**Problem Statement:**

A data science firm wants to have a model that helps to predict whether a gene is from a plant or an animal from the relevant data given from biological experts. A dataset had been created taking input from various sites of the internet. The various attributes of the data set are colour, symbol, type of gene etc. The objective is to predict the class attribute i.e. whether the undiscovered gene type belongs to plant or animal.

The various data mining activities to achieve the objective would be done in Python and its suitable libraries. The data set would be divided into training and test data. Training data would be used to train the model and test data to test the same. Improvement ideas to enhance the model also needs to be provided.

**Perform exploratory data analysis using the statistical techniques and box plot as applicable:**

Boxplot, Histogram and Scatter Matrix plot has been made using Python libraries for understanding of the data set. Below is the python code snippet and output :

Code Snippet :

#boxplot

df\_gene.plot(kind='box', subplots=True, layout=(3,3), sharex=False, sharey=False, figsize=(10,10))

print('\n\t Boxplot Analysis\n')

pyplot.show()

#histogram

df\_gene.hist(figsize=(10,10))

print('\t Histogram Analysis\n')

pyplot.show()

#scatter matrix

scatter\_matrix(df\_gene,figsize=(16,20))

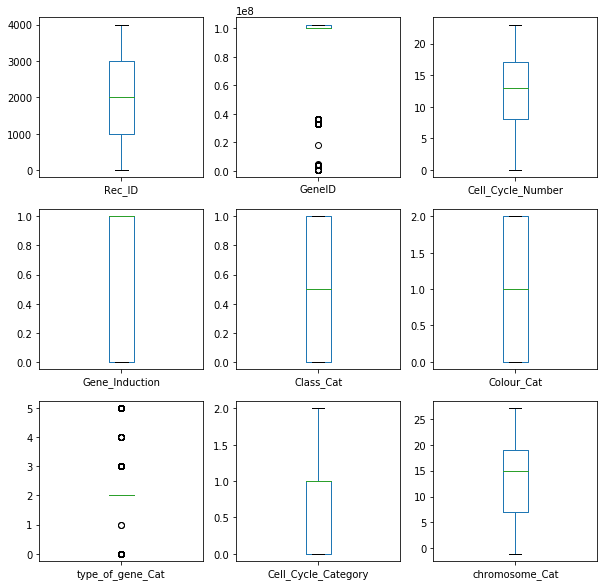
print('\t Scatter Matrix Analysis\n')

pyplot.show()

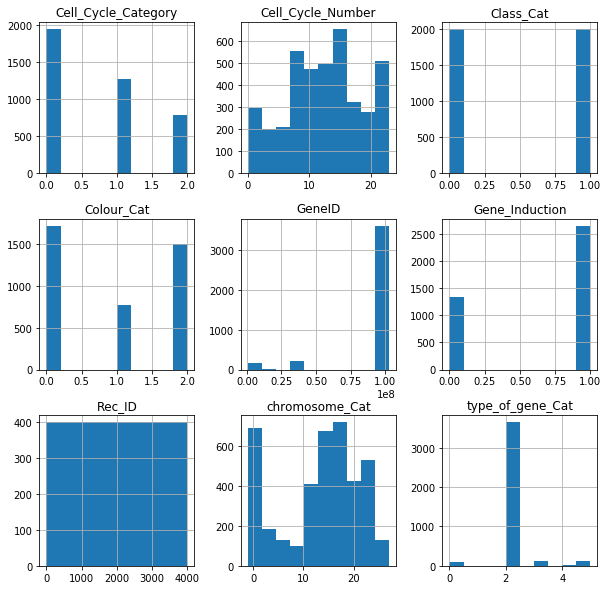
Output:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Analysis Of The Data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

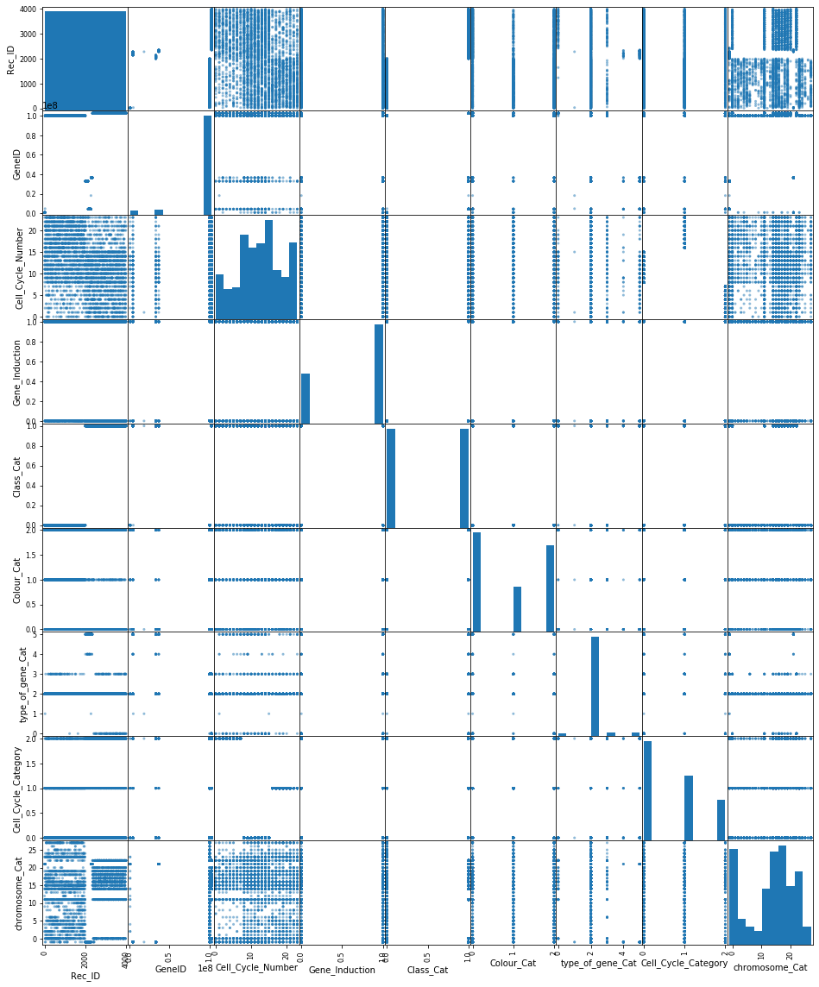
Boxplot Analysis



Histogram Analysis



Scatter Matrix Analysis



**Preprocess the Data as the data from internet source may be noisy**

The received data has been pre processed to handle the missing data.

Code Snippet :

#Replacing the '-' values with NaN

df\_gene=df\_gene.replace('-', np.nan)

#.astype("object")

#Getting Percentages of the Missing values in the respective columns

total = df\_gene.isnull().sum().sort\_values(ascending=False)

percent\_1 = df\_gene.isnull().sum()/df\_gene.isnull().count()\*100

percent\_2 = (round(percent\_1, 1)).sort\_values(ascending=False)

missing\_data = pd.concat([total, percent\_2], axis=1, keys=['Total', '%'])

missing\_data.head(10)

**Apply normalization as applicable as some of the attributes may have larger values**

It is hard to know whether rescaling the data will improve the performance of the algorithms before we apply them. If often can, but not always. In our case the normalization has degraded the accuracy of the model. Hence we did not do the normalization and commented the code.

Code Snippet :

from sklearn import preprocessing

#Defining the X and Y variables

X=df\_gene[feature\_col\_names]

Y=df\_gene.Class\_Cat

#Normalization

normalized\_X = preprocessing.normalize(X)

Score without normalization:

Accuracy Score :

0.8575

Score with normalization:

Accuracy Score :

0.7475

**Select Appropriate Training/Test Data and reduce the dimensions/attributes to simplify the solution:**

We have selected 70% training data and 30% test data.

**Train and Test the model (Predictions and reporting)**

We have used following code to train and test the model

# Split dataset into training set and test set

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=1) # 70% training and 30% test

**Evaluate the model performance using F-Score and other DM Measure**

Accuracy score, Confusion Matrix and Classification report showing precision, recall, f1-score and support had been printed to evaluate the model performance.

Code Snippet:

# Make predictions on validation dataset

model = SVC(gamma='auto')

model.fit(X\_train, Y\_train)

predictions = model.predict(X\_test)

# Evaluate predictions

title\_str='\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Prediction Results with SVC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n'

print(title\_str.title())

print('Accuracy Score :\n', accuracy\_score(Y\_test, predictions),'\n')

print('Confusion Matrix:\n', confusion\_matrix(Y\_test, predictions),'\n')

print('Classification Report:\n',classification\_report(Y\_test, predictions),'\n')

Output:

Accuracy Score :

0.8575

Confusion Matrix:

[[511 105]

[ 66 518]]

Classification Report:

precision recall f1-score support

0 0.89 0.83 0.86 616

1 0.83 0.89 0.86 584

accuracy 0.86 1200

macro avg 0.86 0.86 0.86 1200

weighted avg 0.86 0.86 0.86 1200

**Suggest ways of improving the model:**

We have tried various classifier models and evaluated their performances. Based on the data analysis we have chosen the SVC model which has given the best result. For further improving the result, following could have been done :

* Having a bigger data set
* Having more columns which numerically relates the category and class attributes
* Better domain knowledge for feature selection

Code Snippet for Algorithm evaluation:

# Check Algorithms

models = []

models.append(('LogisticRegression', LogisticRegression(solver='liblinear', multi\_class='ovr')))

models.append(('LinearDiscriminantAnalysis', LinearDiscriminantAnalysis()))

models.append(('KNeighborsClassifier', KNeighborsClassifier()))

models.append(('DecisionTreeClassifier', DecisionTreeClassifier()))

models.append(('GaussianNB', GaussianNB()))

models.append(('SVC', SVC(gamma='auto')))

models.append(('RandomForestClassifier',RandomForestClassifier()))

# evaluate each model in turn

results = []

names = []

for name, model in models:

kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)

cv\_results = cross\_val\_score(model, X\_train, Y\_train, cv=kfold, scoring='accuracy')

results.append(cv\_results)

names.append(name)

print('Name: %s, Mean: %f, Standard Deviation :%f' % (name, cv\_results.mean(), cv\_results.std()))

pyplot.figure(figsize=(18,10))

pyplot.boxplot(results, labels=names)

pyplot.title('Algorithm Comparison By Boxplot \n')

pyplot.show()

Output :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Analysis Of The Classifier Algorithms \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Name: LogisticRegression, Mean: 0.781786, Standard Deviation :0.024715

Name: LinearDiscriminantAnalysis, Mean: 0.778929, Standard Deviation :0.027593

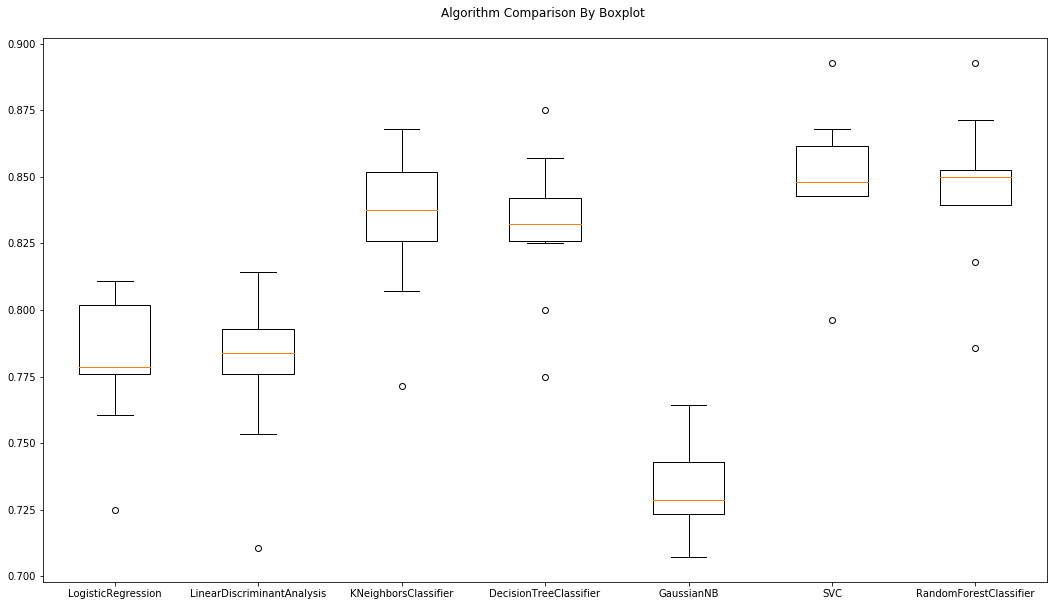
Name: KNeighborsClassifier, Mean: 0.833929, Standard Deviation :0.027258

Name: DecisionTreeClassifier, Mean: 0.830714, Standard Deviation :0.026400

Name: GaussianNB, Mean: 0.732857, Standard Deviation :0.015633

Name: SVC, Mean: 0.850000, Standard Deviation :0.023255

Name: RandomForestClassifier, Mean: 0.845000, Standard Deviation :0.027302



**Assumptions made :**

1. The dataset of 4000 records gives a good overview
2. 70% training data and 30% test data is a good mix
3. The class attributes are related to the category attribute (assumption from domain perspective)

**Appendix A**

Complete Code :

import pandas as pd

import requests

import io

import numpy as np

from pandas.plotting import scatter\_matrix

from matplotlib import pyplot

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import StratifiedKFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

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

#from sklearn import preprocessing

# Downloading the csv file from your GitHub account

url = "https://raw.githubusercontent.com/ravikanthranjith/BITS-DM-ASSIGNMENT/master/GeneData.csv" # Make sure the url is the raw version of the file on GitHub

download = requests.get(url).content

# Reading the downloaded content and turning it into a pandas dataframe

df\_gene = pd.read\_csv(io.StringIO(download.decode('utf-8')))

#Replacing the '-' values with NaN

df\_gene=df\_gene.replace('-', np.nan)

#.astype("object")

#Getting Percentages of the Missing values in the respective columns

total = df\_gene.isnull().sum().sort\_values(ascending=False)

percent\_1 = df\_gene.isnull().sum()/df\_gene.isnull().count()\*100

percent\_2 = (round(percent\_1, 1)).sort\_values(ascending=False)

missing\_data = pd.concat([total, percent\_2], axis=1, keys=['Total', '%'])

missing\_data.head(10)

#Converting all the Category columns

df\_gene["Class"] = df\_gene["Class"].astype('category') # This is the Prediction class

df\_gene["Colour"] = df\_gene["Colour"].astype('category')#Colour

df\_gene["type\_of\_gene"] = df\_gene["type\_of\_gene"].astype('category')#Type of Gene

df\_gene["Cell\_Cycle\_Cat"] = df\_gene["Cell\_Cycle\_Cat"].astype('category')#Cell\_Cycle\_Cat

df\_gene["Cell\_Cycle\_Number"] = df\_gene["Cell\_Cycle\_Number"].astype('category')#Cell\_Cycle\_Number

df\_gene["chromosome"] = df\_gene["chromosome"].astype('category')#chromosome

#Creation of "\_Cat" Columns based on the columns converted as categories

df\_gene["Class\_Cat"] = df\_gene["Class"].cat.codes

df\_gene["Colour\_Cat"] = df\_gene["Colour"].cat.codes

df\_gene["type\_of\_gene\_Cat"] = df\_gene["type\_of\_gene"].cat.codes

df\_gene["Cell\_Cycle\_Category"] = df\_gene["Cell\_Cycle\_Cat"].cat.codes#cell\_Cycle\_Cat

df\_gene["Cell\_Cycle\_Number"] = df\_gene["Cell\_Cycle\_Number"].cat.codes#cell\_Cycle\_Number

df\_gene["chromosome\_Cat"] = df\_gene["chromosome"].cat.codes#chromosome

#Analysis of data

title\_str='\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Analysis of the data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

print(title\_str.title())

#boxplot

df\_gene.plot(kind='box', subplots=True, layout=(3,3), sharex=False, sharey=False, figsize=(10,10))

print('\n\t Boxplot Analysis\n')

pyplot.show()

#histogram

df\_gene.hist(figsize=(10,10))

print('\t Histogram Analysis\n')

pyplot.show()

#scatter matrix

scatter\_matrix(df\_gene,figsize=(16,20))

print('\t Scatter Matrix Analysis\n')

pyplot.show()

#Considering only the required columns Names

feature\_col\_names = ['Cell\_Cycle\_Category','Cell\_Cycle\_Number','chromosome\_Cat','Colour\_Cat','Gene\_Induction','type\_of\_gene\_Cat']

#Defining the X and Y variables

X=df\_gene[feature\_col\_names]

Y=df\_gene.Class\_Cat

#Normalization

#normalized\_X = preprocessing.normalize(X)

# Split dataset into training set and test set

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=1) # 70% training and 30% test

# Compare Algorithms

title\_str='\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Analysis of the classifier algorithms \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n'

print(title\_str.title())

# Check Algorithms

models = []

models.append(('LogisticRegression', LogisticRegression(solver='liblinear', multi\_class='ovr')))

models.append(('LinearDiscriminantAnalysis', LinearDiscriminantAnalysis()))

models.append(('KNeighborsClassifier', KNeighborsClassifier()))

models.append(('DecisionTreeClassifier', DecisionTreeClassifier()))

models.append(('GaussianNB', GaussianNB()))

models.append(('SVC', SVC(gamma='auto')))

models.append(('RandomForestClassifier',RandomForestClassifier()))

# evaluate each model in turn

results = []

names = []

for name, model in models:

kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)

cv\_results = cross\_val\_score(model, X\_train, Y\_train, cv=kfold, scoring='accuracy')

results.append(cv\_results)

names.append(name)

print('Name: %s, Mean: %f, Standard Deviation :%f' % (name, cv\_results.mean(), cv\_results.std()))

pyplot.figure(figsize=(18,10))

pyplot.boxplot(results, labels=names)

pyplot.title('Algorithm Comparison By Boxplot \n')

pyplot.show()

# Make predictions on validation dataset

model = SVC(gamma='auto')

model.fit(X\_train, Y\_train)

predictions = model.predict(X\_test)

# Evaluate predictions

title\_str='\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Prediction Results with SVC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n'

print(title\_str.title())

print('Accuracy Score :\n', accuracy\_score(Y\_test, predictions),'\n')

print('Confusion Matrix:\n', confusion\_matrix(Y\_test, predictions),'\n')

print('Classification Report:\n',classification\_report(Y\_test, predictions),'\n')

**Appendix B:**

Attached is the dataset used for the model.

