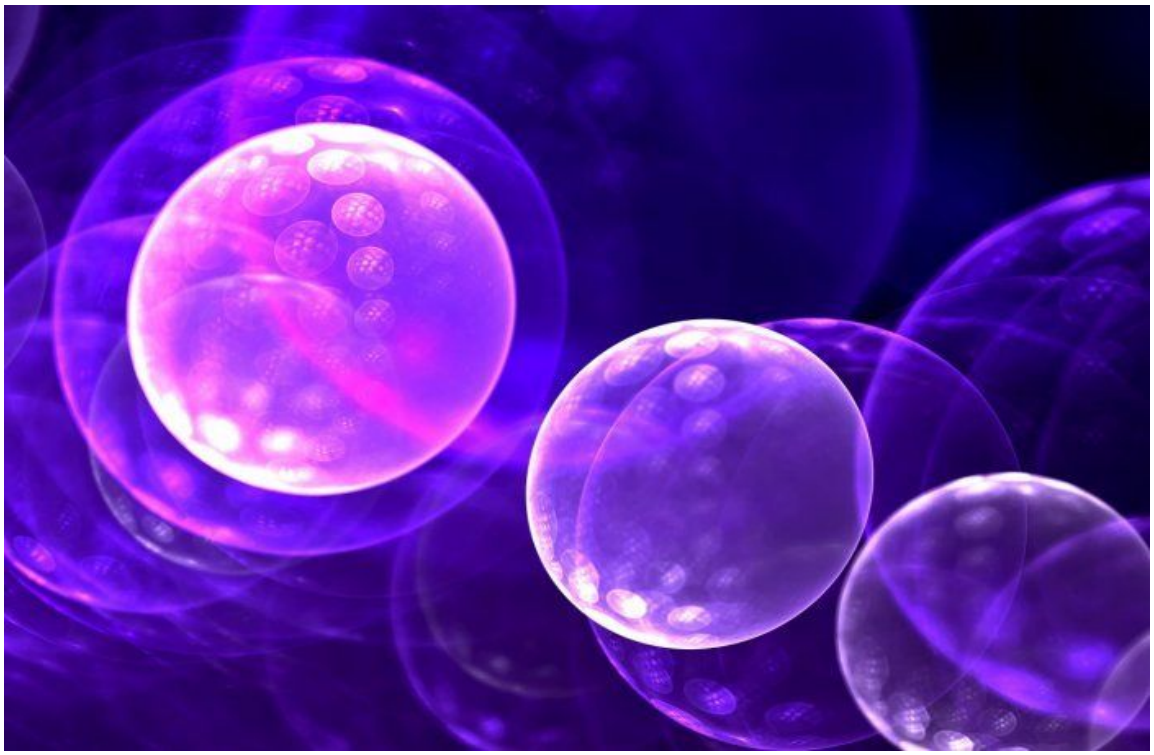
Predict whether Breast Cancer is benign or malignant



Team Members:

16ucs036 (Aneesh Jain)

16ucc026 (Bhavesh Kalra)

16ucc075 (Rishabh Mittal)

16ucs133 (Pranav Kumar Mishra)

Table of Contents

Abstract

Of all the known cancers in the world, breast cancer rates are alarming. In India, we are seeing more and more patients being diagnosed with breast cancer. It is the most common cancer in most parts of India, both urban and rural. During life 8% of womens are diagnosed with breast cancer. Current Indian data show that one in 22 women develop breast cancer. Out of two who develop it, one dies.

Today, Machine Learning (ML) techniques are being broadly used in the breast cancer classification problem. In this report we try to classify breast cancer as **malignant** (tending to be severe and progressively worse) and **benign** (not to spread or invade nearby tissue) using some of most common machine learning algorithm.

We use classification algorithm for diagnosis of cancer into benign and malignant . In this project we use three different classifiers: Naive Bayes (NB), Decision Tree (DT) and k nearest neighbor (KNN) for breast cancer classification. We compare between the different implementations and evaluate their accuracy using cross validation. Results show that KNN gives the highest accuracy (94% appx) with lowest error rate then DT classifier (88% appx.)

Also we will be exploring some of best practices used in machine learning like class imbalance problem, overfitting problem, heatmap analysis to further refine our classification algorithms, so that it can perfectly predict the class label (benign and malignant) of provided unknown data.

Data Collection and Exploration

Introduction to problem:

There has been a gigantic increase in the number of breast cancer cases in the world in last few years.So,we have taken the breast cancer dataset to find out if the breast cancer case is **“Malignant”** or **“Benign”. Malignant** is assumed as positive class and **Benign** is assumed as negative class.

Dataset Overview:

Different attributes types in dataset

```
In [5]: cancer_dataset.dtypes

Out[5]: id                         int64
        diagnosis                 object
        radius_mean              float64
        texture_mean             float64
        perimeter_mean           float64
        area_mean                float64
        smoothness_mean          float64
        compactness_mean         float64
        concavity_mean           float64
        concave points_mean      float64
        symmetry_mean            float64
        fractal_dimension_mean   float64
        radius_se                float64
        texture_se               float64
        perimeter_se             float64
        area_se                  float64
        smoothness_se            float64
        compactness_se           float64
        concavity_se             float64
        concave points_se        float64
```

Dataset Overview (column =31, rows = 589)

```
In [3]:  cancer_dataset = pd.read_csv("data.csv")
         cancer_dataset.head(10)
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280
5	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000
6	844359	M	18.25	19.98	119.60	1040.0	0.09463	0.10900
7	84458202	M	13.71	20.83	90.20	577.9	0.11890	0.16450
8	844981	M	13.00	21.82	87.50	519.8	0.12730	0.19320
9	84501001	M	12.46	24.04	83.97	475.9	0.11860	0.23960

Activate Windows
Go to Settings to activate W

Explaining Features of the dataset:

Diagnosis - Finding the diagnosis of breast tissues(M=malignant,B=benign).
Radius_mean - Mean of distances from center to points on the perimeter.
Texture_mean - standard deviation of gray-scale values.
Perimeter_mean - Mean size of the core tumor
Smoothness_mean - Mean of local variations in radius lengths.
Compactness_mean - mean of perimeter^2 / area - 1.0
Concavity_mean - mean of severity of concave portions of the contour
Concave points_mean - mean for number of concave portions of the contour

Fractal_dimension_mean - mean for "coastline approximation" - 1
Radius-se - standard error for the mean of distances from center to points on the perimeter
Texture_se - standard error for standard deviation of gray-scale values

Smoothness_se - standard error for local variation in radius lengths
Compactness_se - standard error for perimeter^2 / area - 1.0
Concavity_se - standard error for severity of concave portions of the contour
Concave_points_se - standard error for number of concave portions of the contour
Fractal_dimension_se - standard error for “coastline approximation” - 1
Radius_worst - “worst” or largest mean value for mean of distances from center to points on the perimeter
Texture_worst - “worst” or largest mean value for local variation in radius lengths

PreProcessing of Dataset

Class Imbalance Problem

It is the problem in machine learning where **the total number of a class of data (positive) is far less than the total number of another class of data (negative)**.Most machine learning algorithms and works best when the number of instances of each classes are roughly equal. When the number of instances of one class far exceeds the other, problems arise.

In our Breast Cancer dataset first we will check for class Imbalance Problem for Malignant and Benign class attributes.

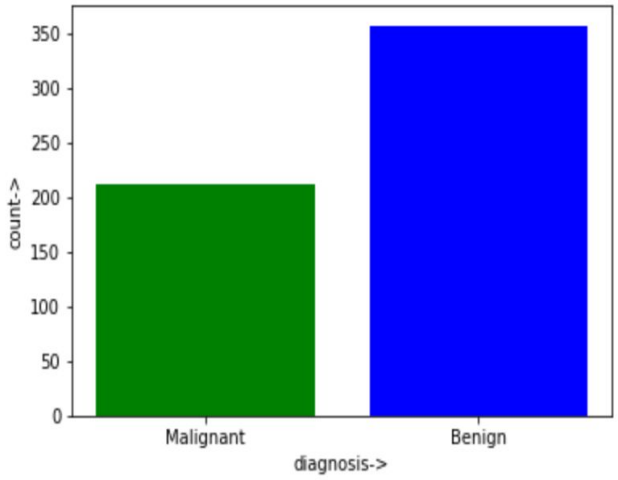
CODE Snippet for checking class imbalance problem (Python)

```
In [83]: total = cancer_dataset['diagnosis'].count()

         # checking for total malignant cases
         mask = cancer_dataset['diagnosis'] == 'M'
         total_malignant = cancer_dataset[mask].count()
         total_malignant['diagnosis']

         # checking for total benign cases
         mask = cancer_dataset['diagnosis'] == 'B'
         total_benign = cancer_dataset[mask].count()
         print("Benign% : ", total_benign['diagnosis']*100/total)
         print("Malignant% : ", total_malignant['diagnosis']*100/total)

         Benign% :  62.74165202108963
         Malignant% :  37.25834797891037
```

Plotting Malignant and Benign Cases

```
In [8]: Malignant,Benign = cancer_dataset['diagnosis'].value_counts()
objects = ('Malignant','Benign')
y_pos = np.arange(len(objects))
performance = [Benign,Malignant]

plt.bar(y_pos, performance, align='center',color=['green', 'blue', 'cyan'])
plt.xticks(y_pos, objects)
plt.xlabel('diagnosis->')
plt.ylabel('count->')

plt.show()
```

Plot



Conclusion： Dataset is not class imbalance problem.

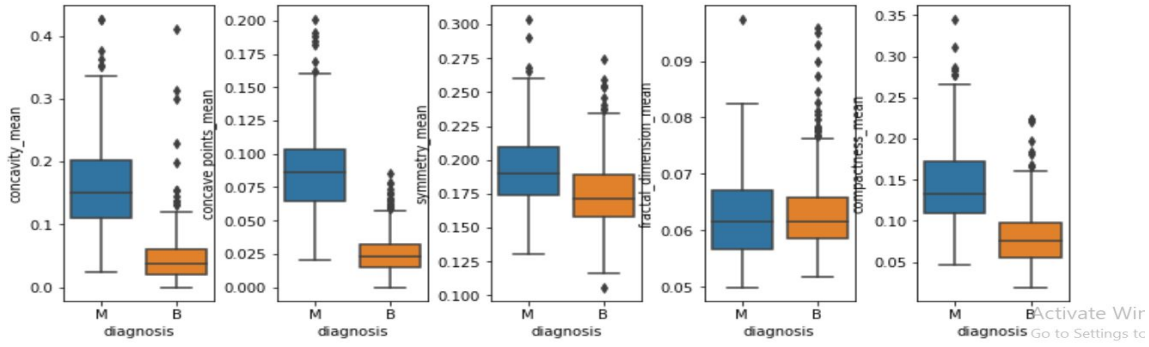
Boxplot of various attributes

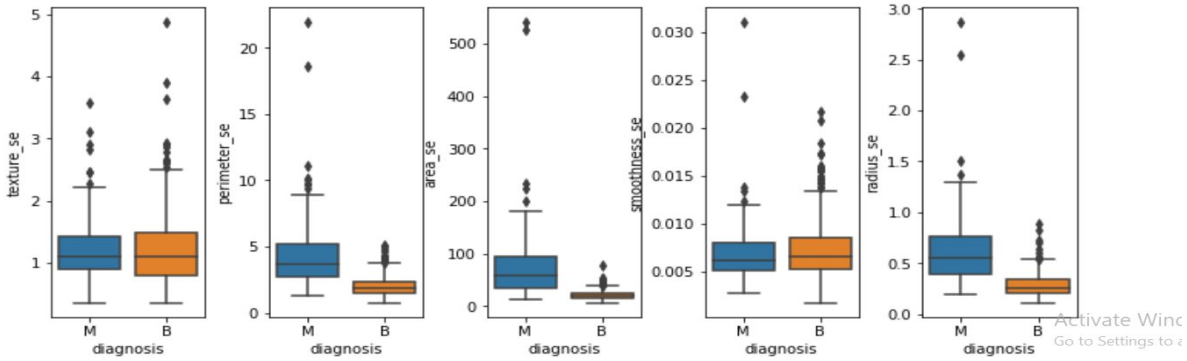
Plotting of boxplots of different attributes against diagnosis(malignant, benign)

Code Snippet of Boxplot (Python)

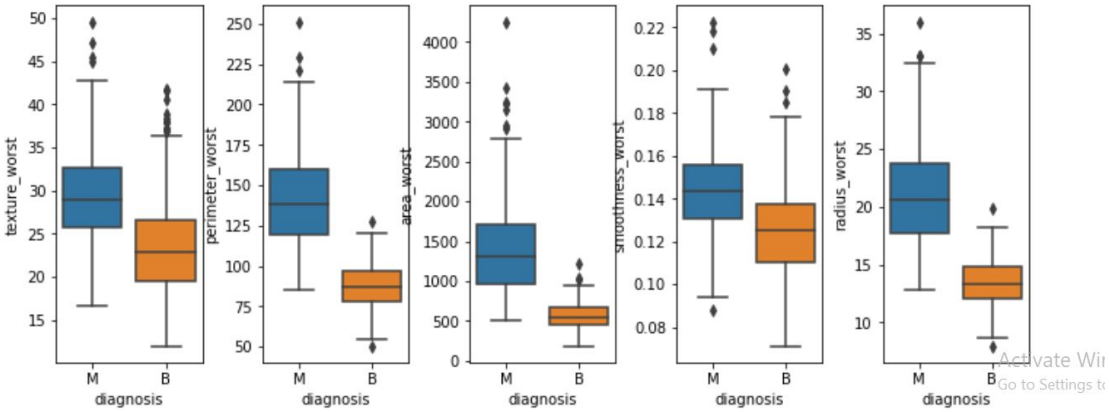
In [39]:
```
%matplotlib inline
df1 = cancer_dataset
l=len(df1.columns)
plt.rcParams['figure.figsize']=(10,4)
columns=list(df1.columns)
columns.remove('diagnosis')
i=0
for col in columns:
    if i%5==0:
        i=0
        axes=[i for i in range(5)]
        f,axes = plt.subplots(1,5)
        f.tight_layout()
    sns.boxplot('diagnosis',y=col,data=df1,ax=axes[i-1])
    i+=1
```

BoxPlots of different attributes vs diagnosis :





Activate Wir
Go to Settings to
Activate Winc
Go to Settings to

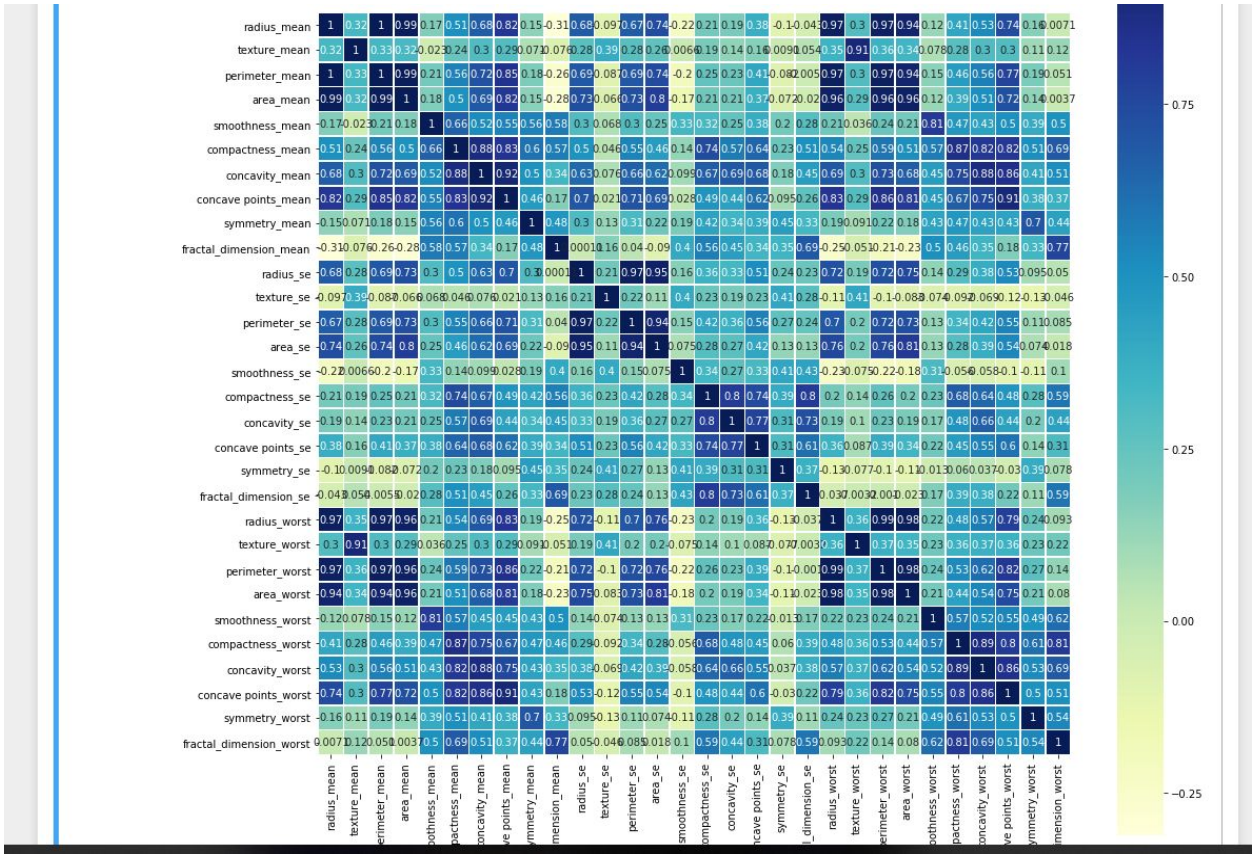


Removing redundant features by Heatmap:

Correlation heatmap give a sober representation of relation between different features, by this we can analyze overall quality of data and can remove some of features to simplify our dataset to further extent. For our analysis we choose a **threshold value of 0.9 for correlation.** If two attributes have a correlation of greater than 0.9 drop one of attribute

Plot a correlation heatmap of all the 31 attributes (correlation of one attribute with remaining 30 attributes) and derive a list of all pairs where **correlation is greater than 0.9.** Then derive a list of pairs where correlation is greater than 0.9 and remove any one attribute. Thus reducing the total number of attributes to 22 only.

Correlation Heatmap



Correlation Heatmap code snippet (Python)

In [30]:

```
fig,ax=plt.subplots(figsize=(16,16))
corr=cancer_dataset.corr()
sns.heatmap(corr,cmap="YlGnBu",annot=True,linewidth=0.5,square=True)
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1ebe7e14860>

Conclusion：After heatmap analysis total non class attributes were reduced to 22

Component Analysis

Scaling Issues in dataset

There are many features or attributes in a dataset all having different statistical parameters (mean, variance, min, max). Some features ranges generally lie between 0-1 while other feature range may lie between 100-1000, due to which some of features are hijacking contribution of other features and hinders accuracy of our model.

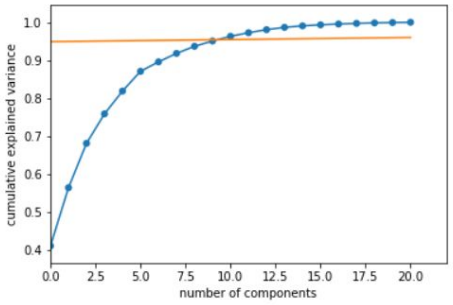
We will be using standard normalisation, mean = 0 and variance = 1 using **StandardScalar()** function. It assumes your data is normally distributed within each feature and will scale them such that the distribution is now centred around 0, with a standard deviation of 1. The mean and standard deviation are calculated for the feature and then the feature is scaled based on 0 and 1.

Cumulative Variance Plot

After performing exploratory analysis we have a total of 21 attributes. It can be further reduced by cumulative variance analysis.
Cumulative Variance analysis : In this we calculate variance of a particular feature of dataset, then taking the cumulative sum over all features of dataset to obtain a graph prescribed below
Set a **threshold** value of value of 0.9 and discard all features that are above the threshold in the plot (we can get it list by a simple python code)

In [71]:
```
scaler=StandardScaler()
X_train_std=scaler.fit_transform(X_train)
X_test_std=scaler.transform(X_test)
pca=PCA().fit(X_train_std)
x1,y1=[0,20],[0.949,0.96]
x=list(range(0,list(df1.shape)[1]-1))
plt.scatter(x,np.cumsum(pca.explained_variance_ratio_),cmap = "coolwarm", edgecolor = "None", alpha=1.0)
plt.plot(x,np.cumsum(pca.explained_variance_ratio_),x1,y1,marker='')

plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.xlim(0,22,2)
```

Out[71]: (0, 22)



The two first components explains the 0.6324 of the variance. We need 10 principal components to explain more than 0.95 of the variance and 17 to explain more than 0.99

Conclusion : Total attributes for classification is reduced to 10 only

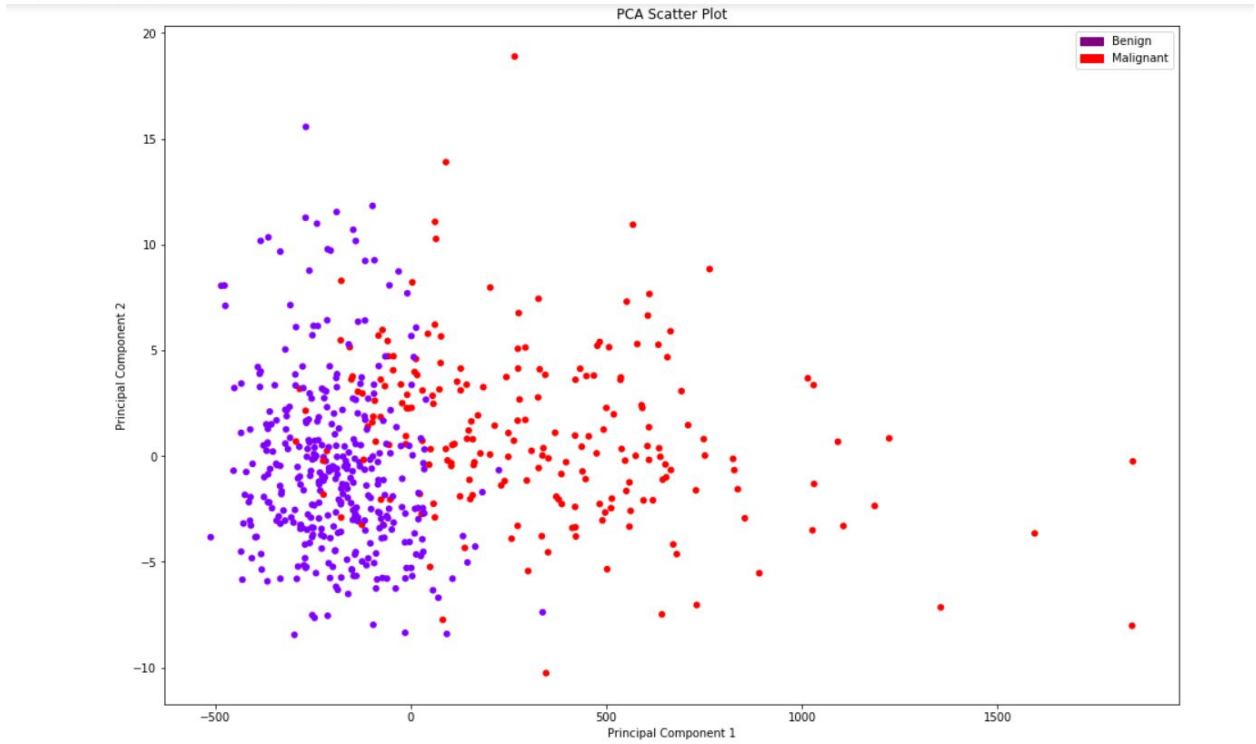
PCA scatter plot :

The PCA scatter plot represent how variantly the malignant and benign cases are spread in the dataset

Code snippet of scatter plot (Python)

In [57]:
```
%matplotlib inline
y_df= pd.get_dummies(df1.diagnosis,drop_first=True) # dropping the column called diagnosis and having a columns of 0 and 1
y_df=y_df['M']
prueba=pd.get_dummies(df1,'diagnosis')
prueba.drop('diagnosis_B',axis=1)

X=df1
X=X.drop('diagnosis',axis=1)
X = X.values
# Call PCA method
# let's call n_components = 2
pca = PCA(n_components=2)
pca_2d = pca.fit_transform(X)
# Plot the PCA
plt.figure(figsize = (16,11))
plt.scatter(pca_2d[:,0],pca_2d[:,1], c = y_df,
            cmap = "rainbow", edgecolor = "None", alpha=0.5,);
plt.title('PCA Scatter Plot');

import matplotlib.patches as mpatches
rects=[]
rects.append(mpatches.Patch(color='purple', label='Benign'));
rects.append(mpatches.Patch(color='red', label='Malignant'));
plt.legend(handles=rects);
```

Scatter Plot



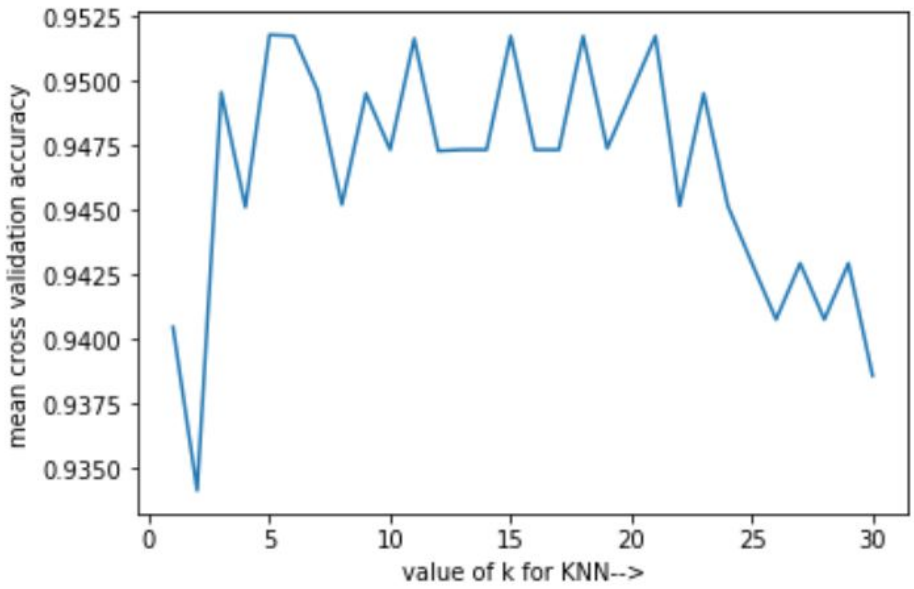
Evaluation Models

Overfitting Problem

A machine learning model is said to be overfitted, when we train it with a lot of data . When a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set. Then the model does not categorize the data correctly, because of too much of details and noise. Overfitting generally happens in non-linear methods like (K nearest neighbour) KNN, Naive Bayes(NB) because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models.

Cross- Validation: A standard way to find out-of-sample prediction error is to use 10-fold cross validation. In this method we are dividing the dataset into 10 parts known as folds thus building a total of ten homogeneous KNN models.

Accuracy after applying Cross validation on KNN algorithm



KNN classification

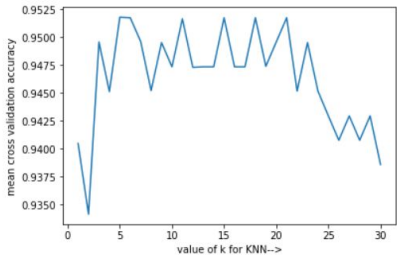
In KNN classification we are building the model over different values of k (ranging from 1 - 31) and applying 10 fold cross validation for each of the model, thus getting ten different accuracy value for each fold.

Now for each value of K, calculate mean accuracy, thus getting a total of 31 different accuracy

The maximum accuracy occur at **K = 7** (by looking at the diagram) thus we build KNN classification on k = 7

In [36]:
```
k_range=range(1,31)
k_accuracy=[]
for k in range(1,31):
  knn=KNeighborsClassifier(n_neighbors= k)
  accuracy = cross_val_score(knn,  X_train_pca, y_train,cv=10,scoring='accuracy')
  x = np.mean(accuracy)
  k_accuracy.append(x)


plt.plot(k_range,k_accuracy)
plt.xlabel('value of k for KNN-->')
plt.ylabel('mean cross validation accuracy')
```

Out[36]: Text(0,0.5,'mean cross validation accuracy')



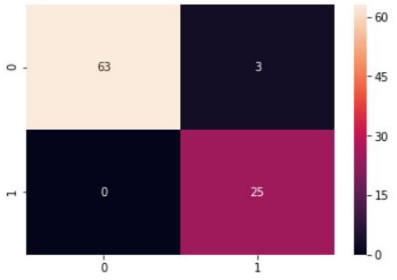
Activate Windov

KNN Classifier

In [15]:
```
classifiers_acc=[]
```

In [39]:
```
knn=KNeighborsClassifier(n_neighbors= 7)
knn_score = np.mean(cross_val_score(knn,  X_train_pca, y_train, scoring='accuracy'))
print("Accuracy: %s" % '{:.2%}'.format(knn_score))

X1_train,X1_test,y1_train,y1_test= train_test_split(X_train_pca, y_train,test_size=0.2,random_state=21)
knn.fit(X1_train,y1_train)
y_pred=knn.predict(X1_test)
con=metrics.confusion_matrix(y1_test,y_pred)
print('Confusion Matrix:')
sns.heatmap(con, annot=True);
```

```
Accuracy: 95.83%
Confusion Matrix:
```

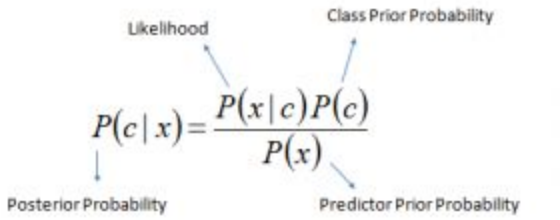


Naive Bayes

Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as ‘Naive’. Naive Bayes model is easy to build and particularly used for large dataset

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

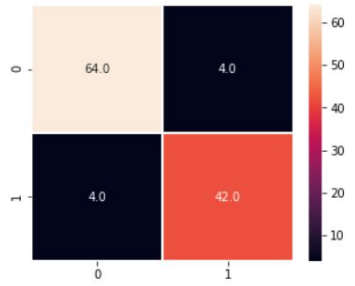
Naive Bayes code snippet (Python)

Naive Bayes Classifier

In [73]:

```
# Create the model
modelo_NB= GaussianNB()
# Fit the model
modelo_NB.fit(X=X_train,y=y_train)
#Prediction
predicion_NB = modelo_NB.predict(X_test)
#Results

#Clasification report
results_NB=metrics.classification_report(y_true=y_test, y_pred=predicion_NB)
print(results_NB)
naive_score=0.93
print("Accuracy for Naive Bayes Classifier: %s" % '{:.2%}'.format(naive_score))

#Confusion Matrix
cm_NB=metrics.confusion_matrix(y_true=y_test, y_pred=predicion_NB)
f,ax = plt.subplots(figsize=(5, 4))
sns.heatmap(cm_NB, annot=True, linewidths=.5, fmt= '.1f',ax=ax);
```

```
              precision    recall  f1-score   support

           B       0.94      0.94      0.94        68
           M       0.91      0.91      0.91        46

   micro avg       0.93      0.93      0.93       114
   macro avg       0.93      0.93      0.93       114
weighted avg       0.93      0.93      0.93       114

Accuracy for Naive Bayes Classifier: 93.00%
```



Decision Tree

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute , each branch represents the outcome of the test, and each leaf node represents a class label. The paths from root to leaf represent classification rules

1. Decision nodes – typically represented by squares
2. Chance nodes – typically represented by circles
3. End nodes – typically represented by triangles

Decision Tree Classifier

```
In [17]: dt=DecisionTreeClassifier(random_state=7)
         score_dt = np.mean(cross_val_score(dt,  X_train_pca, y_train, scoring='accuracy'))
         print("Accuracy for Decision Tree: %s" % '{:.2%}'.format(score_dt))

         Accuracy for Decision Tree: 89.90%
```

Conclusion

After building different classification models namely KNN, Decision Tree and Naive Bayes, we conclude that KNN has the best accuracy out of all three

1. KNN has best accuracy due to 10- fold cross validation accuracy of 95%
2. Decision Tree model has accuracy of 89%
3. Naive bayes has accuracy of 93%

Confusion Matrix

KNN classification (k=7)

Actual/Predicted	Benign(yes)	Malignant(no)
Benign(yes)	63	3
Malignant(no)	1	25

Naive Bayes

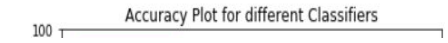
Actual/Predicted	Benign(yes)	Malignant(no)
Benign(yes)	64	4
Malignant(no)	4	42

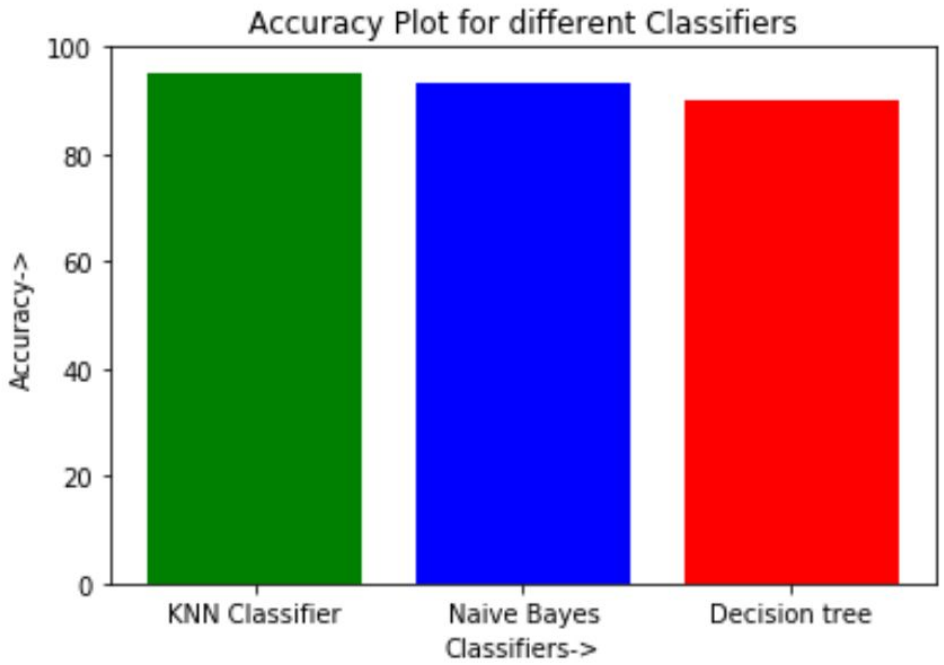
Accuracy comparison of Models

Model Type	KNN classification	Decision Tree	Naive Bayes
Accuracy	95.83%	89.90%	93.00%

Code snippet of comparison model (python)

```
In [29]: classifiers_acc=[knn_score*100,naive_score*100,score_dt*100]
         #print(classifiers_acc)
         objects = ('KNN Classifier','Naive Bayes','Decision tree')
         y_pos = np.arange(len(objects))

         plt.bar(y_pos, classifiers_acc, align='center',color=['green', 'blue', 'red'])
         plt.xticks(y_pos, objects)
         plt.xlabel('Classifiers->')
         plt.ylabel('Accuracy->')
         plt.title('Accuracy Plot for different Classifiers')
         plt.show()
```





References

- Breast cancer analysis research paper
  https://ieeexplore.ieee.org/document/8391453
- Breast cancer dataset
  https://www.kaggle.com/lbronchal/breast-cancer-dataset-analysis
- Overfitting problem
  https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/
- Normalization and scaling
  https://sebastianraschka.com/Articles/2014_about_feature_scaling.html