#AV\_Data pre-processing

#-----------------------------

#https://www.analyticsvidhya.com/blog/2016/07/practical-guide-data-preprocessing-python-scikit-learn/

#Algorithm like XGBoost, specifically requires dummy encoded data while algorithm like decision tree doesn’t seem to care at all (sometimes)!

# 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())

#----------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

#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.

#https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

#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])

#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.

#------------------------

# 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']]))

#---------------------------------

#-----------scaling variable

# 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']])

# 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))

#RESULT::hence accuracy improved 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.

# -------------Standardizing the train and test data

# 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.

#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.

#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

# 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.

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))

#NOTES::

#Choosing between scaling and standardizing is a confusing choice, you can try both the methods and check cross validation score for making a choice.

#https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/

#---------------Label Encoding

#In previous sections, we did the pre-processing for continuous numeric features.

#our data set has other features too such as Gender, Married, Dependents, Self\_Employed and Education.

# 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])

#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))

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

#https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/

#------------One-Hot Encoding

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

#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).

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)

#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))

#more advance

#http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html