**ST: BIG DATA ANALYTICS**

**(CS 696-16) (FA18)**

**Project 2**

**“Classification and Clustering Analysis of Breast Cancer Wisconsin (Diagnostic) Data Set”**

**Submitted By**

**Ashok Kumar Shrestha**

**#A25265227**

**Contents**

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Topics** | **Page No.** |
| 1. | Introduction | 1 |
| 2. | Data Visualization | 1 |
| 3. | Methods | 3 |
| 4. | Results and Discussion | 3 |
| 5. | Conclusion | 7 |
| 6. | References | 8 |
| 7. | Appendix | 9 |

**Introduction:**

In this project, Breast Cancer Wisconsin (Diagnostic) data set[1] is used and various clustering analysis and classification techniques are implemented. This data set can be found in Kaggle[1], UCI Machine Learning Repository[2] and Sklearn[3]. The data set features are computed from digitized image of fine needle aspirate (FNA) of a breast mass. For the sake of simplicity, Kaggle data set is used.

**Attribute Information:**

There are total of 33 features including ID number and Diagnosis (Benign or Malignant). Other attributes include real-valued features computed for cell nuclei. There are listed below:

* 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)

For every feature listed above, mean, standard error and "worst" or largest (mean of the three largest values) were computed. All the values are four significant digits except for diagnosis which is label for each data sample (M: Malignant, B: Benign). The data set consist of 569 data samples.

All the codings are done in Python on Jupyter Notebook[4] with Sklearn library support[5].

**Data Visualization:**

The Data set consist of 569 samples labeled into two classes: Benign and Malignant. Figure 1. bar-graph shows sample distribution between two classes.

In Kaggle data set, last column “Unnamed: 32” contains NaN value which does not represent any useful information. So, it can be dropped along with first column “id”. On the same note, “Diagnosis” contains labels M or B which is formatted to integer value of 1 and 0 respectively.

Fig 10. shows the correlation heatmap for the data set. The heat-map gives general idea about the nature of data and features. Looking at the covariance value, redundant features can be removed. Such as radius\_mean, perimeter\_mean, area\_mean are all highly correlated and also with radius\_worst, perimeter\_worst, area\_worst. Similarly, compactness\_worst, concave\_points\_ worst, concavity\_worst also are highly correlated with compactness\_mean, concave\_mean, and Concavity\_mean. These highly correlated features are redundant and can be drop from the data.

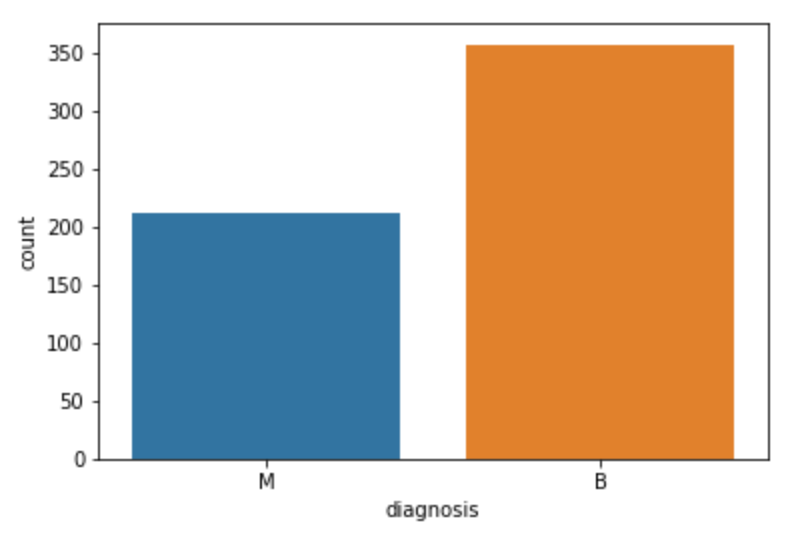
****

Fig 1: Sample distribution between two classes.

Other way to capture important features from the data set is by using decision tree or random forest. From the below figure it is clear that following five features are of most important: mean concave points, worst texture, worst radius, worst area and concave points error.

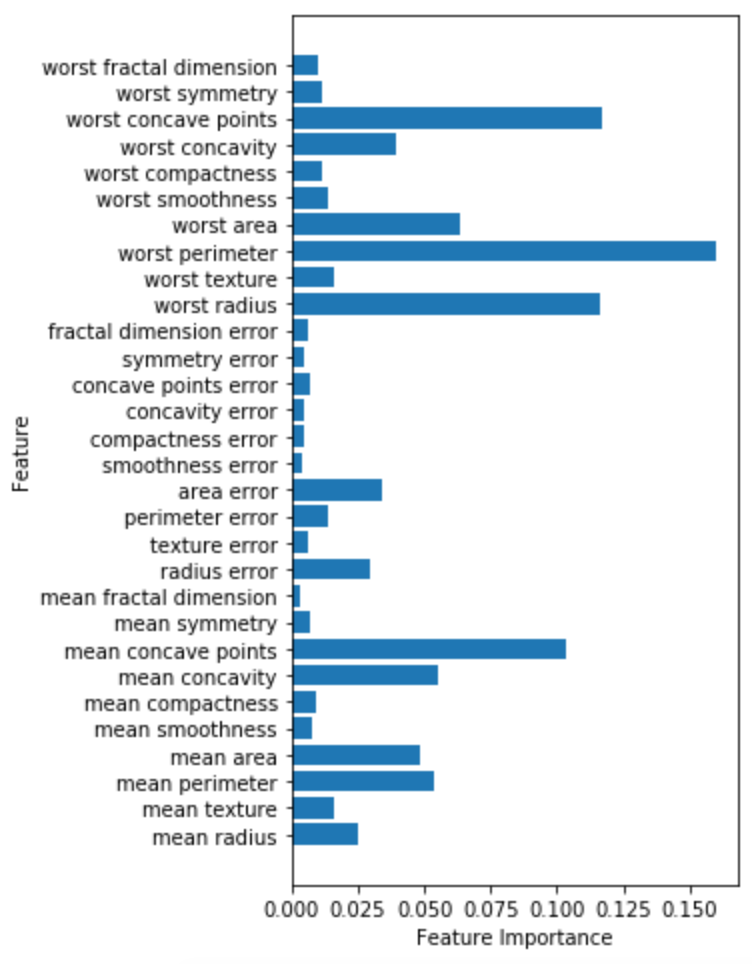
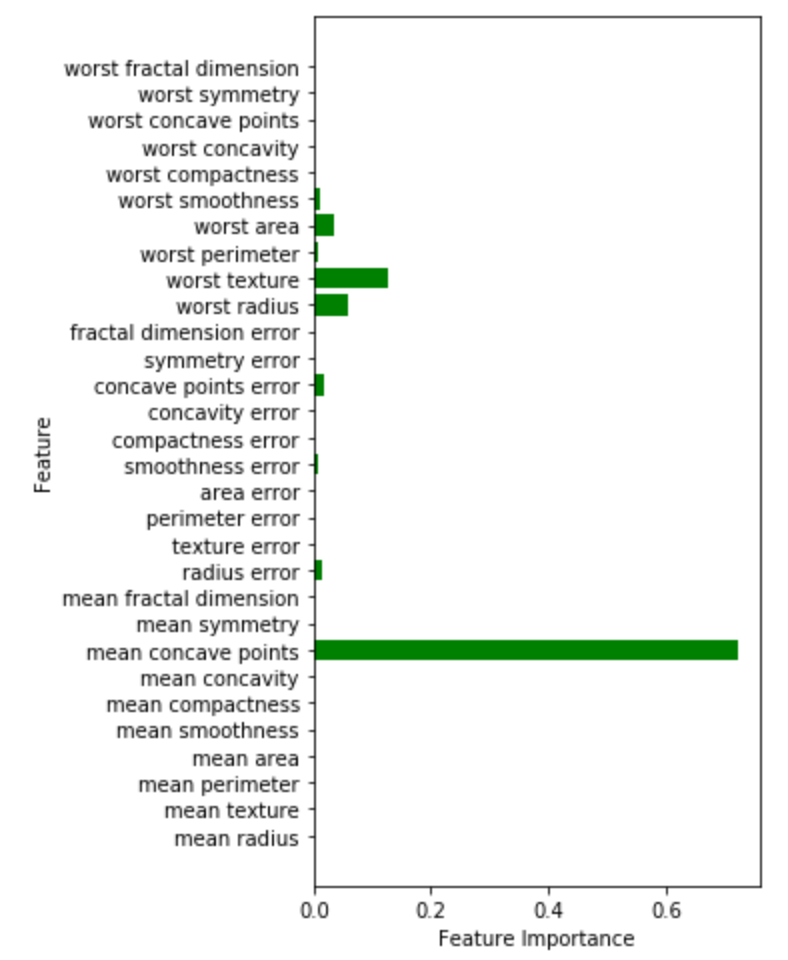


Fig 2: Feature importance in Decision tree. Fig 3: Feature importance in Random Forest

**Methods:**

In this project few classification algorithms such as KNN, Logistic regression, Decision tree, Random forests, MLP and SVM for our dataset have been explored. Here, data sets are preprocessed using StandardScaler. The data consists of 569 samples which are divided into training and testing samples using train\_test\_split.

Similarly, different clustering algorithms such as K-Means Clustering, Hierarchical Agglomerative Clustering and Spectral Clustering are used to analyse the data sets. Here also data sets are preprocessed using StandardScaler.

**Results and Discussion:**

The data consists of 569 samples which are divided into training and testing samples using train\_test\_split. Some Hyperparameters have been tuned to improve testing accuracy and overcome overfitting problems.

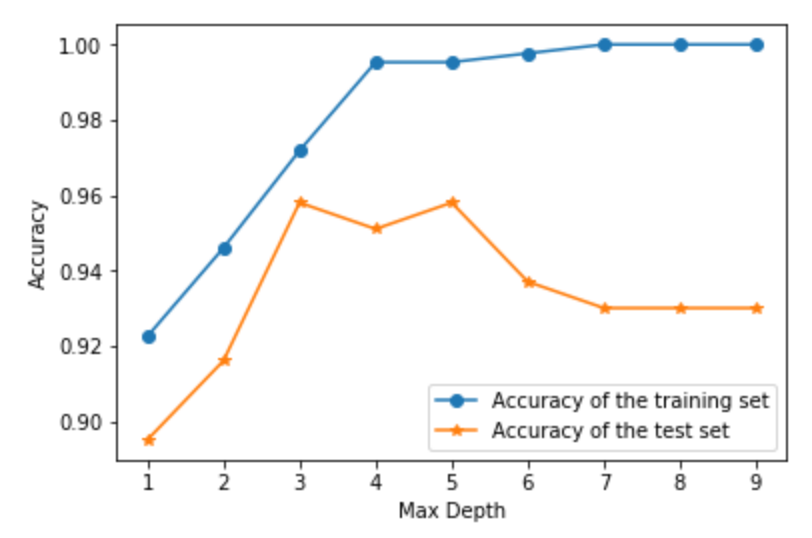
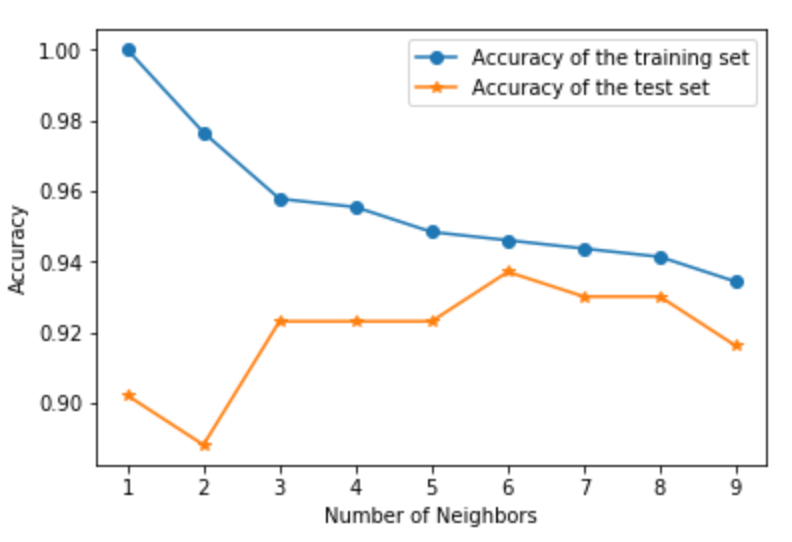


Fig 4: KNN Classifier. Fig 5: Decision Tree

As seen from the above graph, best value of K is 6 for KNN Classifier. Similarly, best value of max depth is 3. Let us compare the accuracy of the classifiers.

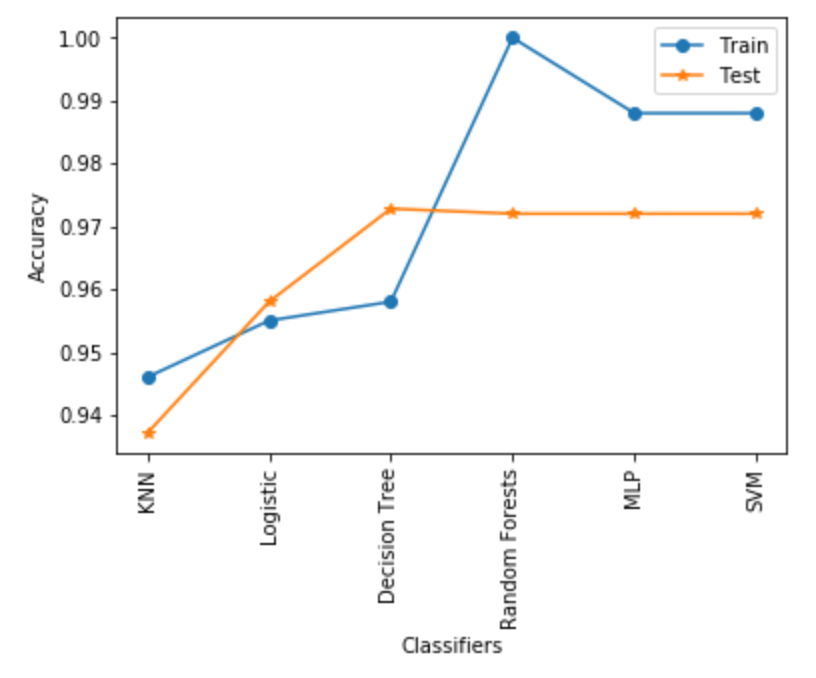


Fig 6: Accuracy comparison of classifiers

**Classifier specifications:**

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',  
 metric\_params=None, n\_jobs=1, n\_neighbors=9, p=2,  
 weights='uniform')

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,  
 intercept\_scaling=1, max\_iter=100, multi\_class='ovr', n\_jobs=1,  
 penalty='l2', random\_state=None, solver='liblinear', tol=0.0001,  
 verbose=0, warm\_start=False)

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=9,  
 max\_features=None, max\_leaf\_nodes=None,  
 min\_impurity\_decrease=0.0, min\_impurity\_split=None,  
 min\_samples\_leaf=1, min\_samples\_split=2,  
 min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=0,  
 splitter='best')

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',  
 max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,  
 min\_impurity\_decrease=0.0, min\_impurity\_split=None,  
 min\_samples\_leaf=1, min\_samples\_split=2,  
 min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=1,  
 oob\_score=False, random\_state=0, verbose=0, warm\_start=False)

MLPClassifier(activation='relu', alpha=1, batch\_size='auto', beta\_1=0.9,  
 beta\_2=0.999, early\_stopping=False, epsilon=1e-08,  
 hidden\_layer\_sizes=(100,), learning\_rate='constant',

learning\_rate\_init=0.001, max\_iter=1000, momentum=0.9,  
 nesterovs\_momentum=True, power\_t=0.5, random\_state=42, shuffle=True,  
 solver='adam', tol=0.0001, validation\_fraction=0.1, verbose=False,  
 warm\_start=False)

SVC(C=1000, cache\_size=200, class\_weight=None, coef0=0.0,  
 decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='rbf',  
 max\_iter=-1, probability=False, random\_state=None, shrinking=True,  
 tol=0.001, verbose=False)

**Clustering:**

In K-Means clustering, elbow method can be used to find optimal no. of clusters. As shown in fig 8, any 2, 3 or 4 no. of clusters can be used. For better analysis, Silhouette analysis can be used as shown in fig 9. As shown in the with 2 clusters maximum Silhouette score is obtained. Thus we can conclude that optimal no. of clusters for our data set is 2 which corresponds to the two classes. So, for further analysis, no. of clusters used is 2.

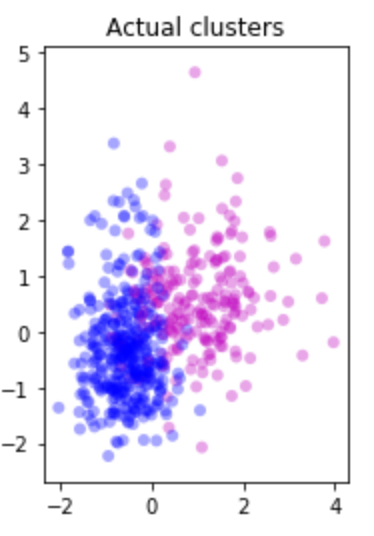
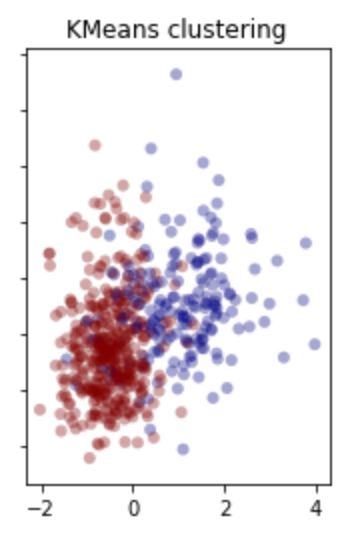
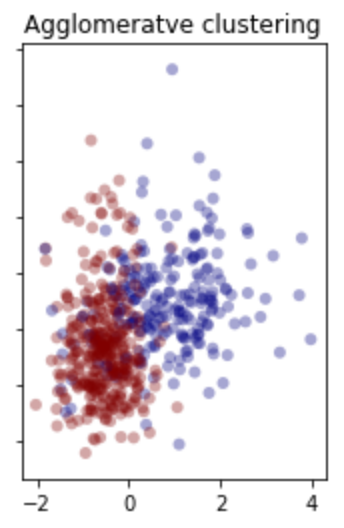
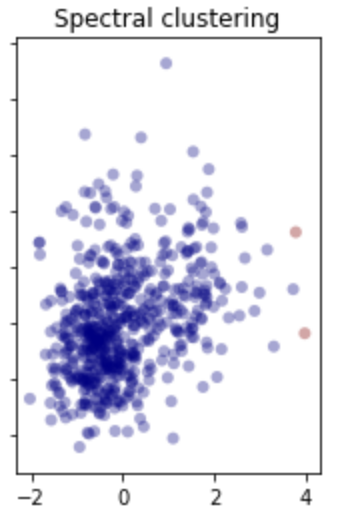
**   **

Fig 7: Different clusters on the data sets.

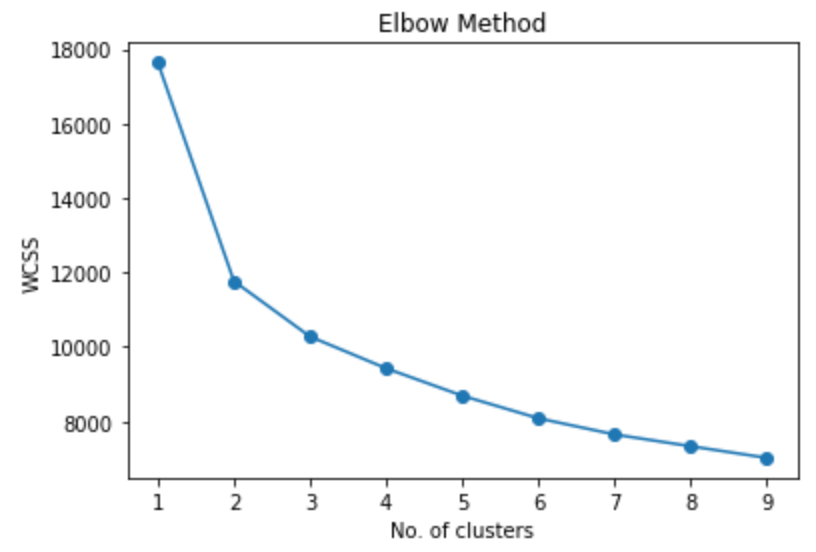
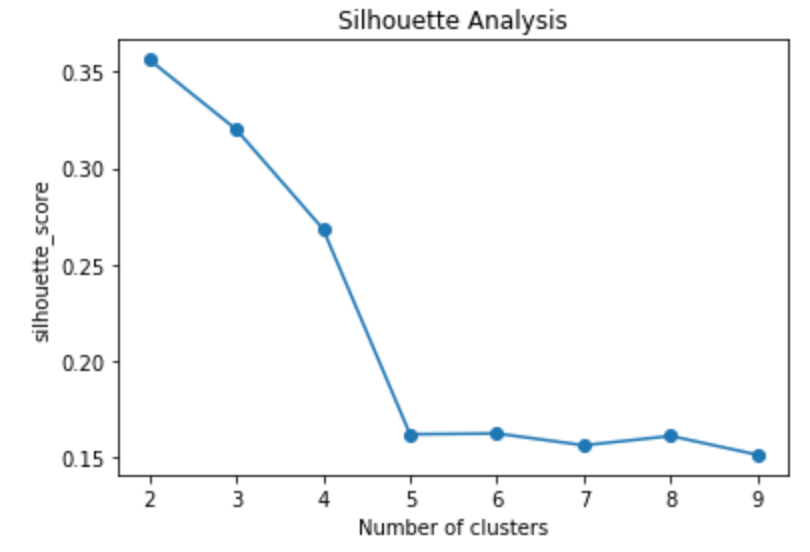
 

Fig 8: Using elbow method to find optimal no. of clusters fig 9: Silhouette analysis to find optimal no. of clusters

**K-Means:**

0.910369068541

precision recall f1-score support  
  
 0 0.93 0.83 0.87 212  
 1 0.90 0.96 0.93 357  
  
avg / total 0.91 0.91 0.91 569

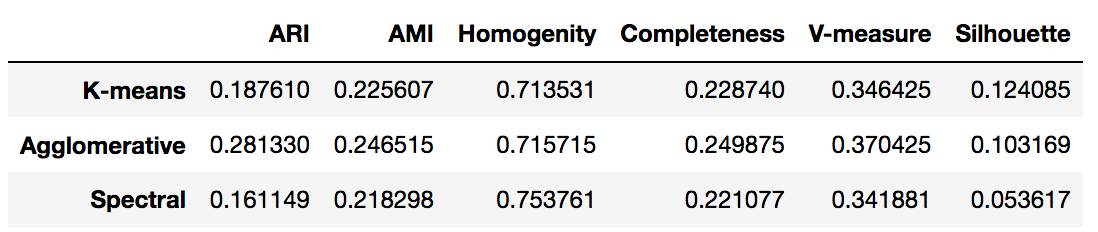
**Agglomerative Clustering**

0.880492091388

precision recall f1-score support  
  
 0 0.89 0.77 0.83 212  
 1 0.88 0.94 0.91 357  
  
avg / total 0.88 0.88 0.88 569

**Spectral Clustering:**

0.6309314586994728  
 precision recall f1-score support  
  
 0 0.63 1.00 0.77 357  
 1 1.00 0.01 0.02 212  
  
avg / total 0.77 0.63 0.49 569



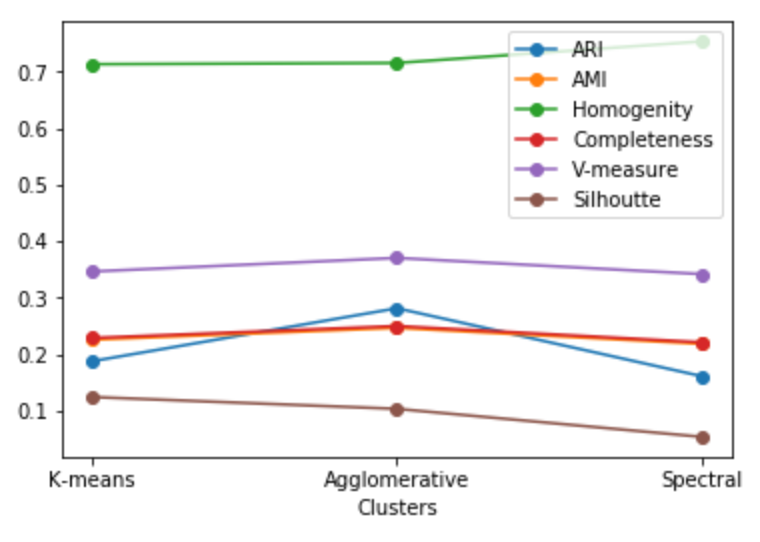


Fig 9: Clusters comparison

**Conclusion:**

All the classifiers discussed above performs reasonably well and Decision tree in particular performs best among them for this data set. Further hyper parameter tuning can be done to improve testing accuracy and overcome overfitting problems. Also, data set is visualized and analyzed using different clustering algorithms. From the analysis, K-Means clustering performed better for this data set.

**References:**

1. Breast Cancer Wisconsin (Diagnostic) Kaggle challenge: <https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/home>
2. UCI Machine Learning Repository: <https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)>
3. Sklearn data set: <http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html>
4. Sklearn Packages: [http://scikit-learn.org/](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html)
5. Jupyter Notebook: http://jupyter.org/

**Appendix:**

1. Correlation Heatmap for Breast Cancer Wisconsin (Diagnostic)

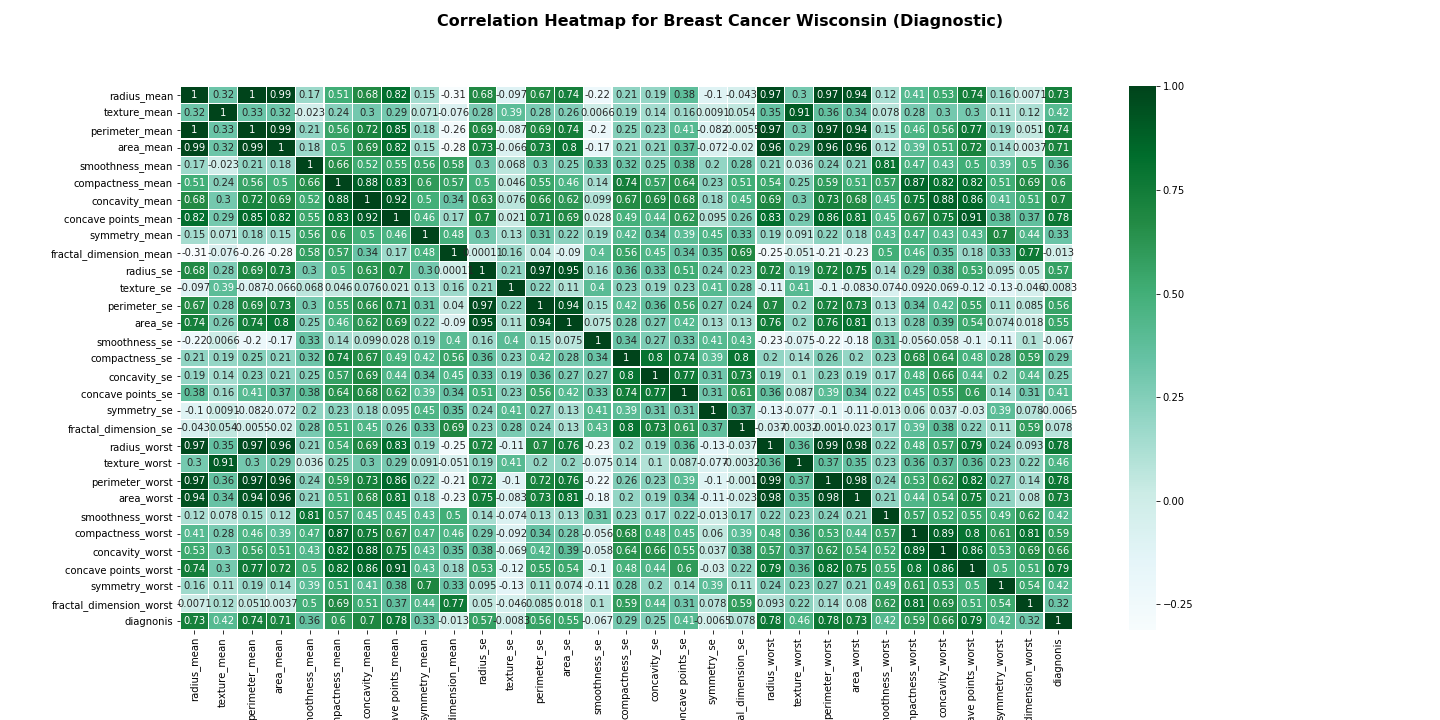


Fig: Correlation Heatmap for Breast Cancer Wisconsin (Diagnostic)

**Source Code:**

*# Breast Cancer Wisconsin (Diagnostic)  
# https://www.kaggle.com/uciml/breast-cancer-wisconsin-data*

*import numpy as np   
import pandas as pd  
import os  
from sklearn.preprocessing import StandardScaler  
import matplotlib.pyplot as plt  
%matplotlib inline*

*# Importing the Dataset  
try:  
 data = pd.read\_csv('data.csv')  
 print(data.head())  
 print('Breast Cancer Wisconsin (Diagnostic) Data set loaded.')  
except:  
 print('Sorry! Dataset could not be loaded.')*

*# Data set information  
data.info()*

*# Formatting data sets  
cols\_drop = ['id', 'Unnamed: 32']  
data = data.drop(cols\_drop, axis=1)*

*# Encode diagnosis label  
y = data.diagnosis  
  
# Scaling Data set  
X = data.drop('diagnosis', axis=1).values  
X = StandardScaler().fit\_transform(X)*

*# Data Visualization for data sets  
import seaborn as sns  
  
ax = sns.countplot(y,label="Count")   
B, M = y.value\_counts()  
print('No. of Benign: ',B)  
print('No. of Malignant : ',M)*

*# Correlation Heatmap for datasets  
figure, (a) = plt.subplots(1, 1, figsize=(20,10))  
  
hm = sns.heatmap(data.corr(),   
 ax=a,  
 cmap="BuGn",   
 annot=True,   
 linewidths=.1)  
  
figure.suptitle('Correlation Heatmap for Breast Cancer Wisconsin (Diagnostic)',   
 fontsize=16,   
 fontweight='bold');  
figure.savefig('heatmap.png')*

*# Classifiers  
# 1. KNN  
# 2. Logistic Regression  
# 3. Decision Tree  
# 4. Random Forest  
# 5. Neural Network MLP  
# 6. SVM*

*from sklearn.datasets import load\_breast\_cancer  
from sklearn.neighbors import KNeighborsClassifier   
from sklearn.linear\_model import LogisticRegression   
from sklearn.tree import DecisionTreeClassifier   
from sklearn.ensemble import RandomForestClassifier   
from sklearn.neural\_network import MLPClassifier   
from sklearn.svm import SVC   
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from sklearn.tree import export\_graphviz  
import matplotlib.pylab as plt  
import numpy as np  
%matplotlib inline*

*#load the breast cancer data and few EDA  
data = load\_breast\_cancer()  
#print(data.DESCR)*

*#----------------------------- KNN Classifier --------------------------------------  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, stratify=data.target, random\_state=66)  
  
training\_accuracy = []  
test\_accuracy = []  
  
  
neighbors\_setting = range(1,10)  
  
for n\_neighbors in neighbors\_setting:  
 knn = KNeighborsClassifier(n\_neighbors=n\_neighbors)  
 knn.fit(X\_train,y\_train)  
 training\_accuracy.append(knn.score(X\_train, y\_train))  
 test\_accuracy.append(knn.score(X\_test, y\_test))  
  
plt.plot(neighbors\_setting,training\_accuracy, marker='o', label='Accuracy of the training set')  
plt.plot(neighbors\_setting,test\_accuracy, marker='\*', label='Accuracy of the test set')  
plt.ylabel('Accuracy')  
plt.xlabel('Number of Neighbors')  
plt.legend()  
  
print("Accuracy of the training set for 6NN: {:3f}".format(training\_accuracy[5]))  
print("Accuracy of the test set for 6NN: {:3f}".format(test\_accuracy[5]))*

*knn*

*#-------------------------------------- Logistic Regression --------------------------------------  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, stratify=data.target, random\_state=42)  
  
log\_reg = LogisticRegression()  
log\_reg.fit(X\_train, y\_train)  
  
print('Accuracy on the training set: {:.3f}'.format(log\_reg.score(X\_train,y\_train)))  
print('Accuracy on the training set: {:.3f}'.format(log\_reg.score(X\_test,y\_test)))*

*log\_reg*

*#-------------------------------------- Decision Tree --------------------------------------  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, random\_state=42)  
  
training\_accuracy = []  
test\_accuracy = []  
  
max\_dep = range(1,10)  
  
for md in max\_dep:  
 tree = DecisionTreeClassifier(max\_depth=md,random\_state=0)  
 tree.fit(X\_train,y\_train)  
 training\_accuracy.append(tree.score(X\_train, y\_train))  
 test\_accuracy.append(tree.score(X\_test, y\_test))  
  
plt.plot(max\_dep,training\_accuracy, marker="o", label='Accuracy of the training set')  
plt.plot(neighbors\_setting,test\_accuracy, marker="\*",label='Accuracy of the test set')  
plt.ylabel('Accuracy')  
plt.xlabel('Max Depth')  
plt.legend()  
  
print("Accuracy of the training set for 3NN: {:3f}".format(training\_accuracy[2]))  
print("Accuracy of the test set for 3NN: {:3f}".format(test\_accuracy[2]))*

*tree*

*from matplotlib.pyplot import savefig  
  
#-------------------------------------- Feature Importance --------------------------------------  
n\_feature = data.data.shape[1]  
plt.figure(figsize=(4,8))  
plt.barh(range(n\_feature), tree.feature\_importances\_, 0.8, color="green", align='center')  
plt.yticks(np.arange(n\_feature), data.feature\_names)  
plt.xlabel('Feature Importance')  
plt.ylabel('Feature')  
plt.show()  
savefig('tree.png')*

*# -------------------------------------- Random Forests --------------------------------------  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, random\_state=0)  
  
forest = RandomForestClassifier(n\_estimators=100, random\_state=0)  
forest.fit(X\_train,y\_train)  
  
print('acc for training data: {:.3f}'.format(forest.score(X\_train,y\_train)))  
print('acc for test data: {:.3f}'.format(forest.score(X\_test,y\_test)))*

*forest*

*#-------------------------------------- Feature Importance --------------------------------------  
n\_feature = data.data.shape[1]  
plt.figure(figsize=(4,8))  
plt.barh(range(n\_feature), forest.feature\_importances\_, align='center')  
plt.yticks(np.arange(n\_feature), data.feature\_names)  
plt.xlabel('Feature Importance')  
plt.ylabel('Feature')  
plt.show()  
savefig('tree.png')*

*# -------------------------------------- Neural Network --------------------------------------  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, random\_state=0)  
  
scaler = StandardScaler()  
X\_train\_scaled = scaler.fit(X\_train).transform(X\_train)  
X\_test\_scaled = scaler.fit(X\_test).transform(X\_test)  
  
mlp = MLPClassifier(max\_iter=1000, alpha=1, random\_state=42)  
mlp.fit(X\_train\_scaled,y\_train)  
  
print('acc for training data: {:.3f}'.format(mlp.score(X\_train\_scaled, y\_train)))  
print('acc for test data: {:.3f}'.format(mlp.score(X\_test\_scaled, y\_test)))*

*mlp*

*# -------------------------------------- SVM (Support Vector Machine) --------------------------------------  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, random\_state=0)  
  
min\_train = X\_train.min(axis=0)  
range\_train = (X\_train - min\_train).max(axis=0)  
  
X\_train\_scaled = (X\_train - min\_train)/range\_train  
X\_test\_scaled = (X\_test - min\_train)/range\_train  
  
svm = SVC(C=1000)  
svm.fit(X\_train\_scaled, y\_train)  
print('acc on train dataset: {:.3f}'.format(svm.score(X\_train\_scaled,y\_train)))  
print('acc on test dataset: {:.3f}'.format(svm.score(X\_test\_scaled,y\_test)))*

*svm*

*# Clustering  
# 1. KMeans Clustering  
# 2. Hierarchical Agglomerative Clustering   
# 3. Spectral Clustering*

*import numpy as np   
import pandas as pd  
import os  
from sklearn.preprocessing import StandardScaler  
import matplotlib.pyplot as plt  
%matplotlib inline*

*# Importing the Dataset  
try:  
 data = pd.read\_csv('data.csv')  
 print(data.head())  
 print('Breast Cancer Wisconsin (Diagnostic) Data set loaded.')  
except:  
 print('Sorry! Dataset could not be loaded.')*

*# -------------------------------------- Preprocessing data set --------------------------------------  
  
# Drop unnecessary columns  
cols\_drop = ['id', 'Unnamed: 32']  
data = data.drop(cols\_drop, axis=1)  
  
# Encode diagnosis label  
data['diagnonis'] = data['diagnosis'].map({'M':1,'B':0})  
  
# Featureset creation  
X = data.drop('diagnosis', axis=1).values  
X = StandardScaler().fit\_transform(X)*

*#-------------------------------------- KMeans Clustering --------------------------------------  
  
from sklearn.cluster import KMeans  
km = KMeans(n\_clusters=2, init="k-means++", n\_init=10)  
km\_pred = km.fit\_predict(X)  
  
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)  
  
ax1.scatter(X[:,0], X[:,1], c=data["diagnosis"], cmap="jet", edgecolor="None", alpha=0.35)  
ax1.set\_title("Actual clusters")  
  
ax2.scatter(X[:,0], X[:,1], c=km\_pred, cmap="jet", edgecolor="None", alpha=0.35)  
ax2.set\_title("KMeans clustering")*

*#-------------------------------------- Hierarchical Agglomerative Clustering --------------------------------------  
  
from sklearn.cluster import AgglomerativeClustering  
ac = AgglomerativeClustering(n\_clusters=2, linkage="ward")  
ac\_pred = ac.fit\_predict(X)  
  
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)  
  
ax1.scatter(X[:,0], X[:,1], c=data["diagnosis"], cmap="jet", edgecolor="None", alpha=0.35)  
ax1.set\_title("Actual clusters")  
  
ax2.scatter(X[:,0], X[:,1], c=ac\_pred, cmap="jet", edgecolor="None", alpha=0.35)  
ax2.set\_title("Agglomeratve clustering")*

*#-------------------------------------- Spectral Clustering --------------------------------------  
  
from sklearn.cluster import SpectralClustering  
sc = SpectralClustering(n\_clusters=2, gamma=0.5, affinity="rbf", assign\_labels="discretize")  
sc\_pred = sc.fit\_predict(X)  
  
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)  
  
ax1.scatter(X[:,0], X[:,1], c=data["diagnosis"], cmap="jet", edgecolor="None", alpha=0.35)  
ax1.set\_title("Actual clusters")  
  
ax2.scatter(X[:,0], X[:,1], c=sc\_pred, cmap="jet", edgecolor="None", alpha=0.35)  
ax2.set\_title("Spectral clustering")*