-linear log transformation to the predicted odds ratio
* To avoid over fitting and under fitting, we should include all significant variables. A good approach to ensure this practice is to use a step wise method to estimate the logistic regression
* It requires **large sample sizes** because maximum likelihood estimates are less powerful at low sample sizes than ordinary least square
* The independent variables should not be correlated with each other i.e. **no multi collinearity**.  However, we have the options to include interaction effects of categorical variables in the analysis and in the model.
* If the values of dependent variable is ordinal, then it is called as **Ordinal logistic regression**
* If dependent variable is multi class then it is known as **Multinomial Logistic regression**.

## 3. Polynomial Regression

A regression equation is a polynomial regression equation if the power of independent variable is more than 1. The equation below represents a polynomial equation:

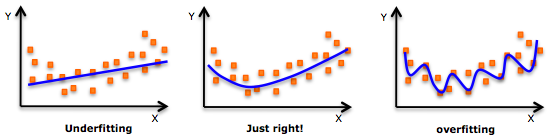
y=a+b\*x^2

In this regression technique, the best fit line is not a straight line. It is rather a curve that fits into the data points.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Polynomial.png)

#### Important Points:

* While there might be a temptation to fit a higher degree polynomial to get lower error, this can result in over-fitting. Always plot the relationships to see the fit and focus on making sure that the curve fits the nature of the problem. Here is an example of how plotting can help:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/underfitting-overfitting.png)

* Especially look out for curve towards the ends and see whether those shapes and trends make sense. Higher polynomials can end up producing wierd results on extrapolation.

## 4. Stepwise Regression

This form of regression is used when we deal with multiple independent variables. In this technique, the selection of independent variables is done with the help of an automatic process, which involves no human intervention.

This feat is achieved by observing statistical values like R-square, t-stats and AIC metric to discern significant variables. Stepwise regression basically fits the regression model by adding/dropping co-variates one at a time based on a specified criterion. Some of the most commonly used Stepwise regression methods are listed below:

* Standard stepwise regression does two things. It adds and removes predictors as needed for each step.
* Forward selection starts with most significant predictor in the model and adds variable for each step.
* Backward elimination starts with all predictors in the model and removes the least significant variable for each step.

The aim of this modeling technique is to maximize the prediction power with minimum number of predictor variables. It is one of the method to handle[higher dimensionality](https://www.analyticsvidhya.com/blog/2015/07/dimension-reduction-methods/) of data set.

## 5. Ridge Regression

Ridge Regression is a technique used when the data suffers from multicollinearity ( independent variables are highly correlated). In multicollinearity, even though the least squares estimates (OLS) are unbiased, their variances are large which deviates the observed value far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.

Above, we saw the equation for linear regression. Remember? It can be represented as:

y=a+ b\*x

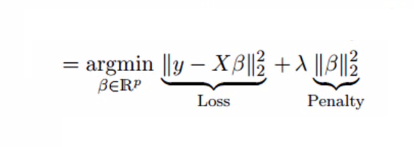
This equation also has an error term. The complete equation becomes:

y=a+b\*x+e (error term),  [error term is the value needed to correct for a prediction error between the observed and predicted value]

=> y=a+y= a+ b1x1+ b2x2+....+e, for multiple independent variables.

In a linear equation, prediction errors can be decomposed into two sub components. First is due to the **biased**and second is due to the **variance**. Prediction error can occur due to any one of these two or both components. Here, we’ll discuss about the error caused due to variance.

Ridge regression solves the multicollinearity problem through [shrinkage parameter](https://en.wikipedia.org/wiki/Shrinkage_estimator) λ (lambda). Look at the equation below.

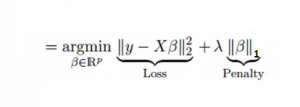
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Ridge2.png)

In this equation, we have two components. First one is least square term and other one is lambda of the summation of β2 (beta- square) where β is the coefficient. This is added to least square term in order to shrink the parameter to have a very low variance.

#### Important Points:

* The assumptions of this regression is same as least squared regression except normality is not to be assumed
* It shrinks the value of coefficients but doesn’t reaches zero, which suggests no feature selection feature
* This is a regularization method and uses [l2 regularization](https://en.wikipedia.org/wiki/Regularization_(mathematics)).

## 6. Lasso Regression

Similar to Ridge Regression, Lasso (Least Absolute Shrinkage and Selection Operator) also penalizes the absolute size of the regression coefficients. In addition, it is capable of reducing the variability and improving the accuracy of linear regression models.  Look at the equation below: [](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Lasso.png)Lasso regression differs from ridge regression in a way that it uses absolute values in the penalty function, instead of squares. This leads to penalizing (or equivalently constraining the sum of the absolute values of the estimates) values which causes some of the parameter estimates to turn out exactly zero. Larger the penalty applied, further the estimates get shrunk towards absolute zero. This results to variable selection out of given n variables.

#### Important Points:

* The assumptions of this regression is same as least squared regression except normality is not to be assumed
* It shrinks coefficients to zero (exactly zero), which certainly helps in feature selection
* This is a regularization method and uses [l1 regularization](https://en.wikipedia.org/wiki/Regularization_(mathematics))
* If group of predictors are highly correlated, lasso picks only one of them and shrinks the others to zero

## 7. ElasticNet Regression

ElasticNet is hybrid of Lasso and Ridge Regression techniques. It is trained with L1 and L2 prior as regularizer. Elastic-net is useful when there are multiple features which are correlated. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.

[Elastic_Net](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Elastic_Net.png)

A practical advantage of trading-off between Lasso and Ridge is that, it allows Elastic-Net to inherit some of Ridge’s stability under rotation.

#### Important Points:

* It encourages group effect in case of highly correlated variables
* There are no limitations on the number of selected variables
* It can suffer with double shrinkage

Beyond these 7 most commonly used regression techniques, you can also look at other models like [Bayesian](https://en.wikipedia.org/wiki/Bayesian_linear_regression), [Ecological](https://en.wikipedia.org/wiki/Ecological_regression) and [Robust regression](https://en.wikipedia.org/wiki/Robust_regression).

## How to select the right regression model?

Life is usually simple, when you know only one or two techniques. One of the training institutes I know of tells their students – if the outcome is continuous – apply linear regression. If it is binary – use logistic regression! However, higher the number of options available at our disposal, more difficult it becomes to choose the right one. A similar case happens with regression models.

Within multiple types of regression models, it is important to choose the best suited technique based on type of independent and dependent variables, dimensionality in the data and other essential characteristics of the data. Below are the key factors that you should practice to select the right regression model:

1. Data exploration is an inevitable part of building predictive model. It should be you first step before selecting the right model like identify the relationship and impact of variables
2. To compare the goodness of fit for different models, we can analyse different metrics like statistical significance of parameters, R-square, Adjusted r-square, AIC, BIC and error term. Another one is the [Mallow’s Cp](http://support.minitab.com/en-us/minitab/17/topic-library/modeling-statistics/regression-and-correlation/goodness-of-fit-statistics/what-is-mallows-cp/) criterion. This essentially checks for possible bias in your model, by comparing the model with all possible submodels (or a careful selection of them).
3. Cross-validation is the best way to evaluate models used for prediction. Here you divide your data set into two group (train and validate). A simple mean squared difference between the observed and predicted values give you a measure for the prediction accuracy.
4. If your data set has multiple confounding variables, you should not choose automatic model selection method because you do not want to put these in a model at the same time.
5. It’ll also depend on your objective. It can occur that a less powerful model is easy to implement as compared to a highly statistically significant model.
6. Regression regularization methods(Lasso, Ridge and ElasticNet) works well in case of high dimensionality and multicollinearity among the variables in the data set.

## End Note

By now, I hope you would have got an overview of regression. These regression techniques should be applied considering the conditions of data. One of the best trick to find out which technique to use, is by checking the family of variables i.e. discrete or continuous.

In this article, I discussed about 7 types of regression and some key facts associated with each technique. As somebody who’s new in this industry, I’d advise you to learn these techniques and later implement them in your models.

# Understanding Support Vector Machine algorithm from examples (along with code)

[**SUNIL RAY**](https://www.analyticsvidhya.com/blog/author/sunil-ray/)**, SEPTEMBER 13, 2017**

**Note: This article was originally published on Oct 6th, 2015 and updated on Sept 13th, 2017**

## Introduction

Mastering machine learning algorithms isn’t a myth at all. Most of the beginners start by learning regression. It is simple to learn and use, but does that solve our purpose? Of course not! Because, you can do so much more than just Regression!

Think of machine learning algorithms as an armory packed with axes, sword, blades, bow, dagger etc. You have various tools, but you ought to learn to use them at the right time. As an analogy, think of ‘Regression’ as a sword capable of slicing and dicing data efficiently, but incapable of dealing with highly complex data. On the contrary, ‘Support Vector Machines’ is like a sharp knife – it works on smaller datasets, but on them, it can be much more stronger and powerful in building models.

By now, I hope you’ve now mastered [Random Forest](https://www.analyticsvidhya.com/blog/2015/09/random-forest-algorithm-multiple-challenges/), [Naive Bayes Algorithm](https://www.analyticsvidhya.com/blog/2015/09/naive-bayes-explained/) and [Ensemble Modeling](https://www.analyticsvidhya.com/blog/2015/09/questions-ensemble-modeling/). If not, I’d suggest you to take out few minutes and read about them as well. In this article, I shall guide you through the basics to advanced knowledge of a crucial machine learning algorithm, support vector machines.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/learn5.jpg)

## Table of Contents

1. What is Support Vector Machine?
2. How does it work?
3. How to implement SVM in Python and R?
4. How to tune Parameters of SVM?
5. Pros and Cons associated with SVM

## What is Support Vector Machine?

“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However,  it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_1.png)

Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

You can look at [definition of support vectors](https://www.analyticsvidhya.com/blog/2014/10/support-vector-machine-simplified/) and a few examples of its working here.

## How does it work?

Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is “How can we identify the right hyper-plane?”. Don’t worry, it’s not as hard as you think!

Let’s understand:

* **Identify the right hyper-plane (Scenario-1):**Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.  
  You need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.
* **Identify the right hyper-plane (Scenario-2):**Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?

Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Let’s look at the below snapshot:[[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_4.png)](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_4.png)Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

* **Identify the right hyper-plane (Scenario-3):**Hint:Use the rules as discussed in previous section to identify the right hyper-plane

**[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_5.png)**Some of you may have selected the hyper-plane **B**as it has higher margin compared to **A.**But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

* **Can we classify two classes (Scenario-4)?:**Below, I am unable to segregate the two classes using a straight line, as one of star lies in the territory of other(circle) class as an outlier.  **[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_61.png)**As I have already mentioned, one star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin. Hence, we can say, SVM is robust to outliers.  
  **[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_71.png)**
* **Find the hyper-plane to segregate to classes (Scenario-5):**In the scenario below, we can’t have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.**[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_8.png)**SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature z=x^2+y^2. Now, let’s plot the data points on axis x and z:  
  [[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_9.png)](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_9.png)In above plot, points to consider are:
  + All values for z would be positive always because z is the squared sum of both x and y
  + In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

In SVM, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, SVM has a technique called the [**kernel**](https://en.wikipedia.org/wiki/Kernel_method)**trick**. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you’ve defined.

When we look at the hyper-plane in original input space it looks like a circle:  
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_10.png)

Now, let’s  look at the methods to apply SVM algorithm in a data science challenge.

## How to implement SVM in Python and R?

In Python, scikit-learn is a widely used library for implementing machine learning algorithms, SVM is also available in scikit-learn library and follow the same structure (Import library, object creation, fitting model and prediction). Let’s look at the below code:

#Import Library

from sklearn import svm

#Assumed you have, X (predictor) and Y (target) for training data set and x\_test(predictor) of test\_dataset

# Create SVM classification object

model = svm.svc(kernel='linear', c=1, gamma=1)

# there is various option associated with it, like changing kernel, gamma and C value. Will discuss more # about it in next section.Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x\_test)

The e1071 package in R is used to create Support Vector Machines with ease. It has helper functions as well as code for the Naive Bayes Classifier. The creation of a support vector machine in R and Python follow similar approaches, let’s take a look now at the following code:

#Import Library

require(e1071) #Contains the SVM

Train <- read.csv(file.choose())

Test <- read.csv(file.choose())

# there are various options associated with SVM training; like changing kernel, gamma and C value.

# create model

model <- svm(Target~Predictor1+Predictor2+Predictor3,data=Train,kernel='linear',gamma=0.2,cost=100)

#Predict Output

preds <- predict(model,Test)

table(preds)

### How to tune Parameters of SVM?

Tuning parameters value for machine learning algorithms effectively improves the model performance. Let’s look at the list of parameters available with SVM.

sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma=0.0, coef0=0.0, shrinking=True, probability=False,tol=0.001, cache\_size=200, class\_weight=None, verbose=False, max\_iter=-1, random\_state=None)

I am going to discuss about some important parameters having higher impact on model performance, “kernel”, “gamma” and “C”.

**kernel**: We have already discussed about it. Here, we have various options available with kernel like, “linear”, “rbf”,”poly” and others (default value is “rbf”).  Here “rbf” and “poly” are useful for non-linear hyper-plane. Let’s look at the example, where we’ve used linear kernel on two feature of iris data set to classify their class.

**Example:**Have linear kernel

import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm, datasets

# import some data to play with

iris = datasets.load\_iris()

X = iris.data[:, :2] # we only take the first two features. We could

# avoid this ugly slicing by using a two-dim dataset

y = iris.target

# we create an instance of SVM and fit out data. We do not scale our

# data since we want to plot the support vectors

C = 1.0 # SVM regularization parameter

svc = svm.SVC(kernel='linear', C=1,gamma=0).fit(X, y)

# create a mesh to plot in

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

h = (x\_max / x\_min)/100

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

np.arange(y\_min, y\_max, h))

plt.subplot(1, 1, 1)

Z = svc.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

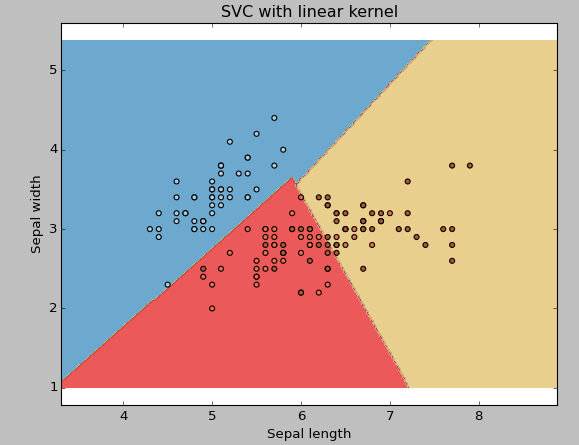
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)

plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())

plt.title('SVC with linear kernel')

plt.show()[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_111.png)

**Example:**Have rbf kernel

Change the kernel type to rbf in below line and look at the impact.

svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)

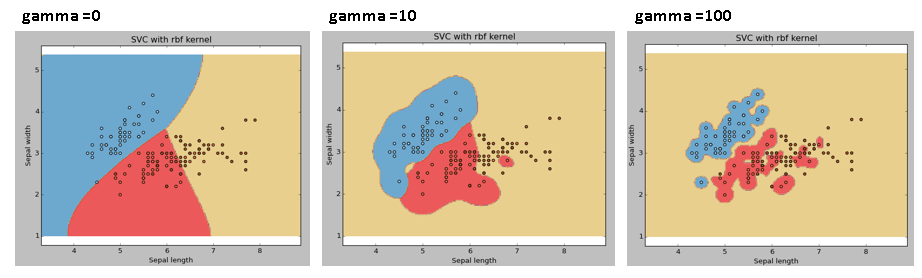
### [SVM_12](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_12.png)

I would suggest you to go for linear kernel if you have large number of features (>1000) because it is more likely that the data is linearly separable in high dimensional space. Also, you can RBF but do not forget to cross validate for its parameters as to avoid over-fitting.

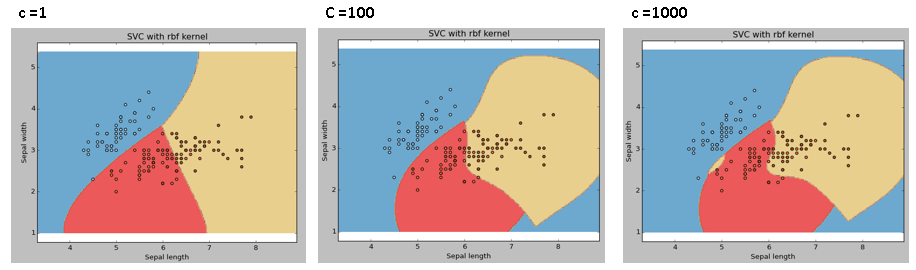
**gamma**: Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.

**Example:**Let’s difference if we have gamma different gamma values like 0, 10 or 100.

svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_15.png)

**C:**Penalty parameter C of the error term. It also controls the trade off between smooth decision boundary and classifying the training points correctly.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_18.png)

We should always look at the cross validation score to have effective combination of these parameters and avoid over-fitting.

In R, SVMs can be tuned in a similar fashion as they are in Python. Mentioned below are the respective parameters for e1071 package:

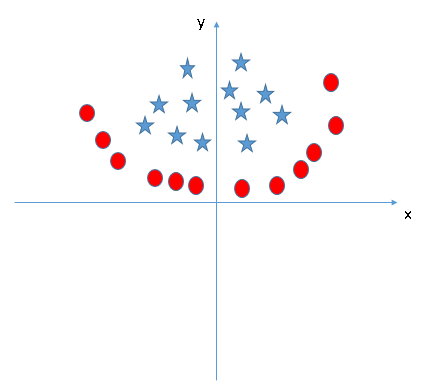
* The kernel parameter can be tuned to take “Linear”,”Poly”,”rbf” etc.
* The gamma value can be tuned by setting the “Gamma” parameter.
* The C value in Python is tuned by the “Cost” parameter in R.

## Pros and Cons associated with SVM

* **Pros:**
  + It works really well with clear margin of separation
  + It is effective in high dimensional spaces.
  + It is effective in cases where number of dimensions is greater than the number of samples.
  + It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
* **Cons:**
  + It doesn’t perform well, when we have large data set because the required training time is higher
  + It also doesn’t perform very well, when the data set has more noise i.e. target classes are overlapping
  + SVM doesn’t directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python scikit-learn library.

## Practice Problem

Find right additional feature to have a hyper-plane for segregating the classes in below snapshot:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_19.png)

Answer the variable name in the comments section below. I’ll shall then reveal the answer.

## End Notes

In this article, we looked at the machine learning algorithm, Support Vector Machine in detail.  I discussed its concept of working, process of implementation in python, the tricks to make the model efficient by tuning its parameters, Pros and Cons, and finally a problem to solve. I would suggest you to use SVM and analyse the power of this model by tuning the parameters. I also want to hear your experience with SVM, how have you tuned parameters to avoid over-fitting and reduce the training time?

# Practical Guide on Data Preprocessing in Python using Scikit Learn

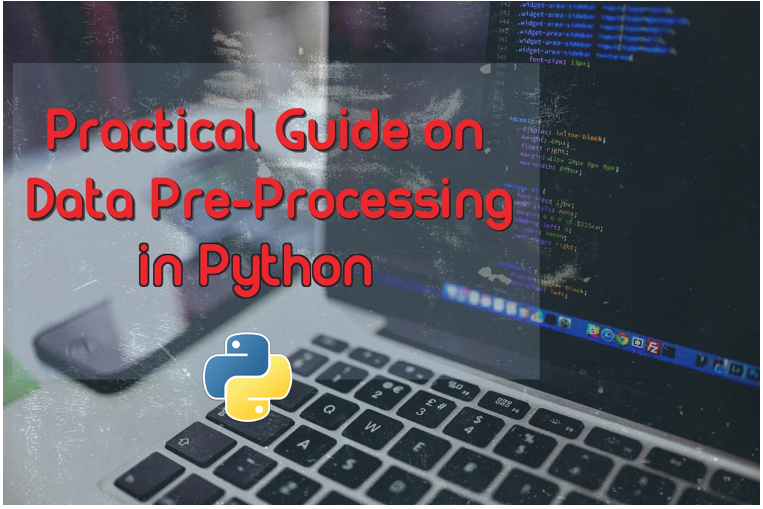
[**SYED DANISH**](https://www.analyticsvidhya.com/blog/author/syed4194/)**, JULY 18, 2016**

## Introduction

This article primarily focuses on data pre-processing techniques in python. Learning algorithms have affinity towards certain data types on which they perform incredibly well. They are also known to give reckless predictions with unscaled or unstandardized features. Algorithm like XGBoost, specifically requires dummy encoded data while algorithm like decision tree doesn’t seem to care at all (sometimes)!

In simple words, pre-processing refers to the transformations applied to your data before feeding it to the algorithm. In python, scikit-learn library has a pre-built functionality under [sklearn.preprocessing](http://scikit-learn.org/stable/modules/preprocessing.html" \t "_blank). There are many more options for pre-processing which we’ll explore.

After finishing this article, you will be equipped with the basic techniques of data pre-processing and their in-depth understanding. For your convenience, I’ve attached some resources for in-depth learning of machine learning algorithms and designed few exercises to get a good grip of the concepts.



## Available Data set

For this article, I have used a subset of the [Loan Prediction](http://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii) (missing value observations are dropped) data set from You can download the final training and testing data set from here: [Download Data](https://www.analyticsvidhya.com/wp-content/uploads/2016/07/loan_prediction-1.zip)

***Note :***Testing data that you are provided is the subset of the training data from Loan Prediction problem.

Now, lets get started by importing important packages and the data set.

# Importing pandas

>> import pandas as pd

# Importing training data set

>> X\_train=pd.read\_csv('X\_train.csv')

>> Y\_train=pd.read\_csv('Y\_train.csv')

# Importing testing data set

>> X\_test=pd.read\_csv('X\_test.csv')

>> Y\_test=pd.read\_csv('Y\_test.csv')

Lets take a closer look at our data set.

>> print (X\_train.head())

Loan\_ID Gender Married Dependents Education Self\_Employed

15 LP001032 Male No 0 Graduate No

248 LP001824 Male Yes 1 Graduate No

590 LP002928 Male Yes 0 Graduate No

246 LP001814 Male Yes 2 Graduate No

388 LP002244 Male Yes 0 Graduate No

ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term

15 4950 0.0 125.0 360.0

248 2882 1843.0 123.0 480.0

590 3000 3416.0 56.0 180.0

246 9703 0.0 112.0 360.0

388 2333 2417.0 136.0 360.0

Credit\_History Property\_Area

15 1.0 Urban

248 1.0 Semiurban

590 1.0 Semiurban

246 1.0 Urban

388 1.0 Urban

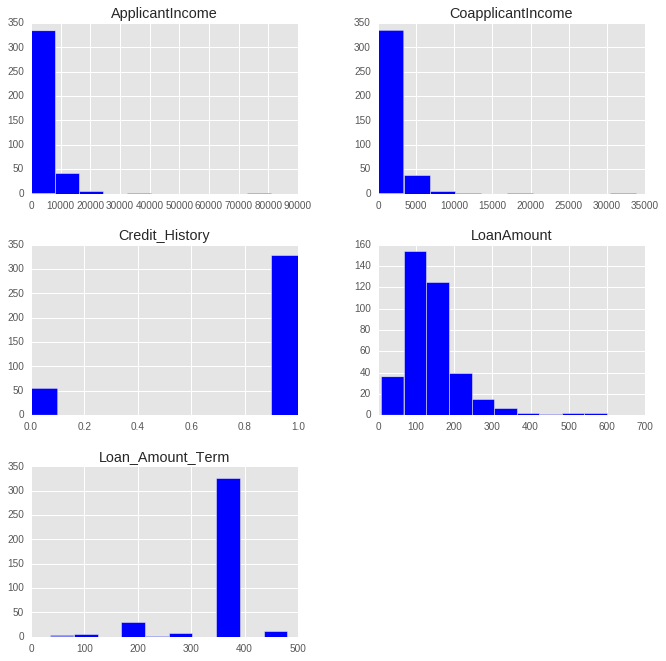
## Feature Scaling

Feature scaling is the method to limit the range of variables so that they can be compared on common grounds. It is performed on continuous variables. Lets plot the distribution of all the continuous variables in  the data set.

>> import matplotlib.pyplot as plt

>> X\_train[X\_train.dtypes[(X\_train.dtypes=="float64")|(X\_train.dtypes=="int64")]

.index.values].hist(figsize=[11,11])

[](https://www.analyticsvidhya.com/wp-content/uploads/2016/06/index-4.png)

After understanding these plots, we infer that ApplicantIncome and CoapplicantIncome are in similar range (0-50000$) where as LoanAmount is in thousands and it ranges from 0 to 600$. The story for Loan\_Amount\_Term is completely different from other variables because its unit is months as opposed to other variables where the unit is dollars.

If we try to apply distance based methods such as kNN on these features, feature with the largest range will dominate the outcome results and we’ll obtain less accurate predictions. We can overcome this trouble using feature scaling. Let’s do it practically.

**Resources :**Check out this article on [kNN](https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/" \t "_blank) for better understanding.

Lets try out kNN on our data set to see how well it will perform.

# Initializing and Fitting a k-NN model

>> from sklearn.neighbors import KNeighborsClassifier

>> knn=KNeighborsClassifier(n\_neighbors=5)

>> knn.fit(X\_train[['ApplicantIncome', 'CoapplicantIncome','LoanAmount',

'Loan\_Amount\_Term', 'Credit\_History']],Y\_train)

# Checking the performance of our model on the testing data set

>> from sklearn.metrics import accuracy\_score

>> accuracy\_score(Y\_test,knn.predict(X\_test[['ApplicantIncome', 'CoapplicantIncome',

'LoanAmount', 'Loan\_Amount\_Term', 'Credit\_History']]))

**Out :** 0.61458333333333337

We got around 61% of correct prediction which is not bad but in real world practices will this be enough ? Can we deploy this model in real world problem? To answer this question lets take a look at distribution of Loan\_Status in train data set.

>> Y\_train.Target.value\_counts()/Y\_train.Target.count()

Out : Y 0.705729

     N 0.294271

Name: Loan\_Status, dtype: float64

There are 70% of approved loans, since there are more number of approved loans we will generate a prediction where all the loans are approved and lets go ahead and check the accuracy of our prediction

>> Y\_test.Target.value\_counts()/Y\_test.Target.count()

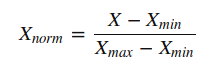
**Out :**  Y 0.635417

      N 0.364583

Name: Loan\_Status, dtype: float64

Wow !! we got an accuracy of 63% just by guessing, What is the meaning of this, getting better accuracy than our prediction model ?

This might be happening because of some insignificant variable with larger range will be dominating the objective function. We can remove this problem by scaling down all the features to a same range. sklearn provides a tool MinMaxScaler that will scale down all the features between 0 and 1. Mathematical formula for MinMaxScaler is.

[](https://www.analyticsvidhya.com/wp-content/uploads/2016/06/Screenshot-from-2016-06-29-14-16-29.png)

Lets try this tool on our problem.

# Importing MinMaxScaler and initializing it

>> from sklearn.preprocessing import MinMaxScaler

>> min\_max=MinMaxScaler()

# Scaling down both train and test data set

>> X\_train\_minmax=min\_max.fit\_transform(X\_train[['ApplicantIncome', 'CoapplicantIncome',

'LoanAmount', 'Loan\_Amount\_Term', 'Credit\_History']])

>> X\_test\_minmax=min\_max.fit\_transform(X\_test[['ApplicantIncome', 'CoapplicantIncome',

'LoanAmount', 'Loan\_Amount\_Term', 'Credit\_History']])

Now, that we are done with scaling, lets apply kNN on our scaled data and check its accuracy.

# Fitting k-NN on our scaled data set

>> knn=KNeighborsClassifier(n\_neighbors=5)

>> knn.fit(X\_train\_minmax,Y\_train)

# Checking the model's accuracy

>> accuracy\_score(Y\_test,knn.predict(X\_test\_minmax))

**Out :** 0.75

Great !! Our accuracy has increased from 61% to 75%. This means that some of the features with larger range were dominating the prediction outcome in the domain of distance based methods(kNN).

It should be kept in mind while performing distance based methods we must attempt to scale the data, so that the feature with lesser significance might not end up dominating the objective function due to its larger range. In addition, features having different unit should also be scaled thus providing each feature equal initial weightage and at the end we will have a better prediction model.

### Exercise 1

Try to do the same exercise with a logistic regression model(parameters : penalty=’l2′,C=0.01) and provide your accuracy before and after scaling in the comment section.

## Feature Standardization

Before jumping to this section I suggest you to complete Exercise 1.

In the previous section, we worked on the Loan\_Prediction data set and fitted a kNN learner on the data set. After scaling down the data, we have got an accuracy of 75% which is very considerably good. I tried the same exercise on Logistic Regression and I got the following result :

**Before Scaling :**61%

**After Scaling :**63%

The accuracy we got after scaling is close to the prediction which we made by guessing, which is not a very impressive achievement. So, what is happening here? Why hasn’t the accuracy increased by a satisfactory amount as it increased in kNN?

**Resources :**Go through this article on [Logistic Regression](https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/) for better understanding.

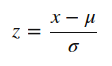
Here is the answer:

In logistic regression, each feature is assigned a weight or coefficient (Wi). If there is a feature with relatively large range and it is insignificant in the objective function then logistic regression will itself assign a very low value to its co-efficient, thus neutralizing the dominant effect of that particular feature, whereas distance based method such as kNN does not have this inbuilt strategy, thus it requires scaling.

Aren’t we forgetting something ? Our logistic model is still predicting with an accuracy almost closer to a guess.

Now, I’ll be introducing a new concept here called standardization. Many machine learning algorithms in sklearn requires standardized data which means having zero mean and unit variance.

Standardization (or Z-score normalization) is the process where the features are rescaled so that they’ll have the properties of a standard normal distribution with μ=0 and σ=1, where μ is the mean (average) and σ is the standard deviation from the mean. Standard scores (also called z scores) of the samples are calculated as follows :

[](https://www.analyticsvidhya.com/wp-content/uploads/2016/06/Screenshot-from-2016-06-29-14-06-42.png)

Elements such as l1 ,l2 regularizer in linear models (logistic comes under this category) and RBF kernel in SVM in objective function of learners assumes that all the features are centered around zero and have variance in the same order.

Features having larger order of variance would dominate on the objective function as it happened in the previous section with the feature having large range. As we saw in the Exercise 1 that without any preprocessing on the data the accuracy was 61%, lets standardize our data apply logistic regression on that. Sklearn provides scale to standardize the data.

# Standardizing the train and test data

>> from sklearn.preprocessing import scale

>> X\_train\_scale=scale(X\_train[['ApplicantIncome', 'CoapplicantIncome',

'LoanAmount', 'Loan\_Amount\_Term', 'Credit\_History']])

>> X\_test\_scale=scale(X\_test[['ApplicantIncome', 'CoapplicantIncome',

'LoanAmount', 'Loan\_Amount\_Term', 'Credit\_History']])

# Fitting logistic regression on our standardized data set

>> from sklearn.linear\_model import LogisticRegression

>> log=LogisticRegression(penalty='l2',C=.01)

>> log.fit(X\_train\_scale,Y\_train)

# Checking the model's accuracy

>> accuracy\_score(Y\_test,log.predict(X\_test\_scale))

**Out :** 0.75

We again reached to our maximum score that was attained using kNN after scaling. This means standardizing the data when using a estimator having l1 or l2 regularization helps us to increase the accuracy of the prediction model. Other learners like kNN with euclidean distance measure, k-means, SVM, perceptron, neural networks, linear discriminant analysis, principal component analysis may perform better with standardized data.

Though, I suggest you to understand your data and what kind of algorithm you are going to apply on it; over the time you will be able to judge weather to standardize your data or not.

**Note :**Choosing between scaling and standardizing is a confusing choice, you have to dive deeper in your data and learner that you are going to use to reach the decision. For starters, you can try both the methods and check cross validation score for making a choice.

**Resources :**Go through this article on [cross validation](https://www.analyticsvidhya.com/blog/2015/11/improve-model-performance-cross-validation-in-python-r/) for better understanding.

### Exercise 2

Try to do the same exercise with SVM model and provide your accuracy before and after standardization in the comment section.

**Resources :**Go through this article on [support vector machines](https://www.analyticsvidhya.com/blog/2015/10/understaing-support-vector-machine-example-code/) for better understanding.

## Label Encoding

In previous sections, we did the pre-processing for continuous numeric features. But, our data set has other features too such as Gender, Married, Dependents, Self\_Employed and Education. All these categorical features have string values. For example, Gender has two levels either Male or Female. Lets feed the features in our logistic regression model.

# Fitting a logistic regression model on whole data

>> log=LogisticRegression(penalty='l2',C=.01)

>> log.fit(X\_train,Y\_train)

# Checking the model's accuracy

>> accuracy\_score(Y\_test,log.predict(X\_test))

Out : ValueError: could not convert string to float: Semiurban

We got an error saying that it cannot convert string to float. So, what’s actually happening here is learners like logistic regression, distance based methods such as kNN, support vector machines, tree based methods etc. in sklearn needs numeric arrays. Features having string values cannot be handled by these learners.

Sklearn provides a very efficient tool for encoding the levels of a categorical features into numeric values. LabelEncoder encode labels with value between 0 and n\_classes-1.

Lets encode all the categorical features.

# Importing LabelEncoder and initializing it

>> from sklearn.preprocessing import LabelEncoder

>> le=LabelEncoder()

# Iterating over all the common columns in train and test

>> for col in X\_test.columns.values:

# Encoding only categorical variables

if X\_test[col].dtypes=='object':

# Using whole data to form an exhaustive list of levels

data=X\_train[col].append(X\_test[col])

le.fit(data.values)

X\_train[col]=le.transform(X\_train[col])

X\_test[col]=le.transform(X\_test[col])

All our categorical features are encoded. You can look at your updated data set using X\_train.head(). We are going to take a look at Gender frequency distribution before and after the encoding.

**Before :** Male 318

Female 66

Name: Gender, dtype: int64

**After :** 1 318

0 66

Name: Gender, dtype: int64

Now that we are done with label encoding, lets now run a logistic regression model on the data set with both categorical and continuous features.

# Standardizing the features

>> X\_train\_scale=scale(X\_train)

>> X\_test\_scale=scale(X\_test)

# Fitting the logistic regression model

>> log=LogisticRegression(penalty='l2',C=.01)

>> log.fit(X\_train\_scale,Y\_train)

# Checking the models accuracy

>> accuracy\_score(Y\_test,log.predict(X\_test\_scale))

Out : 0.75

Its working now. But, the accuracy is still the same as we got with logistic regression after standardization from numeric features. This means categorical features we added are not very significant in our objective function.

### Exercise 3

Try out decision tree classifier with all the features as independent variables and comment your accuracy.

**Resources :**Go through this article on [decision trees](https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/) for better understanding.

## One-Hot Encoding

One-Hot Encoding transforms each categorical feature with n possible values into n binary features, with only one active.

Most of the ML algorithms either learn a single weight for each feature or it computes distance between the samples. Algorithms like linear models (such as logistic regression) belongs to the first category.

Lets take a look at an example from loan\_prediction data set. Feature Dependents have 4 possible values 0,1,2 and 3+ which are then encoded without loss of generality to 0,1,2 and 3.

We, then have a weight “W” assigned for this feature in a linear classifier,which will make a decision based on the constraints W\*Dependents + K > 0 or eqivalently  W\*Dependents < K.

Let f(w)= W\*Dependents

Possible values that can be attained by the equation are 0, W, 2W and 3W. A problem with this equation is that the weight “W” cannot make decision based on four choices. It can reach to a decision in following ways:

* All leads to the same decision (all of them <K or vice versa)
* 3:1 division of the levels (Decision boundary at f(w)>2W)
* 2:2 division of the levels (Decision boundary at f(w)>W)

Here we can see that we are loosing many different possible decisions such as the case where “0” and “2W” should be given same label and “3W” and “W” are odd one out.

This problem can be solved by One-Hot-Encoding as it effectively changes the dimensionality of the feature “Dependents” from one to four, thus every value in the feature “Dependents” will have their own weights. Updated equation for the decison would be f'(w) < K.

where,  f'(w) = W1\*D\_0 + W2\*D\_1 + W3\*D\_2 + W4\*D\_3  
All four new variable has boolean values (0 or 1).

The same thing happens with distance based methods such as kNN. Without encoding, distance between “0” and “1” values of Dependents is 1 whereas distance between “0” and “3+” will be 3, which is not desirable as both the distances should be similar. After encoding, the values will be new features (sequence of columns is 0,1,2,3+) : [1,0,0,0] and [0,0,0,1] (initially we were finding distance between “0” and “3+”), now the distance would be √2.

For tree based methods, same situation (more than two values in a feature) might effect the outcome to extent but if methods like random forests are deep enough, it can handle the categorical variables without one-hot encoding.

Now, lets take look at the implementation of one-hot encoding with various algorithms.

Lets create a logistic regression model for classification without one-hot encoding.

# We are using scaled variable as we saw in previous section that

# scaling will effect the algo with l1 or l2 reguralizer

>> X\_train\_scale=scale(X\_train)

>> X\_test\_scale=scale(X\_test)

# Fitting a logistic regression model

>> log=LogisticRegression(penalty='l2',C=1)

>> log.fit(X\_train\_scale,Y\_train)

# Checking the model's accuracy

>> accuracy\_score(Y\_test,log.predict(X\_test\_scale))

**Out :** 0.73958333333333337

Now we are going to encode the data.

>> from sklearn.preprocessing import OneHotEncoder

>> enc=OneHotEncoder(sparse=False)

>> X\_train\_1=X\_train

>> X\_test\_1=X\_test

>> columns=['Gender', 'Married', 'Dependents', 'Education','Self\_Employed',

'Credit\_History', 'Property\_Area']

>> for col in columns:

# creating an exhaustive list of all possible categorical values

data=X\_train[[col]].append(X\_test[[col]])

enc.fit(data)

# Fitting One Hot Encoding on train data

temp = enc.transform(X\_train[[col]])

# Changing the encoded features into a data frame with new column names

temp=pd.DataFrame(temp,columns=[(col+"\_"+str(i)) for i in data[col]

.value\_counts().index])

# In side by side concatenation index values should be same

# Setting the index values similar to the X\_train data frame

temp=temp.set\_index(X\_train.index.values)

# adding the new One Hot Encoded varibales to the train data frame

X\_train\_1=pd.concat([X\_train\_1,temp],axis=1)

# fitting One Hot Encoding on test data

temp = enc.transform(X\_test[[col]])

# changing it into data frame and adding column names

temp=pd.DataFrame(temp,columns=[(col+"\_"+str(i)) for i in data[col]

.value\_counts().index])

# Setting the index for proper concatenation

temp=temp.set\_index(X\_test.index.values)

# adding the new One Hot Encoded varibales to test data frame

X\_test\_1=pd.concat([X\_test\_1,temp],axis=1)

Now, lets apply logistic regression model on one-hot encoded data.

# Standardizing the data set

>> X\_train\_scale=scale(X\_train\_1)

>> X\_test\_scale=scale(X\_test\_1)

# Fitting a logistic regression model

>> log=LogisticRegression(penalty='l2',C=1)

>> log.fit(X\_train\_scale,Y\_train)

# Checking the model's accuracy

>> accuracy\_score(Y\_test,log.predict(X\_test\_scale))

**Out :** 0.75

Here, again we got the maximum accuracy as 0.75 that we have gotten so far. In this case, logistic regression regularization(C) parameter 1 where as earlier we used C=0.01.

## End Notes

The aim of this article is to familiarize you with the basic data pre-processing techniques and have a deeper understanding of the situations of where to apply those techniques.

These methods work because of the underlying assumptions of the algorithms. This is by no means an exhaustive list of the methods. I’d encourage you to experiment with these methods since they can be heavily modified according to the problem at hand.

I plan to provide more advance techniques of data pre-processing such as [pipeline](http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html) and noise reduction in my next post, so stay tuned to dive deeper into the data pre-processing.