Classify Breast Cancer Benign Or Malignant

Using Random Forest Algorithm

Machine Learning Assignment

NAME : Widanagamachchi L.L

ID : IT15098474

BATCH : Weekend SE

Table of Contents

1. Problem Overview

Cancer is a group of diseases that cause cells in the body to change and spread out of control. Among those different cancer types, breast cancer can be considered as one of the most harmful ones. Breast cancer is the most common cancer among women which spreads widely. And also, it's the second main cause of cancer in death of women, after lung cancer. Most breast cancers begin either in the breast tissue, or in the ducts which is called ' ductal carcinomas '.

It is estimated that around 40,610 women and 460 men died from breast cancer in 2017. Breast facing cancer death rates are highest in African American women. There is a more probability of effecting the breast cancer in women which are above 80 years of age. There are many stages in a breast cancer and the tumor which grows inside the body causes death. Those tumors are mainly of 2 kinds.

- Benign Tumor (Non-Cancerous)

  Most of the breast changes are benign and they are not life-threatening. These tumor cells slowly grow locally and cannot spread all over. Benign Tumor can be fully recovered by proper treatments.

- Malignant Tumor (Cancerous)

  This kind of tumor is harmful and causes death when infected. These tumors grow so fast and spread to other surrounding tissues.

I have selected this topic because, Breast cancers are one of the major problems that most of the people around the world facing currently. Here I investigated to classify whether a breast cancer is Benign or Malignant based on a set of characteristics. I have predicted the possibility that a breast cancer being benign or malignant by analyzing a breast cancer dataset. Basically, Random Forest Algorithm is used to predict the results here.

2. Dataset

2.1 Dataset Description

I have used Breast Cancer Wisconsin (Diagnostic) Dataset to predict the results here. The relevant dataset can be found from the following links and features are computed from a digitized image.

This is one of the high-quality datasets and as the data can be downloaded in csv format, it can be easily parsed to the python language I have used Breast Cancer Wisconsin (Diagnostic) Dataset to predict the results here.

The relevant dataset can be found from the following links and features are computed from a digitized image.

On Kaggle,

https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/data

Also, can be found on UCI (University of California Irvine) Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

2.2 Attributes of the Dataset

Sample code number

1. ID number - To uniquely identify

One target class

2. Diagnosis (M = Malignant, B = Benign)

Likewise, there are all together 32 attributes in this dataset.
Ten real-valued features are computed here 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)

Missing attribute values: none

Number of Instances: 569

Class distribution: 357 benign, 212 malignant

Ratio of Malignant cases: 37.25%

For each characteristic three measures are given:

1. Mean
2. Standard Error
3. Largest / 'Worst' (mean of the three largest values)
   For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

3. Data Preprocessing

As the dataset is huge there is a huge probability of missing data. Missing data can impact an analysis as can incorrect data or outliers.

So here len() function is used to get the size of the data frame while shape() function is used to get the number of features.

```
# No of cases included in the dataset, size of the dataframe
length = len(dataset)
print 'No of cases in the dataset ->', str(len(dataset))

#No of features in the dataset
print 'No of features in the dataset ->', dataset.shape[1]-1
```

```
No of cases in the dataset -> 569
No of features in the dataset -> 32
```

And to identify anomalies describe() is used. It shows the data in descriptive way.

```
# Descriptive statistics for each column
print('---- Descriptive statistics For Breast Cancer Dataset ----')
dataset.describe()
```

```
---- Descriptive statistics For Breast Cancer Dataset ----
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341

3.1 Data Cleaning

I have used some data cleaning mechanisms before feeding it to the algorithm.

```
# Data Cleaning
dataset = dataset.drop(['id', 'Unnamed: 32'], axis = 1)
```

And for more efficiency I have converted the pandas dataframes in to array format.

```
# Convert to array
dataset['diagnosis'].unique()
```

```
array(['M', 'B'], dtype=object)
```

3.2 Data Preparation

We cannot just feed raw data into a model. So I have converted 'diagnosis' variable in to binary representation using map() function in pandas.

0 == Benign and 1 == Malignant

```
# Preparing data
dataset['diagnosis'] = dataset['diagnosis'].map({'M':1,'B':0})
dataset.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	c
0	1	17.99	10.38	122.80	1001.0	0.11840	

3.3 Dimensionality Reduction

In here, the variable 'diagnosis' have been removed.

```
# Remove the labels from the features
# Remove the factorial column 'diagnosis' to find the co-relation between numerical columns
features = dataset.drop('diagnosis', axis = 1)

# Saving features
new_features = list(features.columns)

features.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001

4. Methodology

My target was to classify whether a breast cancer is Malignant or Benign by analyzing a relevant dataset. As this is a classification this problem comes under Supervised Learning Module. So, to predict the results I basically used Random Forest Algorithm as it is more efficient for classification problems and for the implementation I have used python. Apart from that to predict more results I used Decision Tree Model. I imported the random forest regression model for creating and training the model.

4.1 Random Forest Algorithm

Random forest is one of popular machine learning algorithm which is mainly used for classification and regression. From the name itself random forest includes multiple decision trees, and the average of the result of each relevant decision tree would be the final outcome for random forest. Each decision tree can be considered as a single classifier. There are some drawbacks in decision tree such as over fitting on training dataset, but it was solved in random forest by the concept of Bagging (Bootstrap Aggregation).

Random forest is more appropriate for supervised learning such as classification and regression as it's an ensemble classification method. Ensemble classifier means a set of classifiers. Instead of using only one classifier to predict the final target, in ensemble, multiple classifiers are used to predict the target.

In random forest train set is divided in to smaller parts and make each small part as independent tree which its result has no effect on other trees besides them. Random Forest uses Bagging concept for creating multiple different models from a single training dataset. This increases accuracy by preventing over fitting and decreasing variance.

5. Implementation

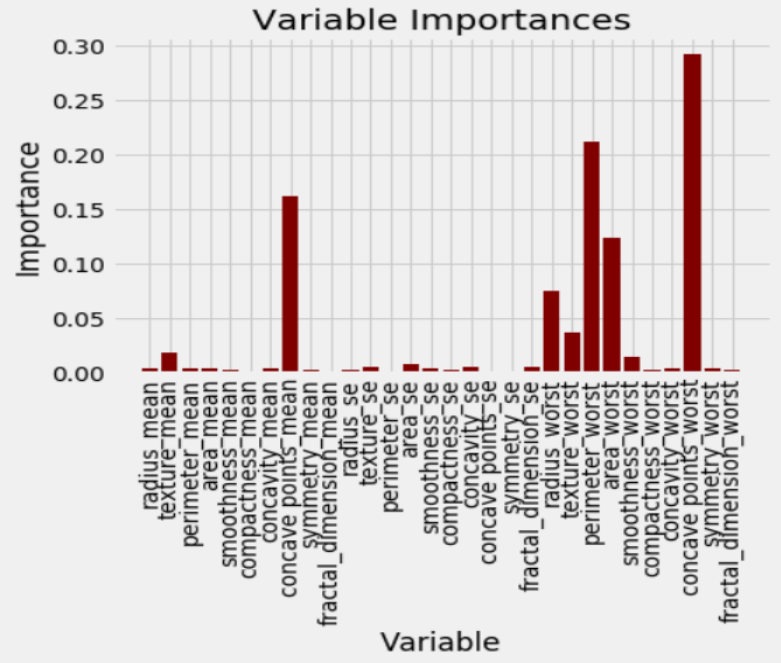
For the proper implementation I have used,

- Relevant Coding Standards
- Comments
- Proper Indentations
- Naming conventions - lower_case_with_underscore For Variables, functions, methods, modules

6. Results

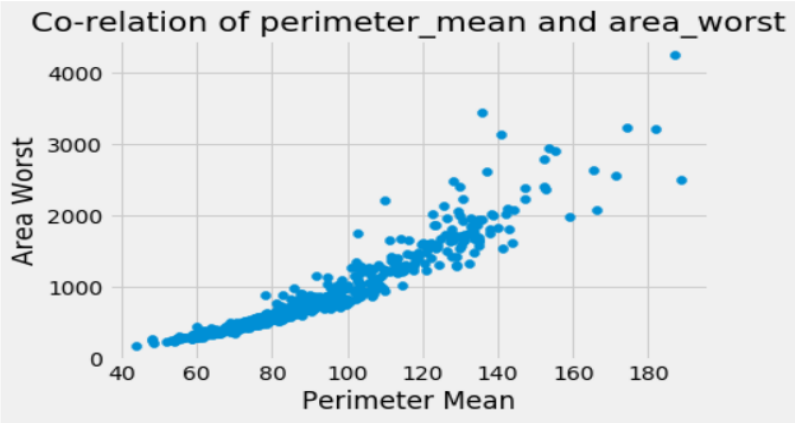
matplotlib is used to generate statistical graphs.

6.1 Variable Importances



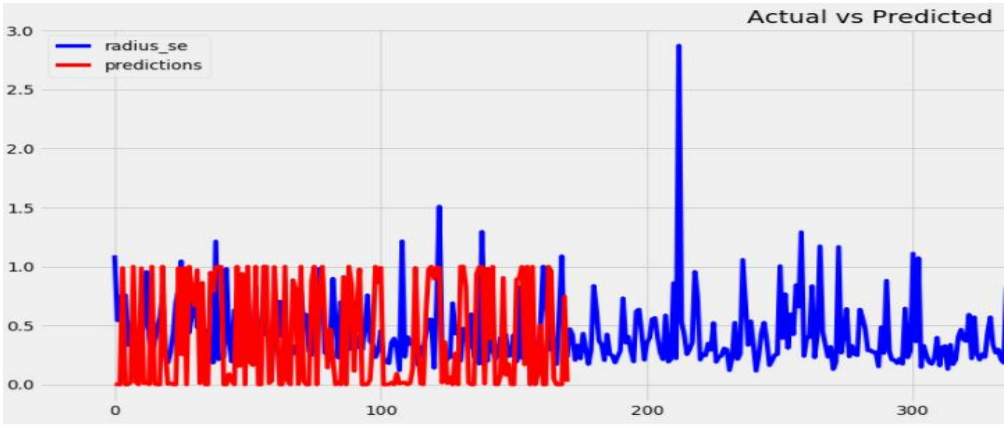
This graph shows the importance of each and every variable and how much each variable improves the prediction. So according to the graph 'concave points_worst' has more importance than the other variables. The second most factor is the 'perimeter_worst'. This can be used as a feature selection method and can remove the variables that has no importance. It doesn't affect the performance.

6.2 Co-Relation between Variables



Normally variables that relate to the same attribute are highly correlated with each other (as expected). Above graph shows one of these examples where the perimeter mean and radius worst are highly correlated. Here the correlation of determination is .94. This indicates that 94% of the changes in perimeter mean are explained by changes happen in the radius worst. Most researches show that the variables which are highly co-related should be removed from further analysis.

6.3 Actual VS Predicted



This shows the actual values and the values that were predicted by the Random Forest Algorithm. According to the above graph most of the values have being predicted correctly. It proves that the Random Forest Algorithms is more suitable for predicting the breast cancer dataset.

7. Critical Analysis and Discussion

7.1 Evaluation Metrics

```
print "In Random Forest \n"
print "  f1_score is -> " , score
print '  Accuracy is -> ' , metrics.accuracy_score(test_y, pred)
print "  cross_val_score is ->", (cross_val_score(model, features[selected_cols], target , cv = 10).mean())
print "  Time for training dataset -> ", training_time
print "  Time for prediction -> ", prediction_time
```

```
In Random Forest

  f1_score is ->  0.923076923076923
  Accuracy is ->  0.9415204678362573
  cross_val_score is -> 0.9561641171895255
  Time for training dataset ->  0.170000076294
  Time for prediction ->  0.00600004196167
```

F1 Score

F1 Score can be defined as a measurement of accuracy or the ratio of the data that was predicted. When F1 score is closer to 1 it's the best the prediction and when closer to 0 it's the worse the prediction. F1 score considers the true positives and the true negatives and is best used when comparing various classifiers. F-score can be considered as the best evaluation metric to be used for this type of classification problem. Here is the formula for the F1 score from the sklearn documentation,

F1 Score = 2* (precision * recall) / (precision + recall). F1 Score = 92%

Accuracy

According to the results, using a single predictor rather than using all the features improves the prediction accuracy. In Random Forest, it returns a feature importance matrix which can be used to select features.

Normally when there are more trees in the forest the more robust the forest looks like. Likewise, in the random forest classifier, the higher the number of trees in the forest gives the high accuracy results.

Accuracy = 94%

It proves that Random Forest is more suitable for this problem.

Cross Val Score

Using all the features improves the cross-validation score.

cross_val_score = 95 %

7.2 Improve Performance

The performance of any model is directly proportional to the amount of valid data. So can improve the performance by increasing the valid data amount.

Some hyperparameters in random forest can be used to increase the predictive power of the model or to make the model faster.

1. For Increasing Predictive Power:

   n_estimators - The number of trees the algorithm builds, higher number of trees increase the performance.

   max_features - The maximum number of features in an individual tree

2. For Increasing the Models Speed:

   n_jobs - How many processors it is allowed to use

   random_state - Makes the model's output replicable

8. Appendix

```
# coding: utf-8

# In[45]:

# For data manipulation, data processing

import pandas as pd

from time import time

# Import sklearn models

from sklearn.metrics import f1_score

from sklearn import metrics

from sklearn.metrics import classification_report

# To calculate the accuracy of the trained classifier

from sklearn.metrics import accuracy_score

# To understand the trained classifier behavior

from sklearn.metrics import confusion_matrix

# For spliting data into training and testing sets
```

```
from sklearn.model_selection import train_test_split
# For Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
# For Desicion Tree Model
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import cross_val_score
get_ipython().magic(u'matplotlib inline')
import matplotlib.pyplot as plt
from matplotlib import pyplot
# In[46]:
# print Header
print('---- Breast Cancer Sample Dataset ----')
# Read data in CSV file
dataset = pd.read_csv('E:\\4th Yr\\ML\\Assignmntz\\data.csv')
# INPUT_PATH = "..//Assignmntz//data.csv"
# dataset = pd.read_csv(INPUT_PATH)
# Display first 10 rows
dataset.head(10)
# In[47]:
# No of cases included in the dataset, size of the dataframe
length = len(dataset)
print 'No of cases in the dataset ->', str(len(dataset))
# No of features in the dataset
print 'No of features in the dataset ->', dataset.shape[1]-1
# In[48]:
# Descriptive statistics for each column
print('---- Descriptive statistics For Breast Cancer Dataset ----')
dataset.describe()
```

```
# In[49]:

# Data Cleaning

dataset = dataset.drop(['id', 'Unnamed: 32'], axis = 1)

# dataset.drop('id',axis=1, inplace=True)

# dataset.drop('Unnamed: 32', axis=1, inplace=True)

# In[50]:

# Convert to array

dataset['diagnosis'].unique()

# In[51]:

# Preparing data

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

dataset.head()

# In[52]:

# target(label) - value we want to predict

target = dataset['diagnosis']

# Remove the labels from the features

# Remove the factorial column 'diagnosis' to find the co-relation between numerical columns

features = dataset.drop('diagnosis', axis = 1)

# Saving features

new_features = list(features.columns)

features.head()

# In[53]:

# Convert to array

target.unique()

# In[54]:

features.describe()

# In[55]:

target.describe()

# In[56]:
```

```
# To split the dataset
def split_data(features, target):
      # Split dataset into train and test dataset
   train_x, test_x, train_y, test_y = train_test_split(features, target, test_size= 0.3, random_state=101)
   print "Training Features Shape -> ", train_x.shape
   print "Training Labels Shape   -> ", train_y.shape
   print "Testing Features Shape  -> ", test_x.shape
   print "Testing Labels Shape    -> ", test_y.shape
   print("\n"
   return train_x, test_x, train_y, test_y
# In[57]:
def random_forest_classifier(train_x, train_y):
   ran_forest = RandomForestClassifier(n_estimators=100)
#    Train the model on training data
   ran_forest.fit(train_x, train_y)
#    print("1")
   print "Trained model -> \n", ran_forest
   return ran_forest
# In[58]:
def main():
   # Train Test Split
   split_data(features, target)
   train_x, test_x, train_y, test_y = train_test_split(features, target, test_size= 0.3, random_state=101)
   # Using Decision Tree Model
   decs_tree = DecisionTreeClassifier()
   decs_tree.fit(train_x, train_y)
   desc_pred = decs_tree.predict(test_x)
   print("---- Decision Tree Model -----\n")
   print " Classification Report -> \n"
```

```
    print(classification_report(test_y, desc_pred))
    print " Confusion Matrix -> "
    print(confusion_matrix(test_y, desc_pred))
    print("\n")
    print("---- Random Forests -----\n")
#    Using Random Forests
    trained_model  = random_forest_classifier(train_x, train_y)
    ran_forest = RandomForestClassifier(n_estimators=100)
#    Train the model on training data
    ran_forest.fit(train_x, train_y)
#    Perform predictions
    ran_pred = ran_forest.predict(test_x)
    print("\n")
    print " Classification Report -> \n"
    print(classification_report(test_y, ran_pred))
    for i in xrange(0, 5):
        print "Actual outcome -> {} and Predicted outcome -> {}" .format(list(test_y)[i], ran_pred[i])
#    Train and Test Accuracy
    print("\n")
    print " Train Accuracy -> ", accuracy_score(train_y, trained_model.predict(train_x))
    print " Test Accuracy  -> ", accuracy_score(test_y, ran_pred)
#    Confusion matrix
    print " Confusion Matrix -> "
    print(confusion_matrix(test_y, ran_pred))
if __name__ == "__main__":
    main()
# In[59]:
def predict_model(model, train_x, test_x, train_y, test_y, selected_cols):
#    Calculate Training Time
```

```
    t0 = time()
    model.fit(train_x[selected_cols], train_y)
    training_time = time() - t0
#    Calculate Prediction Time
    t1 = time()
    pred = model.predict(test_x[selected_cols])
    prediction_time = time() - t1
#     Calculate f1_score
    score = f1_score(test_y, pred
    print "In Random Forest \n"
    print "  f1_score is -> " , score
    print '  Accuracy is -> ' , metrics.accuracy_score(test_y, pred)
    print "  cross_val_score is ->", (cross_val_score(model, features[selected_cols], target , cv = 10).mean())
    print "  Time for training dataset -> ", training_time
    print "  Time for prediction -> ", prediction_time
# In[60]:
Forest = RandomForestClassifier(max_depth=5, n_estimators=50, max_features=1
# In[61]:
train_x, test_x, train_y, test_y = train_test_split(features, target, test_size= 0.3, random_state=101)
predict_model(Forest, train_x, test_x, train_y, test_y, new_features
# In[62]:
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(train_x, train_y);
# In[63]:
# Get numerical feature importances
importances = list(rf.feature_importances_)
# List of tuples with variable and importance
```

```
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(new_features, importances)]

# Sort the feature importances by most important first

feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)

# Print out the feature and importances

for pair in feature_importances:

    print 'Variable: {:30} Importance: {}'.format(*pair)

# Visualize the output in a bar chart

# Set the style

plt.style.use('fivethirtyeight')

# list of x locations for plotting

x_val = list(range(len(importances)))

# Make a bar chart

plt.bar(x_val, importances, orientation = 'vertical' , color='maroon')

# Tick labels for x axis

barlist = plt.xticks(x_val, new_features, rotation='vertical')

# Axis labels and title

plt.ylabel('Importance');

plt.xlabel('Variable');

plt.title('Variable Importances')

# barlist = pyplot.bar(range(len(importance)), importance)

 # print (sorted(zip(map(lambda x: round(x, 4), importance), names), reverse=True))

# Axis labels and title

pyplot.ylabel('Importance');

pyplot.xlabel('Variable');

pyplot.title('Variable Importances');

pyplot.show()

# In[64]:

df = pd.DataFrame(dataset)
```

```
figure, graph = plt.subplots(1)
for i in range(1):
   x=df['perimeter_mean']
   y=df['area_worst']
   graph.scatter(x,y, label=str(i))
plt.ylabel('Area Worst');
plt.xlabel('Perimeter Mean');
plt.title('Co-relation of perimeter_mean and area_worst');
figure.savefig('relation.png')
# In[65]:
yTest = dataset['radius_se'].as_matrix()
# Use the random forest's predict method on the test data
predictions = rf.predict(test_x)
plt.figure(figsize=(21,7))
plt.plot(yTest,label='radius_se',color='blue')
plt.plot(predictions,label='predictions',color='red')
plt.title('Actual vs Predicted')
plt.legend(loc='upper left')
plt.show()
```