Breast Cancer Classification Using Machine Learning and Deep Learning

Name:OjaswI Kumar, Roll number:102296001
Name:Nitish Kumar,Roll number:102166003

Title: Breast Cancer Classification using Machine Learning and Deep Learning Techniques

Objective:

The primary objective of this project is to develop a robust and accurate system for the early detection of breast cancer. By leveraging machine learning and deep learning algorithms, we aim to classify breast tumors as malignant or benign based on various features extracted from medical images.

Machine Learning Classifiers Used:

Logistic Regression
K Neighbors Classifier (KNN)
Support Vector Classifier (SVC)
SGD Classifier
Decision Tree Classifier
Random Forest Classifier
Voting Classifier
Ada Boost Classifier
Gradient Boosting Classifier
Stochastic Gradient Boosting (SGB)
Extreme Gradient Boosting (XGBoost)

Results:

The classification models were trained and evaluated on a dataset of breast cancer samples. The performance metrics, including accuracy, precision, recall, and F1-score, were computed for each classifier. Notably, the Gradient Boosting Classifier achieved the highest accuracy of 97.66%, making it the most effective model for our breast cancer classification task.

Deep Learning Approach:

In addition to traditional machine learning methods, we explored the effectiveness of deep learning using a neural network. The neural network achieved an accuracy of 95.6% in classifying breast cancer samples.

Conclusion:

The promising results obtained from both machine learning and deep learning approaches demonstrate the potential of these techniques in accurately identifying breast cancer. The high accuracy achieved by the Gradient Boosting Classifier suggests its effectiveness in clinical applications for early cancer diagnosis.t

Breast Cancer Classification

Attribute Information:

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

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

radius (mean of distances from center to points on the perimeter)

texture (standard deviation of gray-scale values)

perimeter

area

smoothness (local variation in radius lengths)

compactness (perimeter^2 / area - 1.0)

concavity (severity of concave portions of the contour)

concave points (number of concave portions of the contour)

symmetry

fractal dimension ("coastline approximation" - 1)

Importing libraries

```
# Importing libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno

import warnings
warnings.filterwarnings('ignore')


plt.style.use('ggplot')
```

```
df = pd.read_csv('data.csv')
```

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	poin
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	

5 rows × 33 columns

Data Preprocessing

```
df.drop(['id', 'Unnamed: 32'], axis = 1, inplace = True)
```

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

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

```
df['diagnosis'] = df['diagnosis'].apply(lambda val: 1 if val == 'M' else 0)
```

```
df.head()
```

	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

5 rows × 31 columns

```
df.describe()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.00
mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.09
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.01
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.05
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.08
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.09
75%	1.000000	15.780000	21.800000	104.100000	782.700000	0.10
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.16

8 rows × 31 columns

```
df.info()
```

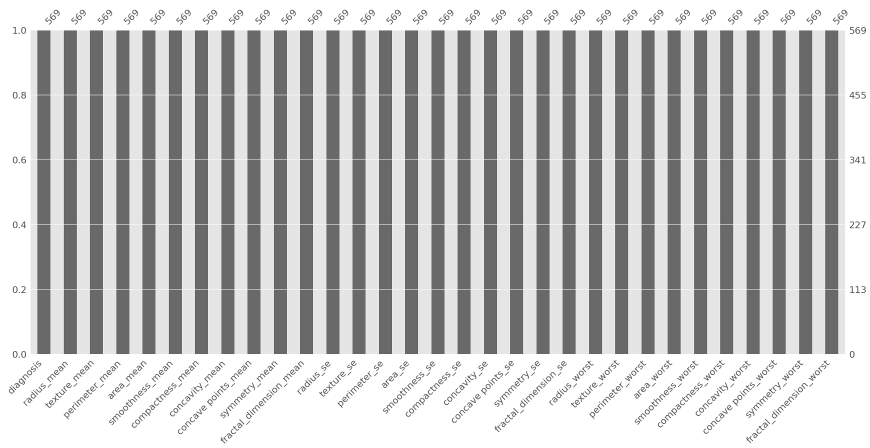
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #  Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   diagnosis               569 non-null    int64
 1  radius_mean              569 non-null    float64
 2  texture_mean             569 non-null    float64
 3   perimeter_mean          569 non-null  569  float64
 4  area_mean                non-null        float64
 5   smoothness_mean         569 non-null    float64
6    compactness_mean        569 non-null    float64
 7   concavity_mean          569 non-null    float64
8    concave points_mean     569 non-null    float64
 9   symmetry_mean           569 non-null  569  float64
 10  fractal_dimension_mean  non-null        float64
radius_se   12              569 non-null    float64
texture_se                  569 non-null  569  float64
 13  perimeter_se            non-null        float64
 14  area_se                 569 non-null    float64
 15  smoothness_se           569 non-null    float64
 16  compactness_se          569 non-null    float64
17   concavity_se            569 non-null    float64
 18  concave points_se       569 non-null    float64
 19 symmetry_se  20 569 non-null   569       float64
fractal_dimension_se                        float64
 21  radius_worst  texture_worst 569 569 non-null  float64
 22
23   perimeter_worst         569 non-null    float64
 24  area_worst              569 non-null    float64
 25  smoothness_worst        569 non-null    float64
 26  compactness_worst       569 non-null    float64
 27  concavity_worst         569 non-null    float64
 28  concave points_worst    569 non-null    float64
 29  symmetry_worst          569 non-null    float64
 30  fractal_dimension_worst 569 non-null    float64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

```
# checking for null values

df.isna().sum()
```

```
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

```
# visualizing null values

msno.bar(df)
```

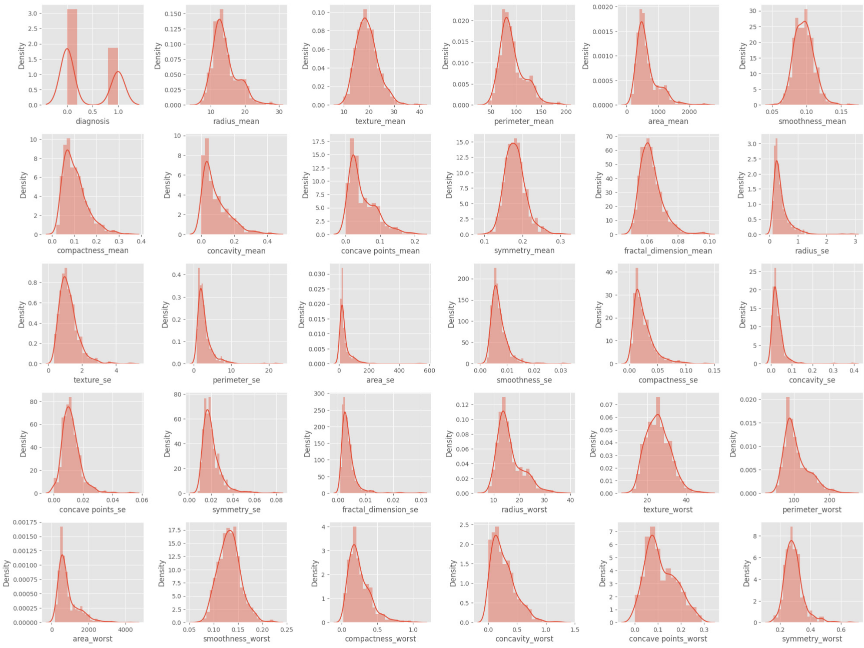
```
<Axes: >
```



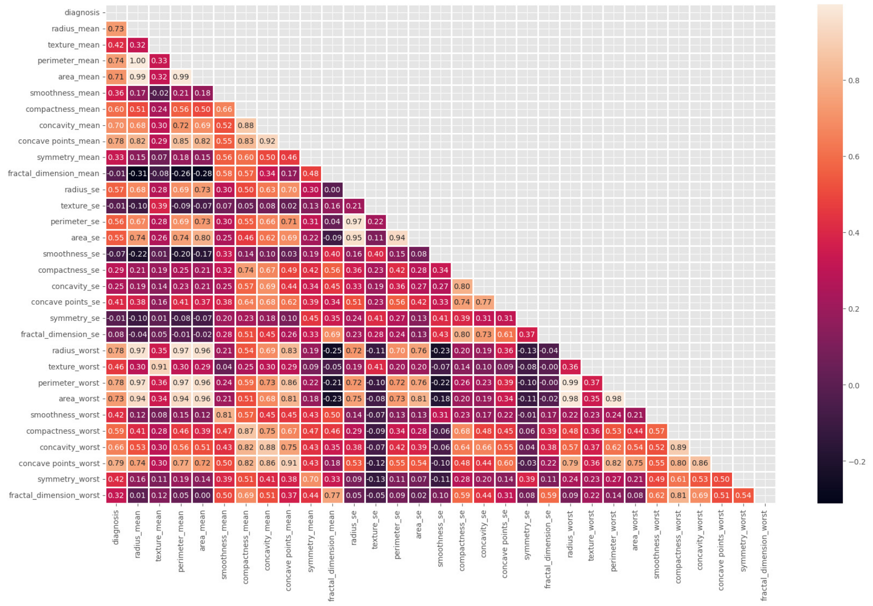
There are no missing values in the data.

Exploratory Data Analysis (EDA)

```
plt.figure(figsize = (20, 15))
plotnumber = 1

for column in df:
    if plotnumber <= 30:
        ax = plt.subplot(5, 6, plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column)

    plotnumber += 1

plt.tight_layout()
plt.show()
```



```
# heatmap

plt.figure(figsize = (20, 12))

corr = df.corr()
mask = np.triu(np.ones_like(corr, dtype = bool))

sns.heatmap(corr, mask = mask, linewidths = 1, annot = True, fmt = ".2f")
plt.show()
```



We can see that there are many columns which are very highly correlated which causes multicollinearity so we have to remove highly correlated features.

```
# removing highly correlated features

corr_matrix = df.corr().abs()

mask = np.triu(np.ones_like(corr_matrix, dtype = bool))
tri_df = corr_matrix.mask(mask)

to_drop = [x for x in tri_df.columns if any(tri_df[x] > 0.92)]

df = df.drop(to_drop, axis = 1)

print(f"The reduced dataframe has {df.shape[1]} columns.")
```

```
The reduced dataframe has 23 columns.
```

```
# creating features and label

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

```
# splitting data into training and test set

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 0)
```

```
# scaling data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Logistic Regression

```
# fitting data to model

from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
# model predictions

y_pred = log_reg.predict(X_test)
```

```
# accuracy score

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

print(accuracy_score(y_train, log_reg.predict(X_train)))

log_reg_acc = accuracy_score(y_test, log_reg.predict(X_test))
print(log_reg_acc)
```

```
0.9899497487437185
0.9590643274853801
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[106  2] [
5  58]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```

```
           0       0.95      0.98      0.97       108
           1       0.97      0.92      0.94        63

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171
```

K Neighbors Classifier (KNN)

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
# model predictions

y_pred = knn.predict(X_test)
```

```
# accuracy score

print(accuracy_score(y_train, knn.predict(X_train)))

knn_acc = accuracy_score(y_test, knn.predict(X_test))
print(knn_acc)
```

```
0.9623115577889447
0.935672514619883
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[105  3] [
8  55]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93  0.97           0.95       108
           1       0.95      0.87      0.91        63

    accuracy                           0.94       171
   macro avg       0.94      0.92      0.93       171
weighted avg       0.94      0.94      0.94       171
```

Support Vector Classifier (SVC)

```
from sklearn.svm import SVC from
sklearn.model_selection import GridSearchCV

svc = SVC()
parameters = {
    'gamma' : [0.0001, 0.001, 0.01, 0.1],
'C' : [0.01, 0.05, 0.5, 0.1, 1, 10, 15, 20] }


grid_search = GridSearchCV(svc, parameters)
grid_search.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

```
# best parameters

grid_search.best_params_
```

```
{'C': 10, 'gamma': 0.01}
```

```
# best accuracy

grid_search.best_score_
```

```
0.9774683544303798
```

```
svc = SVC(C = 10, gamma = 0.01)
svc.fit(X_train, y_train)
```

```
▾ SVC SVC(C=10,
gamma=0.01)
```

```
# model predictions

y_pred = svc.predict(X_test)
```

```
# accuracy score

print(accuracy_score(y_train, svc.predict(X_train)))

svc_acc = accuracy_score(y_test, svc.predict(X_test))
print(svc_acc)
```

```
0.9874371859296482
0.9766081871345029
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[107  1] [
3  60]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       108
           1       0.98      0.95      0.97        63

    accuracy                           0.98       171
   macro avg       0.98      0.97      0.97       171
weighted avg       0.98      0.98      0.98       171
```

SGD Classifier

```
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier()
parameters = {
    'alpha' : [0.0001, 0.001, 0.01, 0.1, 1],
    'loss' : ['hinge', 'log'],
    'penalty' : ['l1', 'l2']
}

grid_search = GridSearchCV(sgd, parameters, cv = 10, n_jobs = -1)
grid_search.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: SGDClassifier
    ▸ SGDClassifier
```

```
# best parameter

grid_search.best_params_
```

```
{'alpha': 0.001, 'loss': 'log', 'penalty': 'l2'}
```

```
sgd = SGDClassifier(alpha = 0.001, loss = 'log', penalty = 'l2')
sgd.fit(X_train, y_train)
```

```
• SGDClassifier
SGDClassifier(alpha=0.001, loss='log')
```

```
# model predictions

y_pred = sgd.predict(X_test)
```

```
# accuracy score

print(accuracy_score(y_train, sgd.predict(X_train)))

sgd_acc = accuracy_score(y_test, sgd.predict(X_test))
print(sgd_acc)
```

```
0.9899497487437185
0.9766081871345029
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[106  2] [
2  61]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       108
           1       0.97      0.97      0.97        63

    accuracy                           0.98       171
   macro avg       0.97  0.97          0.97       171
weighted avg       0.98      0.98      0.98       171
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()

parameters = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : range(2, 32, 1),
    'min_samples_leaf' : range(1, 10, 1),
    'min_samples_split' : range(2, 10, 1),
    'splitter' : ['best', 'random']
}

grid_search_dt = GridSearchCV(dtc, parameters, cv = 5, n_jobs = -1, verbose = 1)
grid_search_dt.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 8640 candidates, totalling 43200 fits
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 32),
                         'min_samples_leaf': range(1, 10),
                         'min_samples_split': range(2, 10),
                         'splitter': ['best', 'random']},
             verbose=1)
```

```
▸ GridSearchCV
```

```
# best parameters

grid_search_dt.best_params_
```

```
{'criterion': 'entropy',
 'max_depth': 23,
 'min_samples_leaf': 5,
 'min_samples_split': 3,
 'splitter': 'random'}
```

```
# best score

grid_search_dt.best_score_
```

```
0.9572784810126581
```

```
dtc = DecisionTreeClassifier(criterion = 'entropy', max_depth = 28, min_samples_leaf = 1, min_samples_split = 8, splitter = 'random')
dtc.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=28, min_samples_split=8,
splitter='random')
```

```
y_pred = dtc.predict(X_test)
```

```
# accuracy score

print(accuracy_score(y_train, dtc.predict(X_train)))

dtc_acc = accuracy_score(y_test, dtc.predict(X_test))
print(dtc_acc)
```

```
0.9798994974874372
0.9298245614035088
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[104  4] [
8  55]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0  0.93           0.96      0.95       108
           1        0.93      0.87      0.90        63

    accuracy                           0.93       171
   macro avg       0.93  0.92          0.92       171
weighted avg       0.93  0.93          0.93       171
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

rand_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11, max_features = 'auto', min_samples_leaf = 2, min_samples_split
rand_clf.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=11, max_features='auto',
min_samples_leaf=2, min_samples_split=3,                       n_estimators=130)
```

```
y_pred = rand_clf.predict(X_test)
```

```
# accuracy score

print(accuracy_score(y_train, rand_clf.predict(X_train)))

ran_clf_acc = accuracy_score(y_test, y_pred)
print(ran_clf_acc)
```

```
0.9974874371859297
0.9590643274853801
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

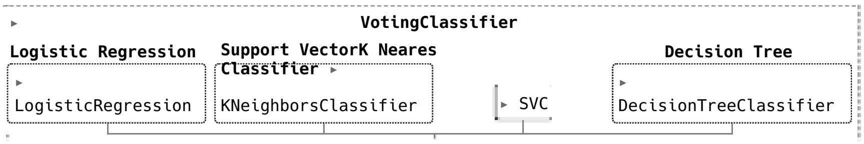
```
[[107  1] [
6  57]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0  0.95  0.99              0.97       108
           1  0.98           0.90      0.94        63

    accuracy                          0.96       171
   macro avg       0.96  0.95         0.96       171
weighted avg       0.96  0.96         0.96       171
```

Voting Classifier

```
from sklearn.ensemble import VotingClassifier

classifiers = [('Logistic Regression', log_reg), ('K Nearest Neighbours', knn), ('Support Vector Classifier', svc),
('Decision Tree', dtc)]

vc = VotingClassifier(estimators = classifiers)

vc.fit(X_train, y_train)
```




```
y_pred = vc.predict(X_test)
```

```
# accuracy score

print(accuracy_score(y_train, vc.predict(X_train)))

vc_acc = accuracy_score(y_test, y_pred)
print(vc_acc)
```

```
0.9874371859296482
0.9649122807017544
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[108  0] [
6  57]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       108
           1       1.00      0.90      0.95        63

    accuracy                           0.96       171
   macro avg       0.97      0.95      0.96       171
```

```
weighted avg       0.97      0.96      0.96       171
```

Ada Boost Classifier

```
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(base_estimator = dtc)

ada = AdaBoostClassifier(dtc, n_estimators = 180)
ada.fit(X_train, y_train)
```

```
▸ AdaBoostClassifier
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
y_pred = ada.predict(X_test)
```

```
# accuracy score

print(accuracy_score(y_train, ada.predict(X_train)))

ada_acc = accuracy_score(y_test, y_pred)
print(ada_acc)
```

```
1.0
0.9766081871345029
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[108  0] [
4  59]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0  0.96           1.00      0.98       108
           1       1.00      0.94      0.97        63

    accuracy                           0.98       171
   macro avg       0.98      0.97      0.97       171
weighted avg       0.98  0.98          0.98       171
```

Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier()

parameters = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': [0.001, 0.1, 1, 10],
    'n_estimators': [100, 150, 180, 200]
}

grid_search_gbc = GridSearchCV(gbc, parameters, cv = 5, n_jobs = -1, verbose = 1)
grid_search_gbc.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
▸ GridSearchCV
▸ estimator: GradientBoostingClassifier
  ▸ GradientBoostingClassifier
```

```
# best parameters

grid_search_gbc.best_params_
```

```
{'learning_rate': 1, 'loss': 'exponential', 'n_estimators': 100}
```

```
# best score

grid_search_gbc.best_score_
```

```
0.9673417721518988
```

```
gbc = GradientBoostingClassifier(learning_rate = 1, loss = 'exponential', n_estimators = 200)
gbc.fit(X_train, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=1, loss='exponential',
n_estimators=200)
```

```
y_pred = gbc.predict(X_test)
```

```
# accuracy score

print(accuracy_score(y_train, gbc.predict(X_train)))

gbc_acc = accuracy_score(y_test, y_pred)
print(gbc_acc)
```

```
1 0.976608187134502
  9
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[106  2]
 [  2  61]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       108
1                  0.97      0.97      0.97        63

    accuracy                           0.98       171
   macro avg       0.97  0.97          0.97       171
weighted avg       0.98  0.98          0.98       171
```

Stochastic Gradient Boosting (SGB)

```
sgbc = GradientBoostingClassifier(max_depth=4, subsample=0.9, max_features=0.75, n_estimators=200, random_state=0)

sgbc.fit(X_train, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(max_depth=4, max_features=0.75, n_estimators=200,
random_state=0, subsample=0.9)
```

```
y_pred = sgbc.predict(X_test)
```

```
# accuracy score

print(accuracy_score(y_train, sgbc.predict(X_train)))

sgbc_acc = accuracy_score(y_test, y_pred)
print(sgbc_acc)
```

```
1.0
0.9590643274853801
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[104  4] [
3  60]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       108
           1       0.94  0.95          0.94        63

    accuracy                           0.96       171
   macro avg       0.95      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

Extreme Gradient Boosting

```
from xgboost import XGBClassifier

xgb = XGBClassifier(objective = 'binary:logistic', learning_rate = 0.5, max_depth = 5, n_estimators = 180)

xgb.fit(X_train, y_train)
```

```
▾                                  XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
colsample_bytree=None, device=None, early_stopping_rounds=None,
enable_categorical=False, eval_metric=None, feature_types=None,
gamma=None, grow_policy=None, importance_type=None,
interaction_constraints=None, learning_rate=0.5, max_bin=None,
max_cat_threshold=None, max_cat_to_onehot=None,
max_delta_step=None, max_depth=5, max_leaves=None,
min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=180, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
y_pred = xgb.predict(X_test)
```

```
# accuracy score

print(accuracy_score(y_train, xgb.predict(X_train)))

xgb_acc = accuracy_score(y_test, y_pred)
print(xgb_acc)
```

```
1.0
0.9649122807017544
```

```
# confusion matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[105  3]
 [  3  60]]
```

```
# classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       108
1                  0.95      0.95      0.95        63

    accuracy                           0.96       171
   macro avg       0.96      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

FINAL SCORE COMPARISION OF ALL MODELS USED

```
models = pd.DataFrame({
    'Model': ['Logistic Regression', 'KNN', 'SVC', 'SGD Classifier', 'Decision Tree Classifier', 'Random Forest Classifier', 'Voting Cla
    'Gradient Boosting Classifier', 'Stochastic Gradient Boosting', 'XgBoost'],
    'Score': [log_reg_acc, knn_acc, svc_acc, sgd_acc, dtc_acc, ran_clf_acc, vc_acc, ada_acc, gbc_acc, sgbc_acc, xgb_acc]
})

models.sort_values(by = 'Score', ascending = False)
```

	Model	Score
2	SVC	0.976608
3	SGD Classifier	0.976608
7	Ada Boost Classifier	0.976608
8	Gradient Boosting Classifier	0.976608
6	Voting Classifier	0.964912
10	XgBoost	0.964912
0	Logistic Regression	0.959064
5	Random Forest Classifier	0.959064
9	Stochastic Gradient Boosting	0.959064
1	KNN	0.935673
4	Decision Tree Classifier	0.929825

Best model for diagnosing breast cancer is "Gradient Boosting Classifier" with an accuracy of 97.6608%

Breast Cancer Classification with a simple Neural Network (NN)

Importing the Dependencies

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.datasets
from sklearn.model_selection import train_test_split
```

Data Collection & Processing

```
# loading the data from sklearn
breast_cancer_dataset = sklearn.datasets.load_breast_cancer()
```

```
print(breast_cancer_dataset)
```


```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]), 'frame': None, 'target_names': array(['malignant', 'benign'], dtyp
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'], dtype='<U23'), 'filename': 'breast_cancer.csv', 'data_module': 'sklearn.dataset
```


```
# loading the data to a data frame
data_frame = pd.DataFrame(breast_cancer_dataset.data, columns = breast_cancer_dataset.feature_names)
```

```
# print the first 5 rows of the dataframe
data_frame.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	wor perimet
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152

```
# adding the 'target' column to the data frame
data_frame['label'] = breast_cancer_dataset.target
```

```
# print last 5 rows of the dataframe
data_frame.tail()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	w
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	20
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	17
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	18
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	2

5 rows × 31 columns

```
# number of rows and columns in the dataset
data_frame.shape
```

```
(569, 31)
```

```
# getting some information about the data
data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  label                    569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

```
# checking for missing values
data_frame.isnull().sum()
```

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
```

```
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
label                      0
dtype: int64
```

```
# statistical measures about the data
data_frame.describe()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	56
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	2
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	1
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	2
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	2
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	2
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	4

8 rows × 31 columns

```
# checking the distribution of Target Varibale
data_frame['label'].value_counts()
```

```
1    357
0    212
Name: label, dtype: int64
```

1 --> Benign

0 --> Malignant

```
data_frame.groupby('label').mean()
```

label	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius
0	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775	0.087990	0.192909	0.062680	...	21.134811
1	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058	0.025717	0.174186	0.062867	...	13.379801

2 rows × 30 columns

Separating the features and target

```
X = data_frame.drop(columns='label', axis=1)
Y = data_frame['label']
```

```
print(X)
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
1          20.57         17.77          132.90     1326.0          0.08474
2          19.69         21.25          130.00     1203.0          0.10960
3          11.42         20.38           77.58      386.1          0.14250
4          20.29         14.34          135.10     1297.0          0.10030
..           ...           ...             ...        ...              ...
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.28390         0.24140              0.10520         0.2597
4             0.13280         0.19800              0.10430         0.1809
..                ...             ...                  ...            ...
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  ...  worst radius  worst texture  \
0                   0.07871  ...        25.380          17.33
1                   0.05667  ...        24.990          23.41
2                   0.05999  ...        23.570          25.53
3                   0.09744  ...        14.910          26.50
4                   0.05883  ...        22.540          16.67
..                      ...  ...           ...            ...
564                 0.05623  ...        25.450          26.40
565                 0.05533  ...        23.690          38.25
566                 0.05648  ...        18.980          34.12
567                 0.07016  ...        25.740          39.42
568                 0.05884  ...         9.456          30.37

     worst perimeter  worst area  worst smoothness  worst compactness  \
0             184.60      2019.0           0.16220            0.66560
1             158.80      1956.0           0.12380            0.18660
2             152.50      1709.0           0.14440            0.42450
3              98.87       567.7           0.20980            0.86630
4             152.20      1575.0           0.13740            0.20500
..               ...         ...               ...                ...
564           166.10      2027.0           0.14100            0.21130
565           155.00      1731.0           0.11660            0.19220
566           126.70      1124.0           0.11390            0.30940
567           184.60      1821.0           0.16500            0.86810
568            59.16       268.6           0.08996            0.06444

     worst concavity  worst concave points  worst symmetry  \
0             0.7119                0.2654          0.4601
1             0.2416                0.1860          0.2750
2             0.4504                0.2430          0.3613
3             0.6869                0.2575          0.6638
4             0.4000                0.1625          0.2364
```

```
print(Y)
```

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: label, Length: 569, dtype: int64
```

Splitting the data into training data & Testing data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

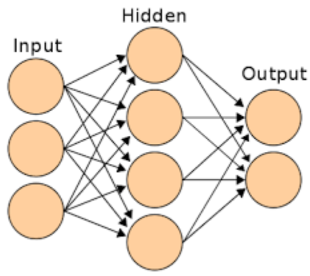
```
(569, 30) (455, 30) (114, 30)
```

Standardize the data

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()

X_train_std = scaler.fit_transform(X_train)

X_test_std = scaler.transform(X_test)
```

Building the Neural Network




```
# importing tensorflow and Keras
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
```

```
# setting up the layers of Neural Network

model = keras.Sequential([
                          keras.layers.Flatten(input_shape=(30,)),
                          keras.layers.Dense(20, activation='relu'),
                          keras.layers.Dense(2, activation='sigmoid')
])
```

```
# compiling the Neural Network

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

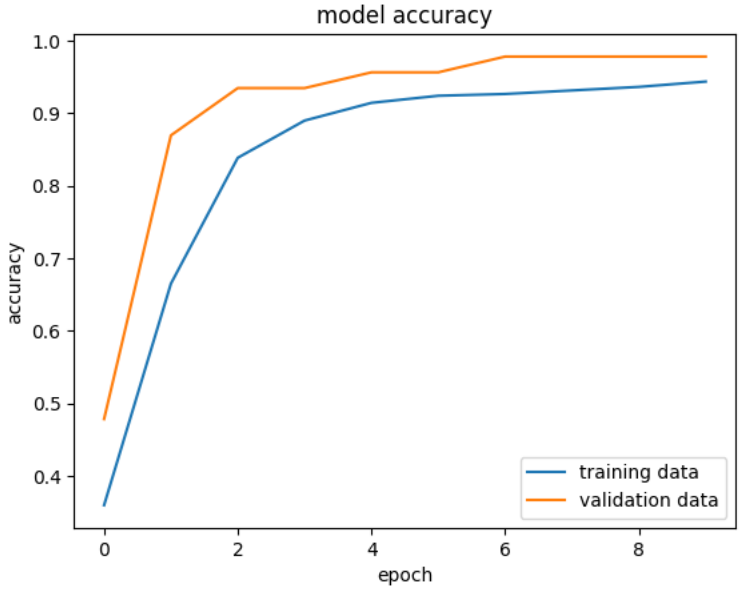
```
# training the Meural Network

history = model.fit(X_train_std, Y_train, validation_split=0.1, epochs=10)
```

```
Epoch 1/10
13/13 [==============================] - 1s 33ms/step - loss: 0.8601 - accuracy: 0.3594 - val_loss: 0.7031 - val_accuracy: 0.4783
Epoch 2/10
13/13 [==============================] - 0s 9ms/step - loss: 0.5978 - accuracy: 0.6650 - val_loss: 0.5059 - val_accuracy: 0.8696
Epoch 3/10
13/13 [==============================] - 0s 10ms/step - loss: 0.4534 - accuracy: 0.8386 - val_loss: 0.3922 - val_accuracy: 0.9348
Epoch 4/10
13/13 [==============================] - 0s 10ms/step - loss: 0.3660 - accuracy: 0.8900 - val_loss: 0.3151 - val_accuracy: 0.9348
Epoch 5/10
13/13 [==============================] - 0s 10ms/step - loss: 0.3023 - accuracy: 0.9144 - val_loss: 0.2605 - val_accuracy: 0.9565
Epoch 6/10
13/13 [==============================] - 0s 9ms/step - loss: 0.2579 - accuracy: 0.9242 - val_loss: 0.2210 - val_accuracy: 0.9565
Epoch 7/10
13/13 [==============================] - 0s 8ms/step - loss: 0.2236 - accuracy: 0.9267 - val_loss: 0.1928 - val_accuracy: 0.9783
Epoch 8/10
13/13 [==============================] - 0s 8ms/step - loss: 0.1983 - accuracy: 0.9315 - val_loss: 0.1730 - val_accuracy: 0.9783
Epoch 9/10
13/13 [==============================] - 0s 9ms/step - loss: 0.1799 - accuracy: 0.9364 - val_loss: 0.1572 - val_accuracy: 0.9783
Epoch 10/10
13/13 [==============================] - 0s 8ms/step - loss: 0.1638 - accuracy: 0.9438 - val_loss: 0.1460 - val_accuracy: 0.9783
```

Visualizing accuracy and loss

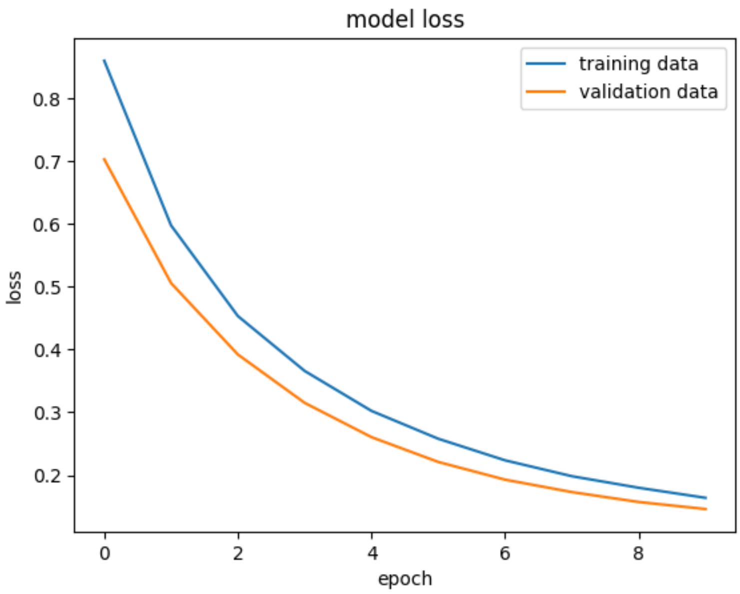
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')

plt.legend(['training data', 'validation data'], loc = 'lower right')
```

```
<matplotlib.legend.Legend at 0x7ba376b0be50>
```




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')

plt.legend(['training data', 'validation data'], loc = 'upper right')
```

```
<matplotlib.legend.Legend at 0x7ba3863745b0>
```



Accuracy of the model on test data

```
loss, accuracy = model.evaluate(X_test_std, Y_test)
print(accuracy)
```

```
4/4 [==============================] - 0s 4ms/step - loss: 0.1335 - accuracy: 0.9561
0.9561403393745422
```

```
print(X_test_std.shape)
print(X_test_std[0])
```

```
(114, 30)
[-0.04462793 -1.41612656 -0.05903514 -0.16234067  2.0202457  -0.11323672
  0.18500609  0.47102419  0.63336386  0.26335737  0.53209124  2.62763999
  0.62351167  0.11405261  1.01246781  0.41126289  0.63848593  2.88971815
 -0.41675911  0.74270853 -0.32983699 -1.67435595 -0.36854552 -0.38767294
  0.32655007 -0.74858917 -0.54689089 -0.18278004 -1.23064515 -0.6268286 ]
```

```
Y_pred = model.predict(X_test_std)
```

```
4/4 [==============================] - 0s 3ms/step
```

```
print(Y_pred.shape)
print(Y_pred[0])
```

```
(114, 2)
[0.20836309 0.87977034]
```

```
print(X_test_std)
```

```
[[-0.04462793 -1.41612656 -0.05903514 ... -0.18278004 -1.23064515
  -0.6268286 ]
 [ 0.24583601 -0.06219797  0.21802678 ...  0.54129749  0.11047691
   0.0483572 ]
 [-1.26115925 -0.29051645 -1.26499659 ... -1.35138617  0.269338
  -0.28231213]
 ...
 [ 0.72709489  0.45836817  0.75277276 ...  1.46701686  1.19909344
   0.65319961]
 [ 0.25437907  1.33054477  0.15659489 ... -1.29043534 -2.22561725
  -1.59557344]
 [ 0.84100232 -0.06676434  0.8929529  ...  2.15137705  0.35629355
   0.37459546]]
```

```
print(Y_pred)
```

```
[[2.08363086e-01 8.79770339e-01]
 [4.48962301e-01 5.80053151e-01]
 [2.51291811e-01 9.35574532e-01]
 [9.64931250e-01 9.83884238e-05]
 [4.83905345e-01 4.67568696e-01]
 [9.00945783e-01 9.86850914e-03]
 [3.06370795e-01 7.65810311e-01]
 [2.07173601e-01 9.23297286e-01]
 [3.14787716e-01 7.89721906e-01]
 [3.15892905e-01 8.44666779e-01]
 [4.58597481e-01 5.89740634e-01]
 [3.50984782e-01 8.21357965e-01]
 [2.54664838e-01 8.18512440e-01]
 [3.98019075e-01 6.97004676e-01]
 [2.90783674e-01 8.56350780e-01]
 [8.55257452e-01 1.42879933e-01]
 [3.55348229e-01 8.94209564e-01]
 [1.62940875e-01 8.90240848e-01]
 [3.05653363e-01 9.12827075e-01]
 [8.56424212e-01 1.82643551e-02]
 [5.44420108e-02 9.90473866e-01]
 [2.69445330e-01 9.25823808e-01]
 [3.48066777e-01 8.90235782e-01]
 [2.27415189e-01 9.36207950e-01]
 [2.88123101e-01 8.35466743e-01]
 [6.54182136e-01 1.12178698e-01]
 [2.82397091e-01 8.41755211e-01]
 [2.71216154e-01 7.38942504e-01]
 [6.72613859e-01 2.09155083e-01]
 [6.55997455e-01 1.19338512e-01]
 [2.89074421e-01 7.91157484e-01]
 [2.85255790e-01 7.59480357e-01]
 [2.47811884e-01 8.82905304e-01]
 [8.98758411e-01 2.81256624e-03]
 [6.37099326e-01 3.61731797e-02]
 [3.55107009e-01 8.28391552e-01]
 [2.98854947e-01 9.24291432e-01]
 [4.09898818e-01 6.28928483e-01]
 [2.00336561e-01 9.67213213e-01]
 [2.67110705e-01 7.87054002e-01]
 [9.41523135e-01 6.88287546e-04]
 [6.10862136e-01 2.90557504e-01]
 [3.59834015e-01 9.90413308e-01]
 [2.00826213e-01 9.05811191e-01]
 [5.76710045e-01 1.46446303e-01]
 [3.63538206e-01 8.65408361e-01]
 [1.93431273e-01 9.48442161e-01]
 [2.52460450e-01 9.21242237e-01]
```

```
 [8.74425709e-01 7.94682000e-03]
 [5.96129656e-01 1.24191150e-01]
 [2.83543706e-01 8.01949799e-01]
 [5.28911829e-01 3.35543334e-01]
 [3.88718069e-01 7.03507006e-01]
 [3.35014343e-01 8.58531475e-01]
 [2.08889127e-01 9.32137430e-01]
 [4.70324367e-01 4.82196003e-01]
 [2.47997671e-01 9.58001554e-01]
 [8.31082761e-02 9.20015037e-01]
```

model.predict() gives the prediction probability of each class for that data point

```
#  argmax function

my_list = [0.25, 0.56]

index_of_max_value = np.argmax(my_list)
print(my_list)
print(index_of_max_value)
```

```
[0.25, 0.56]
1
```

```
# converting the prediction probability to class labels

Y_pred_labels = [np.argmax(i) for i in Y_pred]
print(Y_pred_labels)
```

```
[1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
```

Building the predictive system

```
input_data = (11.76,21.6,74.72,427.9,0.08637,0.04966,0.01657,0.01115,0.1495,0.05888,0.4062,1.21,2.635,28.47,0.005857,0.009758,0.01168,0

# change the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array as we are predicting for one data point
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardizing the input data
input_data_std = scaler.transform(input_data_reshaped)

prediction = model.predict(input_data_std)
print(prediction)

prediction_label = [np.argmax(prediction)]
print(prediction_label)

if(prediction_label[0] == 0):
  print('The tumor is Malignant')

else:
  print('The tumor is Benign')
```

```
1/1 [==============================] - 0s 28ms/step
[[0.3477521  0.90853053]]
[1]
The tumor is Benign
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler wa
  warnings.warn(
```

Accuracy of predicting using neural network comes out to be 95.6%