Breast Cancer Classification Models Comparison

The Dataset for health and it is for Social Good: Women Coders' Bootcamp organized by Artificial Intelligence for Development in collaboration with UNDP Nepal. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. A few of the images can be found at Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes. The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34]. This database is also available through the UW CS ftp server:ftp ftp.cs.wisc.educd math-prog/cpo-dataset/machine-learn/WDBC/

In [38]:
```
#imports
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import RandomizedSearchCV, cross_val_score
from sklearn.model_selection import train_test_split

import xgboost as xgb
```

In [6]:
```
#confirm libraries/packages versions
print("pandas version " + pd.__version__)
print("numpy version " + np.__version__)
print("seaborn version " + sns.__version__)
```

```
pandas version 1.4.2
numpy version 1.21.5
seaborn version 0.11.2
```

In [7]:
```
#ignore future warning code
import warnings
warnings.filterwarnings('ignore')
```

In [8]:
```
#pd.options.display.float_format = '{:,.6f}'.format
pd.set_option('display.max_columns', None) #show all columns
```

In [9]:
```
#read csv into pandas dataframe
df = pd.read_csv('Breast Ca Data.csv')
```

In [10]:
```
#evaluate top of loaded data
df.head()
```

Out[10]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030

Data Preparation and Exploration

In [11]:
```
#drop columns that won't be used in analysis
drop_columns = ['id','radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness
'concave_points_se', 'symmetry_se', 'fractal_dimension_se', 'radius_worst', 'texture_w
'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst', 'concave_poi
'fractal_dimension_worst']
df.drop(drop_columns, axis=1, inplace = True)

#reset index of dataframe
df = df.reset_index(drop = True)
```

In [15]:
```
#look at data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 11 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   diagnosis               569 non-null    object
 1   radius_mean             569 non-null    float64
 2   texture_mean            569 non-null    float64
 3   perimeter_mean          569 non-null    float64
 4   area_mean               569 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    float64
 10  fractal_dimension_mean  569 non-null    float64
dtypes: float64(10), object(1)
memory usage: 49.0+ KB
```

In [23]:
```
#get the number of columns and rows
df.shape
```

Out[23]:
```
(569, 11)
```

In [20]:
```
#evalute for null values
```

```
df.isnull().sum()
```

Out[20]:
```
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
dtype: int64
```

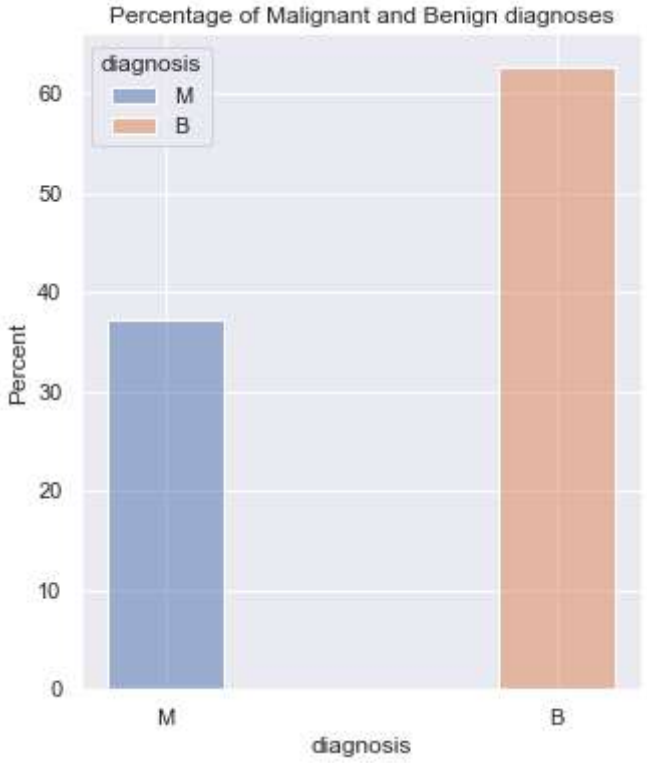
In [21]:
```
#evaluate for duplicate values
df.duplicated()
```

Out[21]:
```
0      False
1      False
2      False
3      False
4      False
       ...
564    False
565    False
566    False
567    False
568    False
Length: 569, dtype: bool
```

Results of exploration= no duplicate values found. No missing values found. There are 569 rows and 11 columns in the dataset.

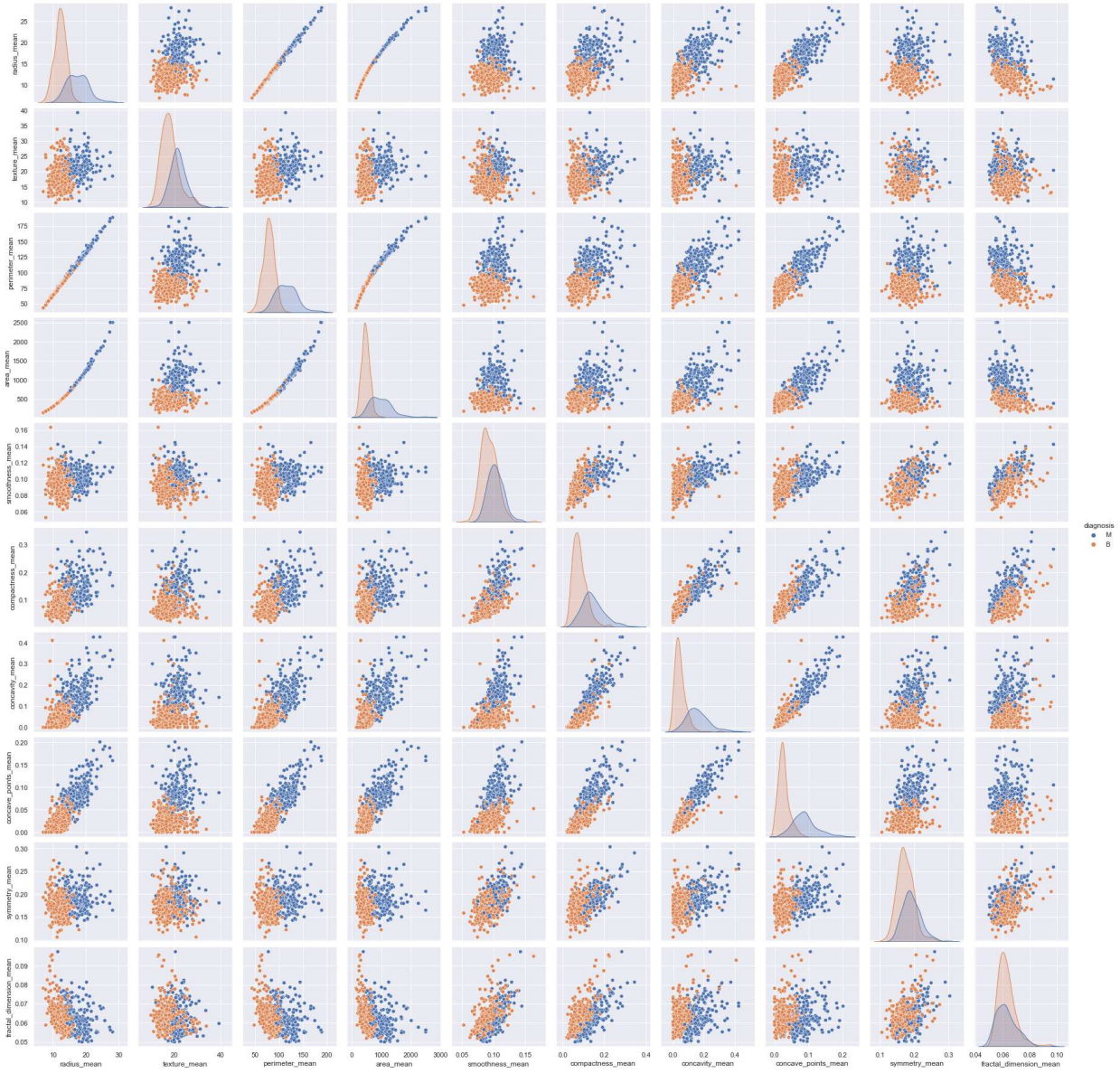
The variable "diagnosis" is an object and will require conversion to int.

In [26]:
```
#plot a histogram to see the percentages of malignant and benign diagnosis in the data
#"M"= Malignant and "B" = Benign
sns.set_theme(style="darkgrid")

fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (5,6))
g = sns.histplot(x = 'diagnosis',data = df, hue ='diagnosis', shrink = 0.3, stat = 'pe
g.set_title("Percentage of Malignant and Benign diagnoses")
sns.despine()
```



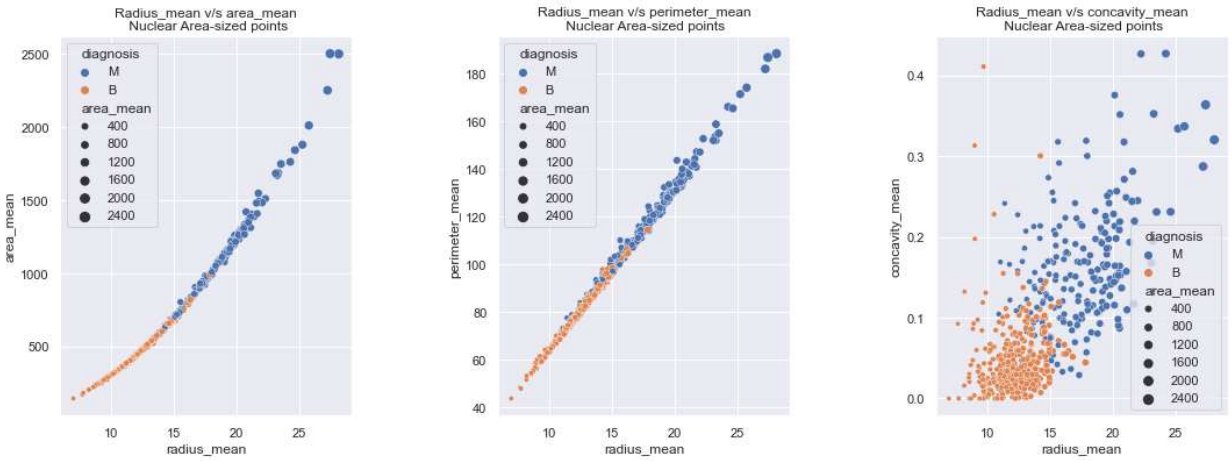
Observations= 38% Malignant and 64% Benign

In [27]:
```
#pairplot of the features to observe correlations
g = sns.pairplot(df, hue = 'diagnosis')
```



Observations = there is a linear correlation between many features.

In [28]:
```
#plot scatter plots between radius_mean and area_mean, perimeter_mean, concavity_mean
cols = ['area_mean','perimeter_mean', 'concavity_mean']
fig, ax = plt.subplots(ncols=len(cols), figsize=(6 * len(cols),6), sharex = True)
for i in range(len(cols)):
  g = sns.scatterplot(data=df, x="radius_mean", y=df[cols[i]], hue="diagnosis", size =

fig.subplots_adjust(wspace=0.5)
```



Observations= As the nuclear radius increases, the area (pie(radius_squared)) and perimeter (2pie*(radius)) are increasing linearly. The nuclear area size is larger in malignant cells(represented by larger circle points) as compared to bening cells.

Data Preprocessing

In [31]:
```
#convert 'diagnosis' to binary
#instantiate LabelEncoder
encoder = LabelEncoder()
#apply encoder on column 'diagnosis'
df['diagnosis'] = df[['diagnosis']].apply(encoder.fit_transform)
df.head()
```

Out[31]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactne
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	

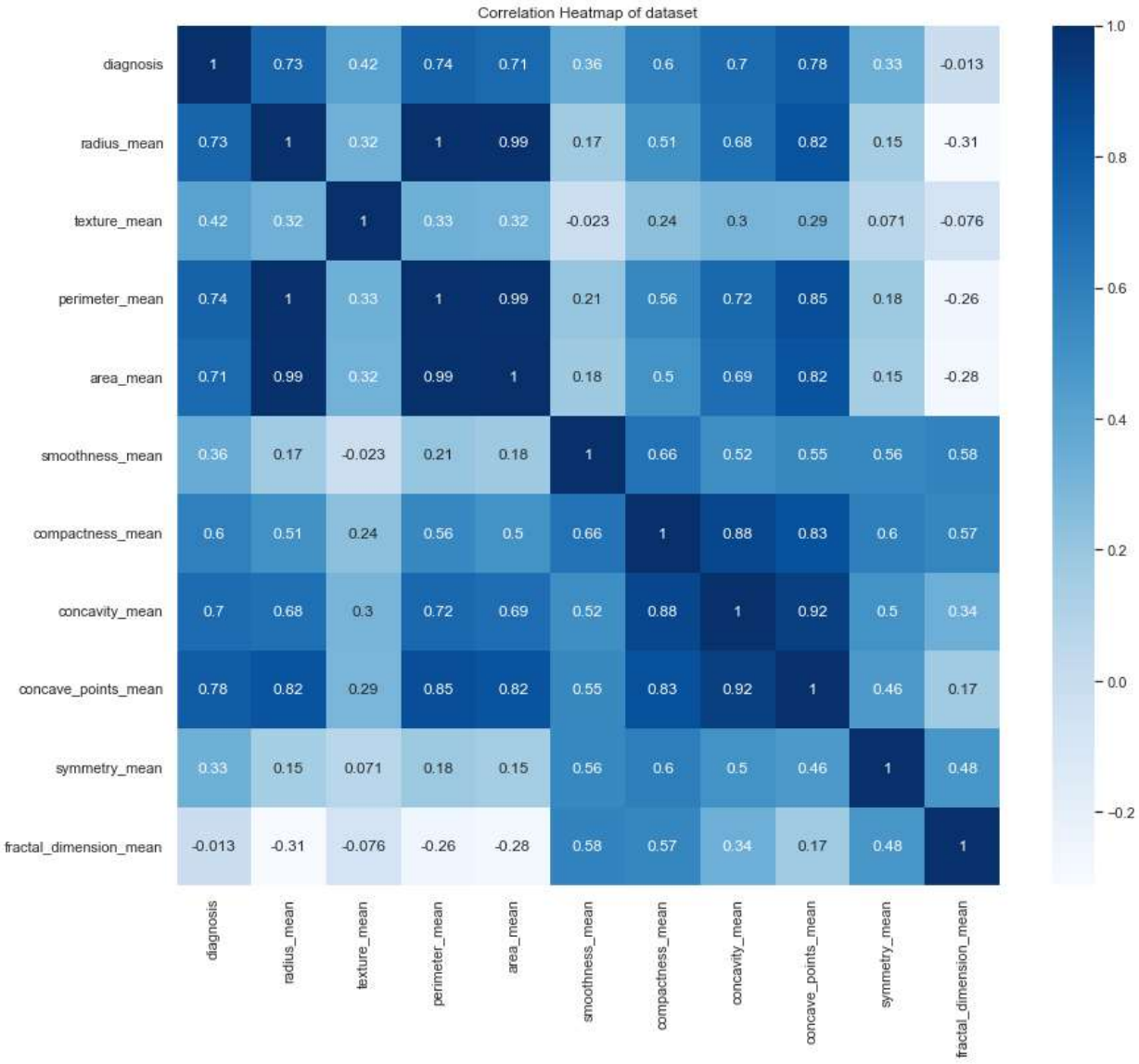
Observations= 'diagnosis' has been converted into values '1' for Malignant and '0' for Benign.

In [33]:
```
#split dataset into feature variables and target variables as X and y respectively
#copy the dataset
df_copy = df.copy(deep = True)

#create X (feature set) and y (target) variables
X = df.drop(['diagnosis'],axis=1)
y = df['diagnosis'].to_frame()

#create X_copy variable
X_copy = df_copy.drop(['diagnosis'],axis=1)
```

In [35]:
```
#apply Standard Scaler to feature variable set X_copy
#instantiate Stander Scaler and fit and tranform feature variable set X
```

```
scaler = StandardScaler(copy = True, with_mean = True, with_std = True)
X_copy_scaled = pd.DataFrame(scaler.fit_transform(X_copy), columns=X_copy.columns)
```

In [36]:
```
#dimensionality reduction using PCA
#plot Seaborn Heatmap to identify correlated features
fig = plt.figure(figsize=(14,12))
g = sns.heatmap(df.corr(), annot = True, cmap = "Blues")
g.set_title("Correlation Heatmap of dataset")
sns.despine()
```



In [37]:
```
#reduce dimensions on the scaled dataset using PCA with 95% variance retained
pca = PCA(n_components = 0.95)
pca.fit(X_copy_scaled)
X_copy_scaled = pca.transform(X_copy_scaled)
X_copy_scaled = pd.DataFrame(X_copy_scaled)
X_copy_scaled.head()
```

Out[37]:

	0	1	2	3	4
0	5.224155	3.204428	-2.171340	-0.169276	1.514252
1	1.728094	-2.540839	-1.019679	0.547539	0.312330
2	3.969757	-0.550075	-0.323569	0.397964	-0.322877
3	3.596713	6.905070	0.792832	-0.604828	0.243176
4	3.151092	-1.358072	-1.862234	-0.185251	0.311342

Model Training and Prediction

In [39]:
```
#split data into Train and Test sets with 80% of data as Train set and 20% as Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_stat

X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = train_test_split(X_copy
```

In [40]:
```
#function to fit classifier and print Accuracy score
def model(classifier, X_train, y_train,
                               X_test, y_test, X = X, y = y):
    classifier.fit(X_train,np.ravel(y_train))
    y_pred = classifier.predict(X_test)

    print("Model: ",type(classifier).__name__)
    print("Test Data Accuracy: %0.2f" % accuracy_score(y_test,y_pred))
```

In [41]:
```
#train and fit Decision Tree classifier
clf_dct = DecisionTreeClassifier()
model(clf_dct, X_train, y_train, X_test, y_test)
```

```
Model:  DecisionTreeClassifier
Test Data Accuracy: 0.93
```

In [42]:
```
#train and fit SVC classifier
clf_svc = SVC()
model(clf_svc, X_train, y_train, X_test, y_test)
```

```
Model:  SVC
Test Data Accuracy: 0.88
```

In [43]:
```
#train and fit XGB classifier
clf_xgb = xgb.XGBClassifier()
model(clf_xgb, X_train, y_train, X_test, y_test)
```

```
Model:  XGBClassifier
Test Data Accuracy: 0.96
```

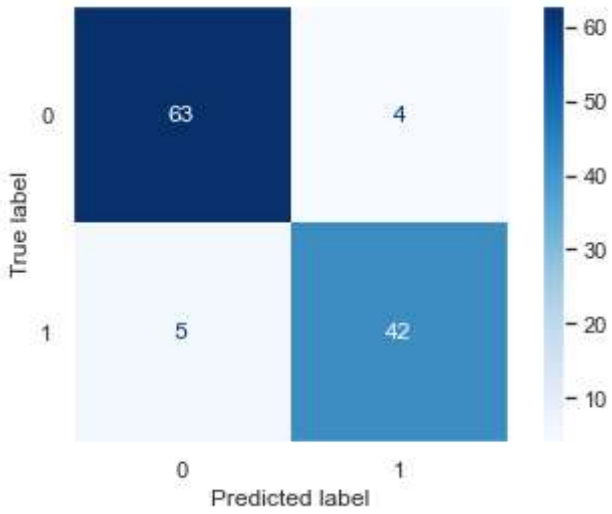
Model Evaluation

In [44]:
```
# Function to fit classifier, perform K-Fold Cross validation print Accuracy score and
def model_hypertuning(classifier, params, X_train, y_train,
                               X_test, y_test, X = X, y = y):
    classifier.set_params(**params)
    classifier.fit(X_train,np.ravel(y_train))

    y_pred = classifier.predict(X_test)
    print("Model: ",type(classifier).__name__)
```

```
    print("Test Data Accuracy: %0.2f" % accuracy_score(y_test,y_pred))
    print("K-Fold Cross Validation Accuracy: %0.2f" % cross_val_score(classifier, X, r
    print("Confusion Matrix:")
    disp = ConfusionMatrixDisplay.from_estimator( classifier,X_test,y_test,
                                  display_labels=classifier.classes_, cmap=plt.cm.Blue
    print(disp.confusion_matrix)
    plt.grid(False)
    plt.show()
```

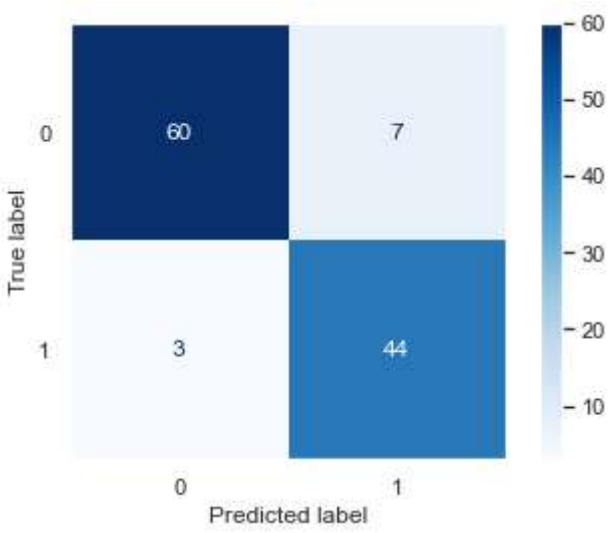
In [45]:
```
#hyperparameter tuning using RandomizedSearchCV on Decision Tree Classifier
clf_dct = DecisionTreeClassifier()

params = { 'max_depth' : [3,5,10,15,20,25,30,35,40,45,50],
           'criterion' : ["gini","entropy"],
           'max_features' : ['auto', 'sqrt', 'log2'],
           'min_samples_split' : [2,4,6,8,10]
         }

grid = RandomizedSearchCV(clf_dct, params, n_jobs=-1)
grid.fit(X_train, np.ravel(y_train))

model_hypertuning(clf_dct,grid.best_params_,X_train, y_train,
                          X_test, y_test)
```

```
Model:  DecisionTreeClassifier
Test Data Accuracy: 0.92
K-Fold Cross Validation Accuracy: 0.90
Confusion Matrix:
[[63  4]
 [ 5 42]]
```



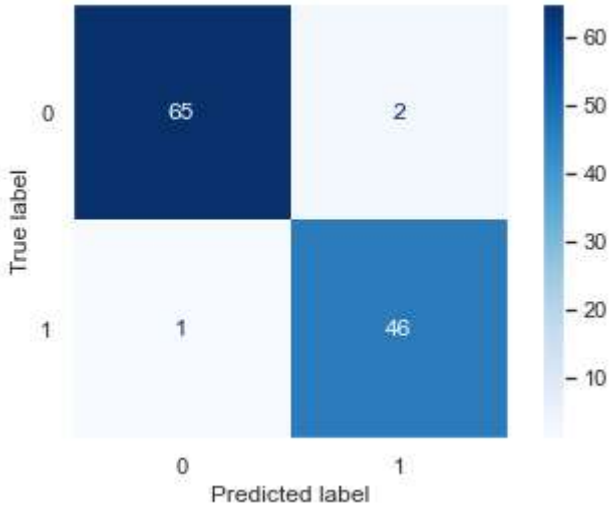
In [46]:
```
#hyperparameter tuning using using RandomizedSearchCV on SVC Classifier
clf_svc = SVC()

params = { 'kernel' : ['linear','rbf','poly','sigmoid'],
           'C' : [0.1,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5,5.5],
           'degree' : [1,2,3,4,5,6]
         }
grid = RandomizedSearchCV(clf_svc, params, n_jobs=-1)
grid.fit(X_train_scaled, np.ravel(y_train_scaled))

model_hypertuning(clf_svc,grid.best_params_,X_train_scaled, y_train_scaled, X_test_sca
```

```
Model:  SVC
Test Data Accuracy: 0.91
K-Fold Cross Validation Accuracy: 0.89
Confusion Matrix:
[[60  7]
 [ 3 44]]
```



In [47]:
```
#hyperparameter tuning using RandomizedSearchCV on XG Boost Classifier
clf_xgb = xgb.XGBClassifier()

params = { 'learning_rate' : [0.05,0.1,0.15,0.2,0.25,0.3],
           'max_depth' : [3,4,5,6,8,10,12,16],
           'min_child_weight' : [1,3,5,7],
           'gamma' : [0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4],
           'colsample_bytree' : [0.3,0.4,0.5,0.6]
         }
grid = RandomizedSearchCV(clf_xgb, params, n_jobs=-1)
grid.fit(X_train, np.ravel(y_train))

model_hypertuning(clf_xgb,grid.best_params_,X_train, y_train,
                          X_test, y_test)
```

```
Model:  XGBClassifier
Test Data Accuracy: 0.97
K-Fold Cross Validation Accuracy: 0.95
Confusion Matrix:
[[65  2]
 [ 1 46]]
```



Observations= 1.Hyperparameter tuning marginally increased the accuracy of all models.
2.There is a scope for improvement in accuracy specially in avoiding False Negatives (meaning the diagnosis of a case of benign when it's actually malignant)

1. Exploring other features and outlier detection and removal may improve model perfomance.

In [ ]: