Explanation

This analysis aims to observe which features are most helpful in predicting malignant or benign cancer and to see general trends that may aid us in model selection and hyperparameter selection. The goal is to classify whether breast cancer is benign or malignant. To achieve this we have used machine learning classification methods to fit a function that can predict the discrete class of new input.

Phase 1 — Data Exploration

We will be using Collab-Notebook to work on this dataset. We will first go with importing the necessary libraries and import our dataset to Collab-Notebook:

#importing libraries  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
#Load the data  
df = pd.read\_csv('data.csv')  
df.head()

data

We can find the dimensions of the data set using the panda dataset ‘shape’ attribute.

#Number and rows and columns in the datasets  
df.shape

Shape

**Missing or Null Data points**

We can find any missing or null data points of the data set (if there is any) using the following panda’s function.

#Count the number of empty values in each column  
df.isna().sum()

null

There is a column that has all the null values so we have to drop it.

#Drop the empty column  
df=df.dropna(axis=1)

Phase 2 — Categorical Data

Categorical data are variables that contain label values rather than numeric values. The number of possible values is often limited to a fixed set.

For example, users are typically described by country, gender, age group etc.

We will use Label Encoder to label the categorical data. Label Encoder is the part of SciKit Learn library in Python and used to convert categorical data, or text data, into numbers, which our predictive models can better understand.

from sklearn.preprocessing import LabelEncoder  
labelencoder\_Y = LabelEncoder()  
df.iloc[:,1] = labelencoder\_Y.fit\_transform(df.iloc[:,1].values)

**Splitting the dataset**

The data we use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. We have the test dataset (or subset) in order to test our model’s prediction on this subset.

We will do this using SciKit-Learn library in Python using the train\_test\_split method.

#split the data set into independent (x) and dependent (y) data sets  
X=df.iloc[:2:31].values  
Y=df.iloc[:1].values  
#split the data set  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y,test\_size =0.20, random\_state=0)

**Finding Correlation**

#Get the correlation of the columns  
df.iloc[:,1:32].corr()  
#visualize the correlation  
plt.figure(figsize=(10,10))  
sns.heatmap(df.iloc[:,1:12].corr(),annot=True,fmt='.0%')

Heatmap

Phase 3 — Feature Scaling

Most of the times, your dataset will contain features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Eucledian distance between two data points in their computations. We need to bring all features to the same level of magnitudes. This can be achieved by scaling. This means that you’re transforming your data so that it fits within a specific scale, like 0–100 or 0–1.

We will use StandardScaler method from SciKit-Learn library.

#Feature Scaling  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.fit\_transform(X\_test)

Phase 4 — Model Selection

This is the most exciting phase in Applying Machine Learning to any Dataset. It is also known as Algorithm selection for Predicting the best results.

Usually Data Scientists use different kinds of Machine Learning algorithms to the large data sets. But, at high level all those different algorithms can be classified in two groups : supervised learning and unsupervised learning.

Without wasting much time, I would just give a brief overview of these two types of learnings.

Supervised learning: Supervised learning is a type of system in which both input and desired output data are provided. Input and output data are labeled for classification to provide a learning basis for future data processing. Supervised learning problems can be further grouped into Regression and Classification problems.

A regression problem is when the output variable is a real or continuous value, such as “salary” or “weight”.

A classification problem is when the output variable is a category like filtering emails as “spam” or “not spam”

Unsupervised Learning: Unsupervised learning is the algorithm using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance.

In our dataset we have the outcome variable or Dependent variable i.e Y having only two sets of values, either M (Malign) or B(Benign). So we will use the Classification algorithm of supervised learning.

We have different types of classification algorithms in Machine Learning:-

Logistic Regression

Decision Tree

Random Forest Classifier

Let’s start applying the algorithms :

We will use the sklearn library to import all the methods of classification algorithms.

# Function for Models  
def models(X\_train , Y\_train):  
 #Logistic Regression  
 from sklearn.linear\_model import LogisticRegression  
 log = LogisticRegression(random\_state=0)  
 log.fit(X\_train, Y\_train)  
   
 #Decision Tree  
 from sklearn.tree import DecisionTreeClassifier  
 tree = DecisionTreeClassifier(criterion='entropy', random\_state=0)  
 tree.fit(X\_train, Y\_train)  
   
 #Random Forest Classifier  
 from sklearn.ensemble import RandomForestClassifier  
 forest = RandomForestClassifier(n\_estimators=10, criterion='entropy',random\_state=0)  
 forest.fit(X\_train,Y\_train)  
   
 print('[0]Logistic Regression trainning accuracy:',log.score(X\_train,Y\_train))  
 print('[1]Decision trainning accuracy:',tree.score(X\_train,Y\_train))  
 print('[2]Random Forest classifier trainning accuracy:',forest.score(X\_train,Y\_train))  
   
 return log,tree,forest  
  
model = models(X\_train,Y\_train)

To check the accuracy we need to import confusion\_matrix method of metrics class. The confusion matrix is a way of tabulating the number of mis-classifications, i.e., the number of predicted classes which ended up in a wrong classification bin based on the true classes.

To check the correct prediction we have to check confusion matrix object and add the predicted results diagonally which will be number of correct prediction and then divide by total number of predictions.

#testing the model on the test data (confusion matrix)  
from sklearn.metrics import confusion\_matrix  
for i in range(len(model)):  
 print('Model ', i)  
 cm = confusion\_matrix(Y\_test,model[i].predict(X\_test))  
 print(cm)  
 TP = cm[0][0]  
 TN = cm[1][1]  
 FP = cm[0][1]  
 FN = cm[1][0]  
 Accuracy = (TP + TN)/(TP+TN+FN+FP)  
 print('Accuracy of model ', Accuracy)