Breast Cancer Prediction

A Project Report

In partial fulfilment of the requirements for the award of the degree

Bachelor of Technology

in

Electronics and Communication Engineering

Under the guidance of

Joyjit Guha

By

Arin Nath, Kartik Midya, Pronay Chandra Mridha,

Rupam Bit, Sorbajit Goswami

Kalyani Government Engineering College

In association with

ARDENT COMPUTECH PVT. LTD.

(ISO9001:2015)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

Title of the Project: Breast Cancer Prediction

Project Members: Arin Nath, Kartik Midya, Sorbajit Goswami, Rupam Bit, Pronay Chandra Mridha

Name of the guide: Mr. Joyjit Guha

Address: Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified) SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal, 70009

Version	Authors	Description of version	Date Completed
Final	Arin Nath Kartik Midya Rupam Bit Sorbajit Goswami Pronay Chandra Mridha	Project Report	29 sept 2022

Declaration:

We hereby declare that the project work being presented in the project proposal entitled “Breast Cancer Prediction” in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology at Ardent Computech PVT. LTD, Saltlake, Kolkata, West Bengal, is an authentic work carried out under the guidance of Mr. Joyjit Guha. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date: 29/09/2022

Name of the Students: Arin Nath

Kartik Midya

Sorbajit Goswami

Rupam Bit

Pronay Chandra Mridha

Certificate:

This is to certify that this proposal of minor project entitled "Breast Cancer Prediction" is a record of Bonafede work, carried out by Arin Nath under my guidance at Ardent Computech PVT. LTD. In my opinion, the report in its present form is in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology and as per regulations of the Ardent®. To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

Guide/Supervisor

Mr. Joyjit Guha

Project Engineer

Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

Acknowledgement:

Success of any project depends largely on the encouragement and guidelines of many others. I take this sincere opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project work.

I would like to show our greatest appreciation to Mr. JOYJIT GUHA, Project Engineer at Ardent, Kolkata. I always feel motivated and encouraged every time by his valuable advice and constant inspiration; without his encouragement and guidance this project would not have materialized.

Words are inadequate in offering our thanks to the other trainees, project assistants and other members at Ardent Computech Pvt. Ltd. for their encouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project, was vital for the success of this project.

Contents:

Overview:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and has fewer syntactical constructions than other languages.

Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to Perl and PHP.

Python is Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs. Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language: Python is a great language for the beginnerlevel programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Environment Setup:

- 64-Bit OS
- Install Python 3
- Setup virtual environment
- Install Package

Variable Types:

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Assigning Values to Variables:

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

```
counter=10                  #An integer assignment
weight=10.60                # A floating point
name="Ardent"               # A string
```

Multiple Assignment:

Python allows you to assign a single value to several variables simultaneously. For example –

```
a = b = c = 1

a,b,c = 1,2,"hello"
```

Standard Data Types:

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has five standard data types –

String

List

Tuple

Dictionary

Number

Data Type Conversion:

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function. There are

several built-in functions to perform conversion from one data type to another.

Sr.No.	Function & Description
1	**int(x [,base])** Converts x to an integer. base specifies the base if x is a string
2	**long(x [,base])** Converts x to a long integer. base specifies the base if x is a string.
3	**float(x)** Converts x to a floating-point number.
4	**complex(real [,imag])** Creates a complex number.
5	**str(x)** Converts object x to a string representation.
6	**repr(x)** Converts object x to an expression string.
7	**eval(str)** Evaluates a string and returns an object.
8	**tuple(s)** Converts s to a tuple.
9	**list(s)** Converts s to a list.

Functions:

Defining a Function:

- def functionname(parameters):

"function_docstring"

function_suite

return [expression]

Pass by reference vs Pass by value:

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For Example-

```
# Function definition is here
 def changeme(mylist):
      "This changes a passed list into this function"
      mylist.append([1,2,3,4]);
      print"Values inside the function: ",mylist
      return
# Now you can call changeme function


       mylist=[10,20,30];
       changeme(mylist);
       print"Values outside the function: ",mylist
```

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result –

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
 Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope. For Example-

```
total=0;                    # This is global variable
```

```
# Function definition is here
def sum( arg1, arg2 ):
 # Add both the parameters and return them."
total= arg1 + arg2;                    # Here total is local variable.
print"Inside the function local total : ", total
return total;

# Now you can call sum function
sum(10,20);
print"Outside the function global total : ", total
```

When the above code is executed, it produces the following result –

```
Inside the function local total : 30
Outside the function global total : 0
```

Packages:

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub- subpackages, and so on.

Consider a file Pots.py available in Phone directory. This file has following line of source code –

```
def Pots():
        print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above –

Phone/Isdn.py file having function Isdn()

Phone/G3.py file having function G3()

Now, create one more file __init__.py in Phone directory –

Phone/__init__.py

To make all of your functions available when you've imported Phone, you need to put explicit import statements in __init__.py as follows –

from Pots import Pots

from Isdn import Isdn

from G3 import

Numpy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

Scipy:

Scientific computing tools for Python

SciPy refers to several related but distinct entities:

- The SciPy ecosystem, a collection of open source software for scientific computing in Python.
- The community of people who use and develop this stack.
- Several conferences dedicated to scientific computing in Python - SciPy, EuroSciPy, and SciPy.in.
- The SciPy library, one component of the SciPy stack, providing many numerical routines.

The SciPy ecosystem

Scientific computing in Python builds upon a small core of packages:

- Python, a general purpose programming language. It is interpreted and dynamically typed and is very well suited

for interactive work and quick prototyping, while being powerful enough to write large applications in.

- NumPy, the fundamental package for numerical computation. It defines the numerical array and matrix types and basic operations on them.
- The SciPy library, a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics, and much more.
- Matplotlib, a mature and popular plotting package that provides publication- quality 2-D plotting, as well as rudimentary 3-D plotting.

On this base, the SciPy ecosystem includes general and specialised tools for data management and computation, productive experimentation, and high- performance computing. Below, we overview some key packages, though there are many more relevant packages.

Data and computation:

- pandas, providing high-performance, easy-to-use data structures.
- SymPy, for symbolic mathematics and computer algebra.
- NetworkX, is a collection of tools for analyzing complex networks.
- scikit-image is a collection of algorithms for image processing.
- scikit-learn is a collection of algorithms and tools for machine learning.
- h5py and PyTables can both access data stored in the HDF5 format.

Productivity and high-performance computing:

- IPython, a rich interactive interface, letting you quickly process data and test ideas.
- The Jupyter notebook provides IPython functionality and more in your web browser, allowing you to document your computation in an easily reproducible form.

- Cython extends Python syntax so that you can conveniently build C extensions, either to speed up critical code or to integrate with C/C++ libraries.
- Dask, Joblib or IPyParallel for distributed processing with a focus on numeric data.

Quality assurance:

- nose, a framework for testing Python code, being phased out in preference for pytest.
- numpydoc, a standard and library for documenting Scientific Python libraries.

Pandas:

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. "Panel data", an econometrics term for multidimensional, structured data sets.

Library features:

¬ Data Frame object for data manipulation with integrated indexing.

¬ Tools for reading and writing data between in-memory data structures and different file formats.

¬ Data alignment and integrated handling of missing data.

¬ Reshaping and pivoting of data sets.

¬ Label-based slicing, fancy indexing, and sub setting of large data sets.

¬ Data structure column insertion and deletion. ¬ Group by engine allowing split-apply-combine operations on data sets. ¬ Data set merging and joining.

¬ Hierarchical axis indexing to work with high-dimensional data in a lower- dimensional data structure.

¬ Time series-functionality: Date range generation.

Python Speech Features:

This library provides common speech features for ASR including MFCCs and filterbank energies. If you are not sure what MFCCs are, and would like to know more have a look at this MFCC

tutorial:http://www.practicalcryptography.com/miscellaneous/machinelearning/guide-mel-frequency-cepstral-coefficients-mfccs/.

You will need numpy and scipy to run these files. The code for this project is available at https://github.com/jameslyons/python_speech_features.
Supported features:

- python_speech_features. mfcc () - Mel Frequency Cepstral Coefficients
- python_speech_features. fbank () - Filterbank Energies
- python_speech_features. logfbank () - Log Filterbank Energies
- python_speech_features.ssc () - Spectral Subband Centroids

To use MFCC features:

```
from python_speech_features import mfccfrom python
_speech_features import logfbankimport scipy.io.wavfile as
wav
 (rate, sig) = wav. read("file.wav")
 mfcc_feat = mfcc(sig,rate)fbank_feat = logfbank(sig,rate)
print(fbank_feat[1:3,:])
```

OS: Miscellaneous operating system interfaces

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module. For

creating temporary files and directories see the tempfile module, and for high-level file and directory handling see the shutil module.

Notes on the availability of these functions:

The design of all built-in operating system dependent modules of Python is such that as long as the same functionality is available, it uses the same interface; for example, the function os.stat(path) returns stat information about path in the same format (which happens to have originated with the POSIX interface).

Extensions peculiar to a particular operating system are also available through the os module, but using them is of course a threat to portability.

All functions accepting path or file names accept both bytes and string objects, and result in an object of the same type, if a path or file name is returned.

On VxWorks, os.fork, os.execv and os.spawn *p* are not supported.

Pickle: Python object serialization

The pickle module implements binary protocols for serializing and de- serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” 1 or “flattening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

Operator: Standard operators as functions The operator module exports a set of efficient functions corresponding to the intrinsic operators of Python. For example, operator.add(x, y) is equivalent to the expression x+y. Many function names are those used for special methods, without the double underscores. For backward compatibility, many of these have a variant with the double underscores kept. The

variants without the double underscores are preferred for clarity.

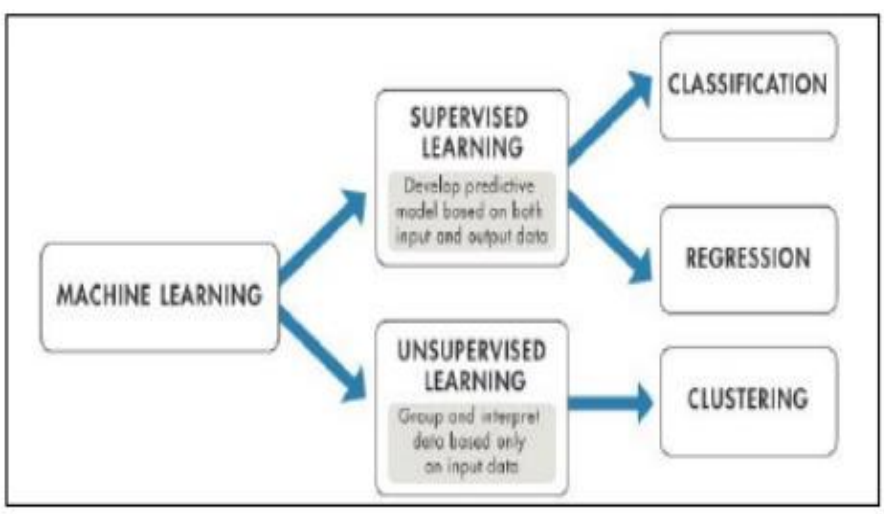
The functions fall into categories that perform object comparisons, logical operations, mathematical operations and sequence operations.

Tempfile: Generate temporary files and directories

This module creates temporary files and directories. It works on all supported platforms. TemporaryFile, NamedTemporaryFile, TemporaryDirectory, and SpooledTemporaryFile are high-level interfaces which provide automatic cleanup and can be used as context managers. mkstemp() and mkdtemp() are lower-level functions which require manual cleanup.

All the user-callable functions and constructors take additional arguments which allow direct control over the location and name of temporary files and directories. Files names used by this module include a string of random characters which allows those files to be securely created in shared temporary directories. To maintain backward compatibility, the argument order is somewhat odd; it is recommended to use keyword arguments for clarity.

Introduction to Machine Learning:



Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed. Evolved from the study of pattern recognition

and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed. Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:-

Supervised Learning:

Supervised learning is the machine learning task of inferring a function from labelled training data. [1] The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value. A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a “reasonable” way.

Unsupervised Learning:

Unsupervised learning is the machine learning task of inferring a function to describe hidden structure from "unlabelled" data (a classification or categorization is not included in the observations). Since the examples given to the learner are unlabelled, there is no evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning and reinforcement learning. A central case of unsupervised learning is the problem of density estimation

in statistics, though unsupervised learning encompasses many other problems (and solutions) involving summarizing and explaining key features of the data.

KNN:

K-nearest neighbours' algorithm is a non-parametric classification method first developed by Evelyn Fix and Joseph Hodges in 1951,[1] and later expanded by Thomas Cover.[2] It is used for classification and regression. In both cases, the input consists of the k closest training examples in data set. The output depends on whether k-NN is used for classification or regression:

- In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbour.
- In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbour.

K-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.[3][4] Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbours, so that the nearer neighbours contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbour a weight of 1/d, where d is the distance to the neighbour.[5] The neighbours are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.

Algorithm:

¬ Data Collection- I have collected data sets of weather from online website. I have downloaded the .csv files in which information was present.

¬ Data Formatting - The collected data is formatted into suitable data sets. I check the collinearity with mean temperature. The data sets which have collinearity nearer to 1.0 has been selected

¬ Model Selection - I have selected different models to minimize the error of the predicted value. The different models used are Linear Regression Linear Model.

¬ Training - The data sets was divided such that x_train is used to train the model with corresponding x_test values and some y_train kept reserved for testing.

¬ Testing - The model was tested with y_train and stored in y_predict . Both y_train and y_predict was compared.

Breast Cancer Prediction Codes

Group members and attribute information

Using the Wisconsin breast cancer diagnostic data set for predictive analysis

Group 5 Kartik Midya,Arin nath,Pronoy Chandra Mridha,Rupam Bit,Sorbajit Goswami

Attribute Information:

- 1. ID number
- 2. Diagnosis (M = malignant, B = benign)

Attribute Information:

- 1. ID number
- 2. Diagnosis (M = malignant, B = benign)

-3-32.Ten real-valued features are computed for each cell nucleus:

- a) radius (mean of distances from center to points on the perimeter)
- b) texture (standard deviation of gray-scale values)
- c) perimeter
- d) area
- e) smoothness (local variation in radius lengths)
- f) compactness (perimeter^2 / area - 1.0)
- g). concavity (severity of concave portions of the contour)
- h). concave points (number of concave portions of the contour)
- i). symmetry
- j). fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

Loading the libraries

```
#Load Libraries

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# keeps the plots in one place. calls image as static pngs
%matplotlib inline
import matplotlib.pyplot as plt # side-stepping mpl backend
import matplotlib.gridspec as gridspec # subplots
import mpld3 as mpl

#Import models from scikit learn module:
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import KFold   #For K-fold cross validation
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import metrics
```

[ ] Python

Preparing and loading the data

```
Load the data

df = pd.read_csv("../input/data.csv",header = 0)
df.head()
```

Python

```
Clean and prepare data

df.drop('id',axis=1,inplace=True)
df.drop('Unnamed: 32',axis=1,inplace=True)
# size of the dataframe
len(df)
```

Python

```
df.diagnosis.unique()
```

```
df.diagnosis.unique()
```

[ ] Python

```
df['diagnosis'] = df['diagnosis'].map({'M':1,'B':0})
df.head()
```

[ ] Python

Exploring the data

```
Explore data

df.describe()
```

[ ] Python

```
df.describe()
plt.hist(df['diagnosis'])
plt.title('Diagnosis (M=1 , B=0)')
plt.show()
```

[ ] Python

+ Code + Markdown

Nucleus features vs diagnosis

nucleus features vs diagnosis

```
features_mean=list(df.columns[1:11])
# split dataframe into two based on diagnosis
dfM=df[df['diagnosis'] ==1]
dfB=df[df['diagnosis'] ==0]
```

Python

```
#Stack the data
plt.rcParams.update({'font.size': 8})
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(8,10))
axes = axes.ravel()
for idx,ax in enumerate(axes):
    ax.figure
    binwidth= (max(df[features_mean[idx]]) - min(df[features_mean[idx]]))/50
    ax.hist([dfM[features_mean[idx]],dfB[features_mean[idx]]], bins=np.arange(min(df[features_mean[idx]]), max(df[features_mean[i
    ax.legend(loc='upper right')
    ax.set_title(features_mean[idx])
plt.tight_layout()
plt.show()
```


```
#Stack the data
plt.rcParams.update({'font.size': 8})
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(8,10))
axes = axes.ravel()
for idx,ax in enumerate(axes):
    ax.figure
    binwidth= (max(df[features_mean[idx]]) - min(df[features_mean[idx]]))/50
    ax.hist([dfM[features_mean[idx]],dfB[features_mean[idx]]], bins=np.arange(min(df[features_mean[idx]]), max(df[features_mean[i
    ax.legend(loc='upper right')
    ax.set_title(features_mean[idx])
plt.tight_layout()
plt.show()
```

Python

Observations

1. mean values of cell radius, perimeter, area, compactness, concavity and concave points can be used in classification of the cancer. Larger values of these parameters tends to show a correlation with malignant tumors.
2. mean values of texture, smoothness, symmetry or fractual dimension does not show a particular preference of one diagnosis over the other. In any of the histograms there are no noticeable large outliers that warrants further cleanup.

Creating a test set and a training set

Since this data set is not ordered, I am going to do a simple 70:30 split to create a training data set and a test data set.

```
traindf, testdf = train_test_split(df, test_size = 0.3)
```

Python

Model Classification

Model Classification

Here we are going to build a classification model and evaluate its performance using the training set.

```
#Generic function for making a classification model and accessing the performance.
# From AnalyticsVidhya tutorial
def classification_model(model, data, predictors, outcome):
  #Fit the model:
  model.fit(data[predictors],data[outcome])

  #Make predictions on training set:
  predictions = model.predict(data[predictors])

  #Print accuracy
  accuracy = metrics.accuracy_score(predictions,data[outcome])
  print("Accuracy : %s" % "{0:.3%}".format(accuracy))

  #Perform k-fold cross-validation with 5 folds
  kf = KFold(data.shape[0], n_folds=5)
  error = []
  for train, test in kf:
    # Filter training data
```

```
    print("Cross-Validation Score : %s" % "{0:.3%}".format(np.mean(error)))

  #Fit the model again so that it can be refered outside the function:
  model.fit(data[predictors],data[outcome])
```

Python

Logistic Regression model

Logistic regression is widely used for classification of discrete data. In this case we will use it for binary (1,0) classification.

Based on the observations in the histogram plots, we can reasonably hypothesize that the cancer diagnosis depends on the mean cell radius, mean perimeter, mean area, mean compactness, mean concavity and mean concave points. We can then perform a logistic regression analysis using those features as follows:

```
predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_mean','concave points_mean']
outcome_var='diagnosis'
model=LogisticRegression()
classification_model(model,traindf,predictor_var,outcome_var)
```

Python

```
predictor_var = ['radius_mean']
model=LogisticRegression()
classification_model(model,traindf,predictor_var,outcome_var)
```

Python

Decision Tree Model

Decision Tree Model

```
predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_mean','concave points_mean']
model = DecisionTreeClassifier()
classification_model(model,traindf,predictor_var,outcome_var)
```

Python

Here we are over-fitting the model probably due to the large number of predictors. Let use a single predictor, the obvious one is the radius of the cell.

```
predictor_var = ['radius_mean']
model = DecisionTreeClassifier()
classification_model(model,traindf,predictor_var,outcome_var)
```

Python

Randome Forest

Randome Forest

```
# Use all the features of the nucleus
predictor_var = features_mean
model = RandomForestClassifier(n_estimators=100,min_samples_split=25, max_depth=7, max_features=2)
classification_model(model, traindf,predictor_var,outcome_var)
```

[ ] Python

Using all the features improves the prediction accuracy and the cross-validation score is great.

+ Code + Markdown

An advantage with Random Forest is that it returns a feature importance matrix which can be used to select features. So lets select the top 5 features and use them as predictors.

```
#Create a series with feature importances:
featimp = pd.Series(model.feature_importances_, index=predictor_var).sort_values(ascending=False)
print(featimp)
```

[ ] Python

```
# Using top 5 features
predictor_var = ['concave points_mean','area_mean','radius_mean','perimeter_mean','concavity_mean',]
model = RandomForestClassifier(n_estimators=100, min_samples_split=25, max_depth=7, max_features=2)
classification_model(model,traindf,predictor_var,outcome_var)
```

[ ] Python

Using the top 5 features only changes the prediction accuracy a bit but I think we get a better result if we use all the predictors.

What happens if we use a single predictor as before? Just check.

```
predictor_var =  ['radius_mean']
model = RandomForestClassifier(n_estimators=100)
classification_model(model, traindf,predictor_var,outcome_var)
```

[ ] Python

This gives a better prediction accuracy too but the cross-validation is not great.

Applying it on the Test data set

Using on the test data set

```
# Use all the features of the nucleus
predictor_var = features_mean
model = RandomForestClassifier(n_estimators=100,min_samples_split=25, max_depth=7, max_features=2)
classification_model(model, testdf,predictor_var,outcome_var)
```

[ ] Python

The prediction accuracy for the test data set using the above Random Forest model is 95%!

Conclusion

The best model to be used for diagnosing breast cancer as found in this analysis is the Random Forest model with the top 5 predictors, 'concave points_mean','area_mean','radius_mean','perimeter_mean','concavity_mean'. It gives a prediction accuracy of ~95% and a cross-validation score ~ 93% for the test data set.

Future Scope

We will see if we can improve this more by tweaking the model further and trying out other models in a later version of this analysis.

THANK YOU