5505 Assignment 4: Random Forest

Shiva Chakravarthy Gollapudi
Student ID: 11468697

Random Forest: It is a supervised learning algorithm and capable of performing both regression and classification tasks.

Implementing a Random Forest Model on a data set containing descriptive attributes of digitized images of a process known as, fine needle aspirate (FNA) of breast mass.

Data: Wisconsin Breast Cancer

Tools and Environment: Using python programming and Google Colab, I have explored the data and created Linear Regression models.

Step 1: Load Libraries

Import the libraries that we use to load, explore data and build the model.

```
In [1]: import warnings
        warnings.filterwarnings('ignore')

In [2]: from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = 'all'

In [3]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import time

        %matplotlib inline

        plt.style.use('fast')
        sns.set_style('whitegrid')
```

Step 2: Load the dataset

Load the dataset, we have a total of 29 features that were computed for each cell nucleus with an ID Number and the Diagnosis (Later converted to binary representations: Malignant = 1, Benign = 0).

```
In [1]: import warnings
        warnings.filterwarnings('ignore')

In [2]: from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = 'all'

In [3]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import time

        %matplotlib inline

        plt.style.use('fast')
        sns.set_style('whitegrid')
```

Step 3: Explore the data:

First finding the missing values.

```
# Finding the missing values

data.isnull().sum()
```

There are 569 missing values in unnamed column, so we are dropping the column and ID is unique which is not helpful for our predictions.

```
# Dropping unnamed and id
data.drop(columns=['Unnamed: 32','id'],inplace=True)
```

After dropping two columns, will be having 29 features for analysis.

Statistical data analysis:

```
# Explore the dataset with some summary statistics
data.describe()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200

Decision tree or random forest algorithms are good in handling the data which is having different scales or ranges. From the above stats our data is having different scales or ranges of Min, Max and Std Dev.

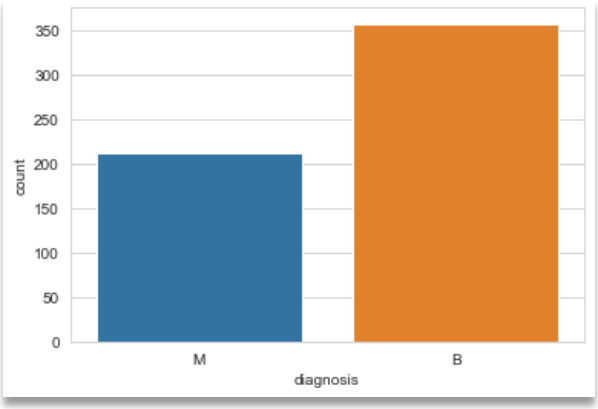
Exploring the target variable:

The dependent variable "diagnosis" contains binary values: M (malignant) - 212 and B (benight) - 357. Convert this diagnosis variable into binary

```
# Exploring the target variable
sns.countplot(x='diagnosis', data=data)
print(data['diagnosis'].value_counts())
plt.show();
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

Visualizing the target data:



The dependent variable "diagnosis" contains binary values: M (malignant) - 212 and B (benight) - 357. Convert this diagnosis variable into binary.

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	1	17.99	10.38	122.80	1001.0	0.11840
1	1	20.57	17.77	132.90	1326.0	0.08474
2	1	19.69	21.25	130.00	1203.0	0.10960
3	1	11.42	20.38	77.58	386.1	0.14250
4	1	20.29	14.34	135.10	1297.0	0.10030

Converted the diagnosis column into binary, malignant as 1 and benign as 0.

Step: 4 Model Building

Random Forest: It is a supervised learning algorithm and capable of performing both regression and classification tasks.

Splitting the data into training and testing sets Here we will split the data into two parts: training set (80%), and test set (20%).

```
# Spliting the data into training (80%) and testing (20%) sets
from sklearn.model_selection import train_test_split
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']
X_train, X_test, y_train, y_test = train_test_split (X, y,train_size = 0.8)
print ('Shapes of X_train, y_train: ', X_train.shape, y_train.shape)
print ('Shapes of X_test, y_test: ', X_test.shape, y_test.shape)
```

```
Shapes of X_train, y_train:  (455, 30) (455,)
Shapes of X_test, y_test:  (114, 30) (114,)
```

K-fold cross validation:

In K-fold cv, training data is further split into K number of subsets, called folds, then iteratively fit the model k times, each time training the data on k-1 of the folds and evaluating on the kth fold. At the end, we average the performance on each of the folds to come up with final validation metrics for the model.

Model 1: K-fold CV:

```
# Validate the model's performance using k-fold cross validation
from sklearn.model_selection import cross_validate
cv = cross_validate (rfc1, X_train, y_train, cv = 10)
```

```
print("Accuracy score of 10-fold cross validation: ", cv['test_score'])
print("Accuracy mean score of CV: ", cv['test_score'].mean())
```

```
Accuracy score of 10-fold cross validation:  [0.95652174 0.95652174 0.97826087 0.95652174 1.         0.97777778
 0.91111111 0.95555556 0.95555556 0.95555556]
Accuracy mean score of CV:  0.9603381642512078
```

The base model performed quite well, with an accuracy score of 0.96 on the training set. However, we still hope to improve the performance by tuning the hyperparameters.

Model 2: **Hyper-Parameter tuning:**

For hyperparameter tuning, we perform many iterations of the entire K-fold CV process, each time using different model settings.

Created a dictionary of all the parameters and their corresponding set of values that we want to test the performance.

```
from sklearn.model_selection import RandomizedSearchCV
```

```
random_grid = {'n_estimators' :[int(x) for x in np.linspace(start = 100, stop = 1000, num = 10, endpoint = True)],
               'max_features': ['auto', 'sqrt', 'log2'],
               'max_depth': [int(x) for x in np.linspace(start = 3, stop = 36, num=33, endpoint = True)],
               'min_samples_split': [5, 10, 15],
               'min_samples_leaf': [3, 4, 5],
               'bootstrap': ['True']}
print(pram_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000], 'max_features': ['auto', 'sqrt', 'log2'], 'max_
depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 36], 'min_samples_split': [5, 10, 15], 'min_samples_leaf': [3, 4, 5], 'bootstrap': ['True']}
```

Create a randomized search cross validation model for searching for the best hyperparameters for the base rainforest model over 100 parameters.

```
# Creating a base randomforest model for tuning
tune_base_rf = RandomForestClassifier(random_state=42)

# Create a randomized search cross validation model for searching for the best hyperparameters for the base rf model ov
random_search_rf = RandomizedSearchCV (estimator=tune_base_rf, param_distributions  = random_grid, random_state=42, cv

# fit the randomized search CV model into the training set
random_search_rf.fit(X_train, y_train)

# Print the best parameters
random_search_rf.best_params_
```

From the above code we found the best parameters for our tuned random forest. Will use these parameters to build the high-performance model.

Model 3: Tuned random forest

Creating a tuned random forest model with the best parameters chosen by the cv randomized search algorithm.

```
# Creating a tuned random forest model with the best parameters choosen by  the cv randomized search algorithm
tuned_rf = RandomForestClassifier (n_estimators = 100, min_samples_split = 5, min_samples_leaf = 3, max_features = 'aut

# Validating the model using k-fold cross validation (k=10)
tuned_cv = cross_validate (tuned_rf, X_train, y_train, cv = 10)
print("Accuracy score: ", tuned_cv['test_score'])
print("Accuracy mean score of CV: ", tuned_cv['test_score'].mean())
```

```
Accuracy score:  [0.93478261 0.97826087 0.97826087 0.95652174 0.95652174 0.97777778
 0.91111111 0.95555556 0.95555556 0.95555556]
Accuracy mean score of CV:  0.9559903381642512
```

The tuned model accuracy is 95.5% where the base model is 96.0%, the tuned model did not show to have better performance compared to the base model. Its because of decreasing the maximum depths of trees in the tuned model. But still hyperparameters is useful for avoiding the overfitting or high variance of the model on the unseen data. Therefore, we decide to go with the tuned model as our final model.

Step 4: Evaluating the selected model

We used accuracy score, confusion matrix, and ROC Curve to evaluate the model.

```
# Fit the selected model to the training set
tuned_rf.fit(X_train, y_train)
```

```
RandomForestClassifier(max_depth=26, min_samples_leaf=3, min_samples_split=5,
                       random_state=42)
```

Applying the selected model to make prediction on the test set and Observing the estimate probability of classes in the test set.

```
# Applying the selected model to make prediction on the test set
pred = tuned_rf.predict(X_test)

# Observing the estimate probability of classess in the test set
pred_prob = tuned_rf.predict_proba (X_test)
print ('class_0','\t', 'class_1')
print(pred_prob[:5])
```

```
class_0          class_1
[[0.99333333 0.00666667]
 [0.97340476 0.02659524]
 [0.00333333 0.99666667]
 [0.96844444 0.03155556]
 [0.86806349 0.13193651]]
```

In the first instance, it was assigned class 0 because it got a probability is almost 1 and higher than the class_1.

Accuracy Scores:

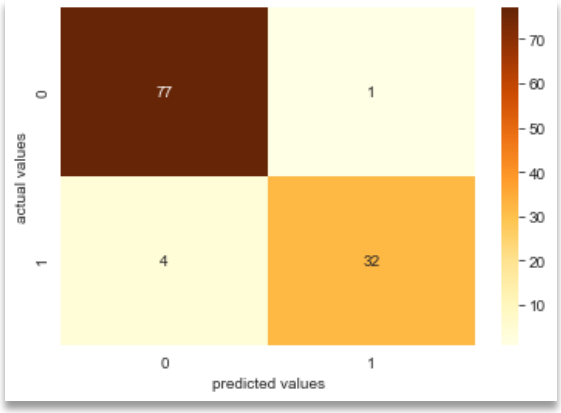
finding the accuracy scores for the test set

```
print('Accuracy of the selected model in the test set: {:.4f}'.format(tuned_rf.score(X_test, y_test)))
```

```
Accuracy of the selected model in the test set: 0.9561
```

Confusion matrix: From the below confusion matrix, we can see among 104 instances in test set, our model correctly predicts 82 Benign and 27 Malignant instances. It only incorrectly predicts 4 instances which are actually Benign (FP), and 1 instance that is actually Malignant (FN). Lower FN indicates that our model satisfies the cost requirement as mentioned previously.

```
# confusion matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, pred )
# visualizing confusion matrix
sns.heatmap(conf_matrix, annot = True)
plt.xlabel ('predicted values')
plt.ylabel ('actual values')
```



With the ROC curve

An ROC (Receiver Operating Characteristic Curve) is a graph showing the performance of a classification model at all classification thresholds.

Higher the AUC is the better the model performs. The AUC is expected to greater than 0.5, or over left-top part compared to the baseline.

TPR: TPR is the probability that an actual positive will test positive.

$$TPR = TP/P = TP/(TP+FN)$$

FPR: FPR is the model mistakenly predicted the positive class

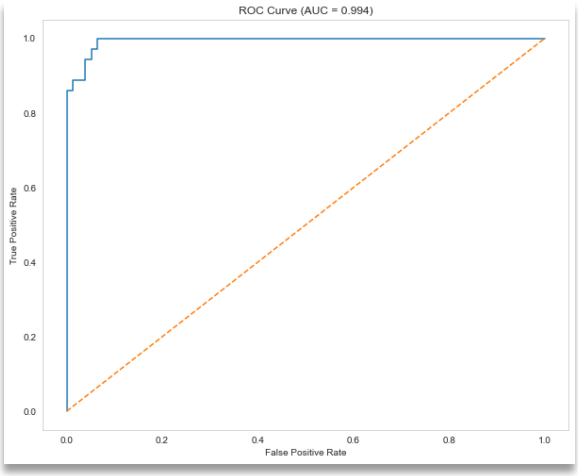
$$FPR = FP/N = FP/ (FP+TN)$$

```
from sklearn.metrics import roc_curve, roc_auc_score
# Taking the probability of the class_1 on the test set
pred_prob_c1 = pred_prob[:,1]

# Getting True Positive Rate (tpr) and False Positive Rate (fpr)
fpr, tpr, threshold = roc_curve (y_test,pred_prob_c1, pos_label = 1)

# Computing Area Under the ROC Curve (roc_auc)
roc_auc_score = roc_auc_score (y_test,pred_prob_c1)
roc_auc_score
```

Accuracy score is 0.994

Visualizing the ROC curve:

```
# Visualizing the ROC Curve
plt.subplots(figsize = (10,8))
plt.plot( fpr, tpr, label='Test AUC: %0.2f'%roc_auc_score)
plt.plot([0,1], '--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title ("ROC Curve (AUC = {:.3f})".format(roc_auc_score))
plt.grid()
plt.show();
```



Conclusion: The tuned model accuracy is 95.5% where the base model is 96.0%, the tuned model did not show to have better performance compared to the base model. It's because of decreasing the maximum depths of trees in the tuned model. But still hyperparameters is useful for avoiding the overfitting or high variance of the model on the unseen data. Therefore, we decide to go with the tuned model as our final model.