

B.M.S College of Engineering

P.O. Box No.: 1908 Bull Temple Road,
Bangalore-560 019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING



Course – Machine Learning
Course Code - 20IS5PCMLG

AY 2019-20

“BREAST CANCER PREDICTION USING KNN, DECISION TREES AND RANDOM FOREST”

Submitted to –

Rashmi R

Assistant Professor,

Dept of Information Science and Engineering

Submitted by –

Prabhu N	1BM19IS096
Mohan D	1BM18IS092
Moksh Jayanth	1BM19IS094

CONTENTS

Table of contents	Page No
1. Abstract	3
2. Introduction	4
3. Problem Statement	5
4. Literature Survey a. Machine Learning Algorithms For Breast Cancer Prediction And Diagnosis (2021) b. Breast Cancer Detection Using K-Nearest Neighbour Algorithm (2018) c. Random forest for breast cancer prediction (2019)	6 7 8
5. System Requirement Specifications	10
6. System Design / Flow diagram	11
7. Implementation and Deploying the Model	12 - 22
8. Test Results	23-25
9. Conclusion	26
10. Reference	27

ABSTRACT

Each year the number of deaths is increasing extremely because of breast cancer. It is the most frequent type of all cancers and the major cause of death in women worldwide. Any development for prediction and diagnosis of cancer disease is capital important for a healthy life.

Consequently, high accuracy in cancer prediction is important to update the treatment aspect and the survivability standard of patients. Machine learning techniques can make a large contribution to the process of prediction and early diagnosis of breast cancer, becoming a research hotspot and has been proved as a strong technique. **In this project, we applied two out of the popular machine learning algorithms: Decision tree and K-Nearest Neighbors (KNN) on the Breast Cancer Wisconsin Diagnostic dataset**, after obtaining the results, a performance evaluation and comparison is carried out between these different classifiers.

The main objective of this project is to predict and diagnose breast cancer, using machine-learning algorithms, and find out the most effective with respect to recall. All the work is done in the Anaconda environment based on the python programming language and Scikit-learn library.

Chapter 1 - Introduction

Breast cancer (BC) is one of the most common cancers among women worldwide, representing the majority of new cancer cases and cancer-related deaths according to global statistics, making it a significant public health problem in today's society.

The early diagnosis of BC can improve the prognosis and chance of survival significantly, as it can promote timely clinical treatment to patients. Further accurate classification of benign tumors can prevent patients undergoing unnecessary treatments. Thus, the correct diagnosis of BC and classification of patients into malignant or benign groups is the subject of much research. Because of its unique advantages in critical features detection from complex BC datasets, machine learning (ML) is widely recognized as the methodology of choice in BC pattern classification and forecast modelling.

Chapter 2 - Problem statement

Using the Breast Cancer Wisconsin (Diagnostic) Data Set we predict the outcome of the tumor cells (i.e., whether it is malignant or benign).

We chose to use two of the classification models:

1. Decision Trees
2. KNN Classifier
3. Random Forest

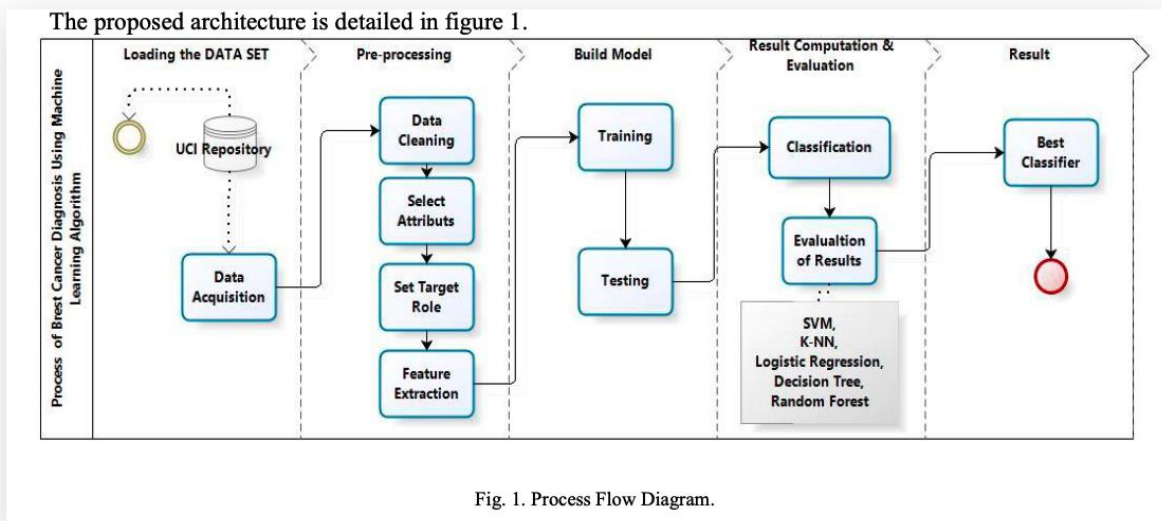
MAIN AGENDA:

Find which model is best suited for prediction of breast cancer by comparing the recall score of the respective models. **And deploying the model for Production on the Internet.**

Chapter 3 - Literature Survey

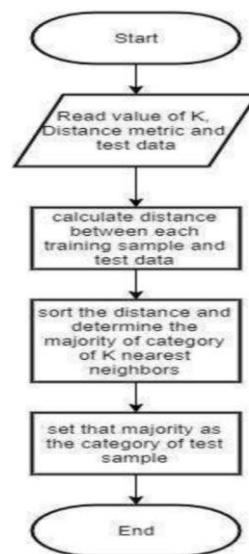
Paper 1 : Machine Learning Algorithms For Breast Cancer Prediction And Diagnosis (2021)

On the Wisconsin Breast Cancer Diagnostic dataset (WBCD) they applied five main algorithms which are: SVM, Random Forests, Logistic Regression, Decision Tree, K-NN, calculate, compare and evaluate different results obtained based on confusion matrix, accuracy, sensitivity, precision, AUC to identify the best machine learning algorithm that are precise, reliable and find the higher accuracy. All algorithms have been programmed in Python using scikit-learn library in Anaconda environment. After an accurate comparison between our models, we found that Support Vector Machine achieved a higher efficiency of 97.2%, Precision of 97.5%, AUC of 96.6% and outperforms all other algorithms. In conclusion, Support Vector Machine has demonstrated its efficiency in Breast Cancer prediction and diagnosis and achieves the best performance in terms of accuracy and precision. It should be noted that all the results obtained are related just to the WBCD database, it can be considered as a limitation of our work, it is therefore necessary to reflect for future works to apply these same algorithms and methods on other databases to confirm the results obtained via this database, as well as, in our future works, we plan to apply our and other machine learning algorithms using new parameters on larger data sets with more disease classes to obtain higher accuracy.



Paper 2 : Breast Cancer Detection Using K-Nearest Neighbour Algorithm (2018)

They have implemented KNearest Neighbour Algorithm using various normalization techniques and distance functions at different values of K. A comparative study using various normalization techniques, i.e., Min-Max normalization, Z-Score normalization and Decimal Scaling normalization, and different distance metrics, i.e., Manhattan distance, Euclidean distance, Chebyshev distance and Cosine distance has been done. The accuracy of each variation is tested and the maximum accurate prediction is considered for the result. Highest accuracy of 98.24% is achieved, with KNN implementation using Manhattan distance metric, at K=14, along with Decimal scale normalization.



Flow chart of K Nearest Neighbours.

KNN accuracy using various normalization techniques and distance functions

Distance Function	Euclidean distance	Manhattan distance	Chebyshev distance	Cosine distance
Normalization Technique				
Min-Max	95.6140, K=8	97.3684, K=6	92.1053, K=6	86.8421, K=14
Z-Score	95.6140, K=4	95.6140, K=6	94.7368, K=4	94.7368, K=4
Decimal Scaling	94.7368, K=4	98.2456, K=14	48.2456, K=4	68.4211, K=10

Paper 3 : Random forest for breast cancer prediction (2019)

In the research paper “*Random forest for breast cancer prediction*”,

They've used x % of data as training data and the rest as testing data on each experiment to validate the algorithm, where x = 10, 20, ..., 90. The experiment was repeated nine times. The data were chosen randomly in every experiment. Classification accuracy was measured by:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \times 100 \%$$

where:

TP : True Positive

TN : True Negative

FP : False Positive

FN : False Negative

They've used a personal computer (PC) with Intel i5 processor, 4 GB RAM, and Canopy Version: 2.1.8.3709 (64 bit) software.

The experiment results by using Random Forest can be seen in Table 2.

From the results in Table 2, we can say that this method for the classification of breast cancer data is auspicious to produce good accuracy. **This method can significantly help doctors in their final decision. By using such an efficient method, doctors can provide very accurate decisions.**

TABLE 2. Accuracy percentage for Classification using Random Forests

Training Data	Accuracy
10 %	99.52 %
20 %	99.82 %
30 %	99.37 %
40 %	99.52 %
50 %	99.43 %
60 %	99.64 %
70 %	99.52 %
80 %	100.00 %
90 %	100.00 %

They've classified breast cancer using a random forest method. The result in this paper is more than 99 %.

Moreover, as we can conclude that when they've used 80 % and 90 % of the data for the training data, the accuracy that they've got was 100 %, which means it is the best overall accuracy for breast cancer prediction.

Chapter 4 - System Requirement Specifications

Hardware requirements

Recommended System Requirements

- Processors: Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAM Intel® Xeon® processor E5-2698 v3 at 2.30 GHz (2 sockets, 16 cores each, 1 thread per core), 64 GB of DRAM Intel® Xeon Phi™ processor 7210 at 1.30 GHz (1 socket, 64 cores, 4 threads per core), 32 GB of DRAM, 16 GB of MCDRAM (flat mode enabled)
- Disk space: 2 to 3 GB
- Operating systems: Windows® 10, macOS*, and Linux*

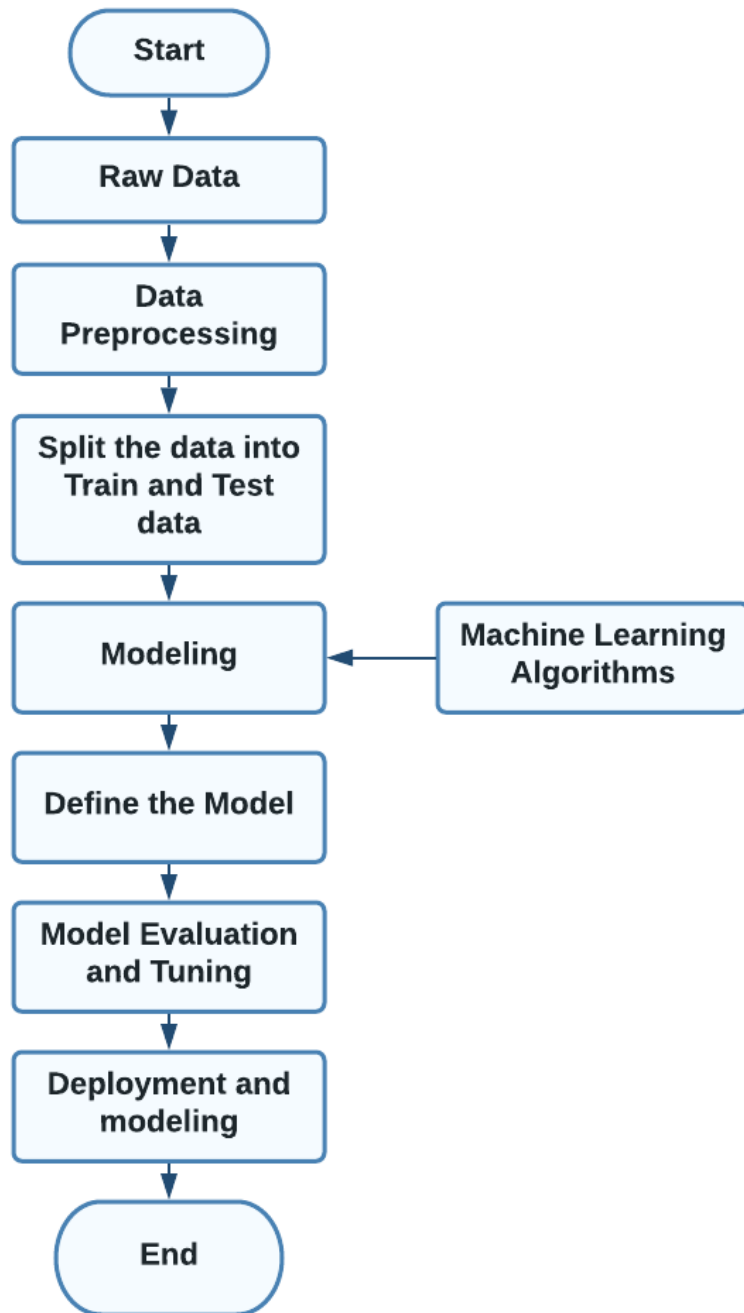
Minimum System Requirements

- Processors: Intel Atom® processor or Intel® Core™ i3 processor
- Disk space: 1 GB
- Operating systems: Windows* 7 or later, macOS, and Linux
- Python* versions: 2.7.X, 3.6.X
- Pandas* version: 0.25.0
- Scikit-learn* version: 0.21

Minimum Application Requirements

- A Standard Web Browser
- Google Colaboratory - <https://colab.research.google.com>
- Jupyter Labs - <https://jupyter.org>

Chapter 5 System Design / flow diagram



Chapter 6 Implementation (minimum 2 models)

Summary of the Project Implementation:

1. Implemented three classifier models:
 - a. KNN Classifier
 - b. Random Forest Classifier
 - c. Decision Tree Classifier
2. Found the Best parameters for each of the models (such as best_k, best_depth, best_estimator) and trained each optimized model on Wisconsin Breast Cancer Diagnostic dataset (WBCD) available on [kaggle](#).
3. Compared each model's performance on the Dataset using Recall Score as the metric.
4. The model with the best Recall Score was selected (*Decision Tree in this case*).
5. We used GridSearchCV to optimize the Tree model further by selecting the best set of parameters.
6. We calculated the feature importance for the model. Turned out only 4 of 32 features were important.
7. So, We Trained the Final Tree model with only 4 features (`concave_points_mean`, `radius_worst`, `area_worst`, `area_se`) and the Deployed the model on the Internet for Production.

Steps taken to Deploy the Model:

1. Trained the Final Tree Model with only 4 features from the Wisconsin Breast Cancer Diagnostic dataset (WBCD).
2. Created a function `predict()`:
 - a. Parameters: `concave_points_mean`, `radius_worst`, `area_worst`, `area_se`
 - b. Return value: *String* (Whether the tumor is *Benign* or *Malignant*).
 - c. Description: Predicts the output class (*Benign* or *Malignant*) using the Tree classifier model based on the arguments passed by the user.
3. We Deployed the `predict()` onto the web by using Python's library `gradio`. (*Code given below.*)
4. Now any user online can pass the arguments to the model by visiting the generated url and get the output.

Code for the Deployment:

1. *Code Snippet for `predict()` :*

```
def predict(concave_points_mean, radius_worst, area_worst, area_se):
    scaled = robust.transform([[concave_points_mean, radius_worst,
area_worst, area_se]])
    print(scaled)
    result = grid.predict(scaled)
    print(result)
    if(result[0]==0):
        return "The Tumor is Benign (noncancerous)."
    else:
        return "The Tumor is Malignant (cancerous)."
```

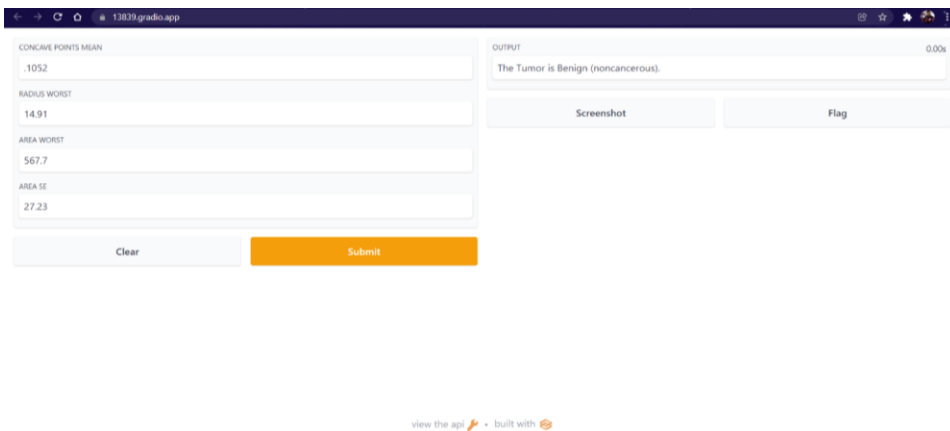
2. Code Snippet for Deploying the model using python's gradio library :

```
!pip install -q gradio

import gradio as gr

iface = gr.Interface(
    fn=predict,
    inputs=["number", "number", "number", "number"],
    outputs=["text"])
iface.launch()
```

3. Snapshot



The screenshot shows a web browser window at the address 13039.gradio.app. The interface has four input fields on the left: 'CONCAVE POINTS MEAN' with the value '.1052', 'RADIUS WORST' with '14.91', 'AREA WORST' with '567.7', and 'AREA SE' with '27.23'. Below these fields are 'Clear' and 'Submit' buttons. On the right, the 'OUTPUT' section shows the prediction 'The Tumor is Benign (noncancerous.)' with a '0.00s' timer. Below the output are 'Screenshot' and 'Flag' buttons. At the bottom of the page, there is a link 'view the api' and a note 'built with' followed by the Gradio logo.

Code for the Project:

Import Libraries

```

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Splitting Data
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler

# Modeling
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.tree import plot_tree

```

Import Dataset

```

cancer = pd.read_csv('../input/breast-cancer-wisconsin-data/data.csv')
cancer

```

```

cancer.info()

```

Data Cleaning

Drop Columns

```

cancer.drop(columns=['id', 'Unnamed: 32'], inplace = True)

```

Missing Value

```

cancer.isna().sum()/len(cancer.index)*100

```

Final Dataset

```

cancer

```

PreProcessing

If the cancer is Benign, it will be 0

If the cancer is Malignant, it will be 1

```
cancer['diagnosis'] = np.where(cancer['diagnosis'] == 'M', 1, 0)
cancer['diagnosis'].value_counts()/cancer.shape[0]*100
```

Data is imbalanced.

```
X = cancer.drop('diagnosis', axis = 1)
y = cancer['diagnosis']
```

```
robust = RobustScaler()
X_scaled = robust.fit_transform(X)
```

In the case of breast cancer, I want to reduce predictions to people who are misdiagnosed, diagnosed as benign, but it turns out to be malignant, that is, the person we predict is not the default (FN). Evaluation metrics used:

Recall

Data Splitting

X.shape

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
                                                    stratify = y,
                                                    test_size = 0.3,
                                                    random_state = 3030)
```

I use 0.3 as the default score for test_size and X.shape for random_state so the data will be divided equally.

Modeling

KNeighborsClassifier

```
k = range(1,100,2)
testing_accuracy = []
```

```

training_accuracy = []
score = 0

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(X_train, y_train)

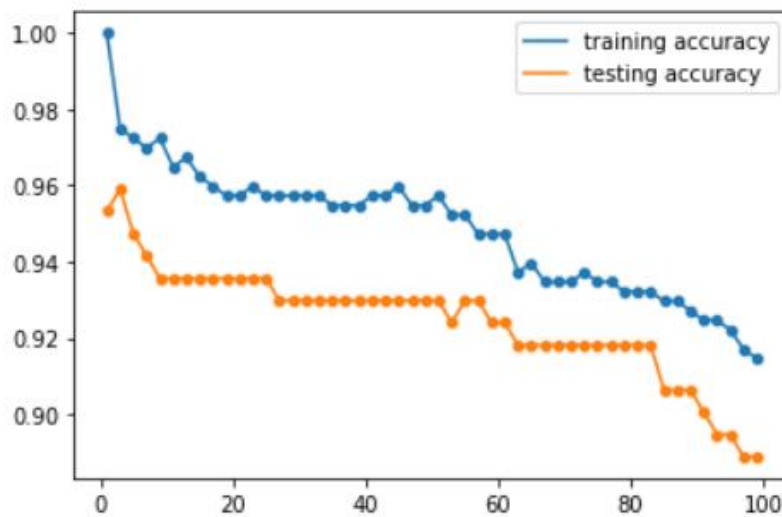
    y_predict_train = knn.predict(X_train)
    training_accuracy.append(accuracy_score(y_train, y_predict_train))

    y_predict_test = knn.predict(X_test)
    acc_score = accuracy_score(y_test, y_predict_test)
    testing_accuracy.append(acc_score)

    if score < acc_score:
        score = acc_score
        best_k = i

sns.lineplot(k, training_accuracy)
sns.scatterplot(k, training_accuracy)
sns.lineplot(k, testing_accuracy)
sns.scatterplot(k, testing_accuracy)
plt.legend(['training accuracy', 'testing accuracy'])

```

```
print("This is the best K for KNeighbors Classifier: ', best_k, '\nAccuracy score is: ', score)
```

This model indicates underfitting because training accuracy and testing accuracy are both decreasing.

Decision Tree Classifier

```
depth = range(1,25)
```

```
testing_accuracy = []
```

```
training_accuracy = []
```

```
score = 0
```

```
for i in depth:
```

```
    tree = DecisionTreeClassifier(max_depth = i, criterion = 'entropy')
```

```
    tree.fit(X_train, y_train)
```

```
    y_predict_train = tree.predict(X_train)
```

```
    training_accuracy.append(accuracy_score(y_train, y_predict_train))
```

```
    y_predict_test = tree.predict(X_test)
```

```
    acc_score = accuracy_score(y_test, y_predict_test)
```

```
    testing_accuracy.append(acc_score)
```

```

if score < acc_score:
    score = acc_score
    best_depth = i

sns.lineplot(depth, training_accuracy)
sns.scatterplot(depth, training_accuracy)
sns.lineplot(depth, testing_accuracy)
sns.scatterplot(depth, testing_accuracy)
plt.legend(['training accuracy', 'testing accuracy'])

print('This is the best depth for Decision Tree Classifier: ', best_depth, '\nAccuracy score is: ', score)

```

This model indicates **overfitting** because training accuracy is good and the testing accuracy is decreased.

Random Forest Classifier

```

estimator = range(5,100,5)
training_accuracy = []
testing_accuracy = []
score = 0
for i in estimator:
    forest = RandomForestClassifier(n_estimators = i, max_depth = 3)
    forest.fit(X_train, y_train)

    y_predict_train = forest.predict(X_train)
    training_accuracy.append(accuracy_score(y_train, y_predict_train))

    y_predict_test = forest.predict(X_test)
    acc_score = accuracy_score(y_test, y_predict_test)
    testing_accuracy.append(acc_score)

    if score < acc_score:
        score = acc_score
        best_estimator = i

sns.lineplot(estimator, training_accuracy)
sns.scatterplot(estimator, training_accuracy)

```

```
sns.lineplot(estimator, testing_accuracy)
sns.scatterplot(estimator, testing_accuracy)
plt.legend(['training accuracy', 'testing accuracy'])

print("This is the best Tree Count for Random Forest Classifier: ', best_estimator, '\nAccuracy score is: ', score)
```

Define Model

- We use **KNeighborsClassifier** with best K score , **Decision Tree Classifier** with best depth score and **Random Forest** with best estimator.

```
knn = KNeighborsClassifier(n_neighbors = 3)
tree = DecisionTreeClassifier(max_depth = 3, random_state = 3030)
forest = RandomForestClassifier(max_depth = 3, n_estimators= best_estimator)

def model_evaluation(model, metric):
    model_cv = cross_val_score(model, X_train, y_train, cv = StratifiedKFold(n_splits = 5), scoring = metric)
    return model_cv

knn_cv = model_evaluation(knn, 'recall')
tree_cv = model_evaluation(tree, 'recall')
forest_cv = model_evaluation(forest, 'recall')

for model in [knn, tree, forest]:
    model.fit(X_train, y_train)

score_cv = [knn_cv.round(5), tree_cv.round(5), forest_cv.round(5)]
score_mean = [knn_cv.mean(), tree_cv.mean(), forest_cv.mean()]
score_std = [knn_cv.std(), tree_cv.std(), forest_cv.std()]
score_recall_score = [recall_score(y_test, knn.predict(X_test)),
                      recall_score(y_test, tree.predict(X_test)),
                      recall_score(y_test, forest.predict(X_test))]
method_name = [ 'KNN Classifier', 'Decision Tree Classifier', 'Random Forest Classifier']
cv_summary = pd.DataFrame({
    'method': method_name,
    'cv score': score_cv,
    'mean score': score_mean,
    'std score': score_std,
    'recall score': score_recall_score
})
cv_summary
```

From the cross validation and model evaluation processes, We decided to continue with **Decision Tree Classifier** even the score is indicated overfitting. Let's tune the model.

HyperParam Tuning

```
tree = DecisionTreeClassifier(max_depth = 3, random_state = 3030)
```

```
hyperparam_space = {  
    'criterion': ['gini', 'entropy'],  
    'splitter': ['best', 'random'],  
    'max_depth': [3, 5, 7, 9, 11],  
    'min_samples_leaf': [3, 9, 13, 15, 17],  
    'class_weight': ['list', 'dict', 'balanced'],  
    'random_state': [3030]  
}
```

```
grid = GridSearchCV(  
    tree,  
    param_grid = hyperparam_space,  
    cv = StratifiedKFold(n_splits = 5),  
    scoring = 'recall',  
    n_jobs = -1)
```

```
grid.fit(X_train, y_train)
```

```
print('best score', grid.best_score_)  
print('best param', grid.best_params_)
```

Comparison Between Before & After Tuning

```
tree.fit(X_train, y_train)  
tree_recall = (recall_score(y_test, tree.predict(X_test)))
```

```
grid.best_estimator_.fit(X_train, y_train)  
grid_recall = (recall_score(y_test, grid.predict(X_test)))
```

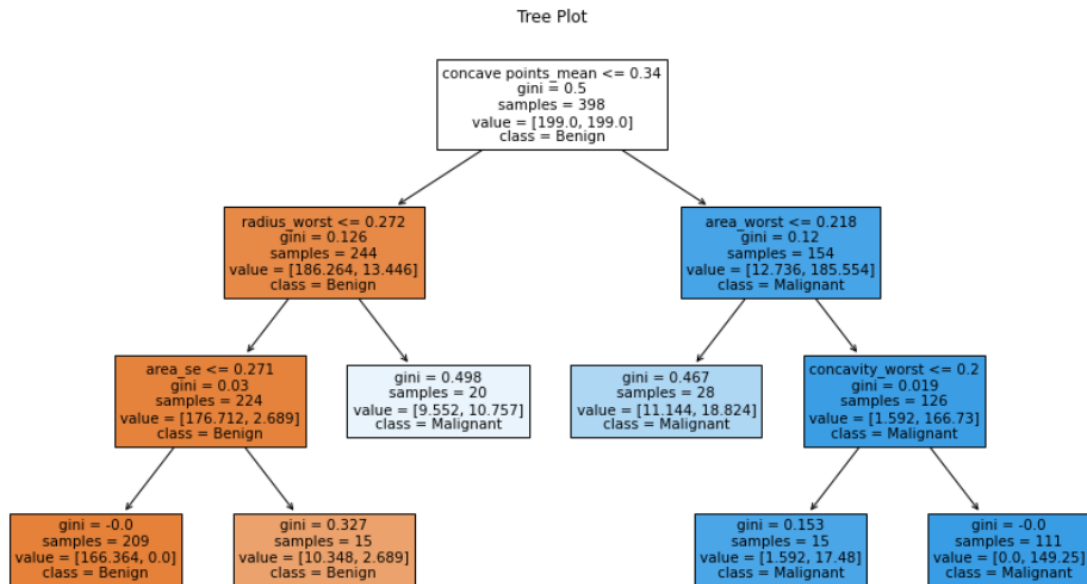
```
score_list = [tree_recall, grid_recall]  
method_name = ['Decision Tree Classifier Before Tuning', 'Decision Tree Classifier After Tuning']  
best_summary = pd.DataFrame({  
    'method': method_name,  
    'score': score_list  
})
```

best_summary

This is the comparison between before tuning score and after tuning score using Decision Tree Classifier. I choose to use the Decision Tree Classifier after tuning score in this section.

Decision Tree Classifier Plot

```
plt.figure(figsize=(15,8))
plot_tree(grid.best_estimator_, feature_names = list(X), class_names = ['Benign','Malignant'], filled =
True)
plt.title('Tree Plot')
plt.show()
```



Feature Importance

```
importance_table = pd.DataFrame({
    'imp': grid.best_estimator_.feature_importances_
}, index = X.columns)
importance_table.sort_values('imp', ascending = False)
```

```
importance_table.sort_values('imp', ascending = True).plot(kind = 'barh', figsize = (15,8))
```

The results suggest perhaps 4 of the 30 features as being important to prediction.

Deploying the Final Tree Model for Production

```
X = cancer[['concave points_mean', 'radius_worst', 'area_worst', 'area_se']]
y = cancer['diagnosis']
```

```
robust = RobustScaler()
X_scaled = robust.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
                                                    stratify = y,
                                                    test_size = 0.3,
                                                    random_state = 3030)
grid.fit(X_train, y_train)
```

Chapter 7 - Test Results (Prediction with performance measures)

- We use KNeighborsClassifier with best K score and Decision Tree Classifier with best depth score.

```
knn = KNeighborsClassifier(n_neighbors = 3)
```

```
tree = DecisionTreeClassifier(max_depth = 3, random_state = 3030)
```

```
def model_evaluation(model, metric):
```

```
    model_cv = cross_val_score(model, X_train, y_train, cv = StratifiedKFold(n_splits = 5), scoring = metric)
```

```
    return model_cv
```

```
knn_cv = model_evaluation(knn, 'recall')
```

```
tree_cv = model_evaluation(tree, 'recall')
```

```
for model in [knn, tree]:
```

```
    model.fit(X_train, y_train)
```

```
score_cv = [knn_cv.round(5), tree_cv.round(5)]
```

```
score_mean = [knn_cv.mean(), tree_cv.mean()]
```

```
score_std = [knn_cv.std(), tree_cv.std()]
```

```
score_recall_score = [recall_score(y_test, knn.predict(X_test)),
```

```
                      recall_score(y_test, tree.predict(X_test))]
```

```
method_name = [ 'KNN Classifier', 'Decision Tree Classifier']
```

```
cv_summary = pd.DataFrame({
```

```
    'method': method_name,
```

```
    'cv score': score_cv,
```

```
    'mean score': score_mean,
```

```

    'std score': score_std,
    'recall score': score_recall_score
})
cv_summary

```

	method	cv score	mean score	std score	recall score
0	KNN Classifier	[0.83333, 0.93333, 0.96667, 0.93103, 0.86207]	0.905287	0.049522	0.890625
1	Decision Tree Classifier	[0.9, 0.93333, 0.73333, 0.93103, 0.86207]	0.871954	0.073970	0.921875

From the cross validation and model evaluation processes, I decide to continue with Decision Tree Classifier even if the score is indicated overfitting. Let's tune the model.

Comparison Between Before & After Tuning

```

tree.fit(X_train, y_train)
tree_recall = (recall_score(y_test, tree.predict(X_test)))

grid.best_estimator_.fit(X_train, y_train)
grid_recall = (recall_score(y_test, grid.predict(X_test)))

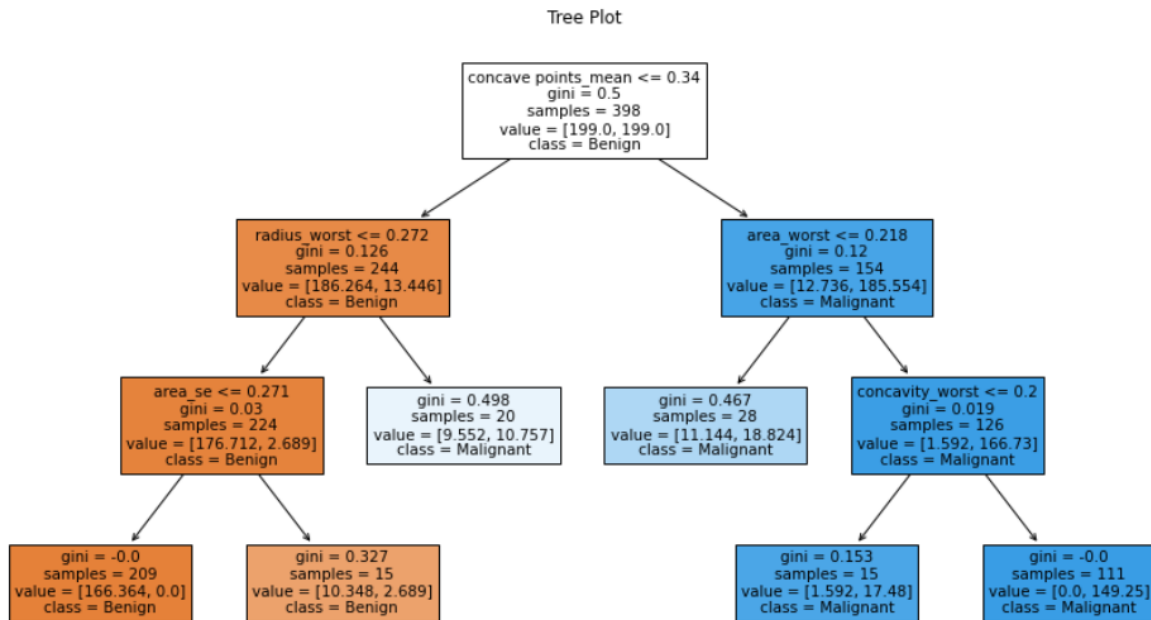
score_list = [tree_recall, grid_recall]
method_name = ['Decision Tree Classifier Before Tuning', 'Decision Tree Classifier After Tuning']
best_summary = pd.DataFrame({
    'method': method_name,
    'score': score_list
})
best_summary

```

	method	score
0	Decision Tree Classifier Before Tuning	0.921875
1	Decision Tree Classifier After Tuning	0.937500

This is the comparison between before tuning score and after tuning score using Decision Tree Classifier. I choose to use the **Decision Tree Classifier after tuning** score in this section.

Decision Tree Classifier Plot



Chapter 8 - Conclusion

- In the first step, We did scaling at X data using Robust Scaler because we believe there are so many outliers.
- We only use the KNeighbor Classifier (KNN) and the Decision Tree Classifier (Tree) in this prediction. We try to find the best K score and best depth for each model and see how the training and testing data works on both models.
- From the cross-validation process, the KNN model has the highest score with 0.9 but after model evaluation using recall metric, the Tree model has the highest score with 0.92. Even though the Tree model is indicated overfitting, We still choose to use this score to continue the process.
- We decide to get the best parameter for the Tree model by Tuning with the best score of 0.95 which is increasing, then compare the Tree model score before and after tuning. The comparison results prove that the Tree model after the Tuning process is higher than before with 0.9375.
- We check again to see the data using the Feature Importance process. Surprisingly, out of 30 features (columns), only 4 features were important to prediction.
- We Trained the Tree model with only those features and Deployed the final Tree model onto the web using Python's Library *graido*.

Future Enhancements:

- To increase the efficiency of the other two classifiers i.e Random Forest Classifier and KNN Classifier.
- And to deploy all the three models onto the web for the production.

Chapter 9 - References

Research Papers

Research Paper	Link for the Paper	Year
Machine Learning Algorithms For Breast Cancer Prediction And Diagnosis	https://www.researchgate.net/publication/352572400_Machine_Learning_Algorithms_For_Breast_Cancer_Prediction_And_Diagnosis	2021
Breast Cancer Detection Using K-Nearest Neighbors Algorithm -	https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3361553	2018
Breast Cancer Prediction using Random Forest Classifier	https://aip.scitation.org/doi/pdf/10.1063/1.5132477	2019

Other References

- Stack Overflow: <https://stackoverflow.com>
- Scikit learn: <https://scikit-learn.org>
- Kaggle - Machine learning and Data science community: <https://www.kaggle.com>
- GeeksforGeeks: <https://www.geeksforgeeks.org>

Text-Book

- *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow_Concepts, Tools, and Techniques to Build Intelligent Systems-O'Reilly Media (2019)*

