Student Name: Arina Sheredekina

Student ID: 300383102

Instructor Name: Parissa Ahmadi

Class Name: Fundamental of Machine Learning (CSIS 3290)

Date and Time of the exam: Nov 01, 2024 12:30- 3:00 PM

Task1:

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn import datasets
from scipy.stats import pearsonr
from scipy.stats import spearmanr
from scipy.stats import chi2_contingency
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
```

[3]:
```
data1=datasets.load_breast_cancer()
```

Print the dataset dimensions

```
data1.data.shape
```

```
(569, 30)
```

Print the column's name

```
data1.feature_names
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       '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')
```

Print the target values

```
data1.target
```

```
array([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, 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, 0, 1, 1, 0, 1, 0, 1, 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, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 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, 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, 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])
```

Convert it to the dataframe

```
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

```
AriDF=pd.DataFrame(data1.data,columns=data1.feature_names)
AriDF['target']=data1.target
```

Build your train and test sets according to 75-25 percent of split ratio

```
x=data1.data
y=data1.target
x_train,x_test,y_train,y_test= train_test_split(x,y, test_size=0.25, random_state=42, stratify=y)
```

Print the correlation among all the attributes

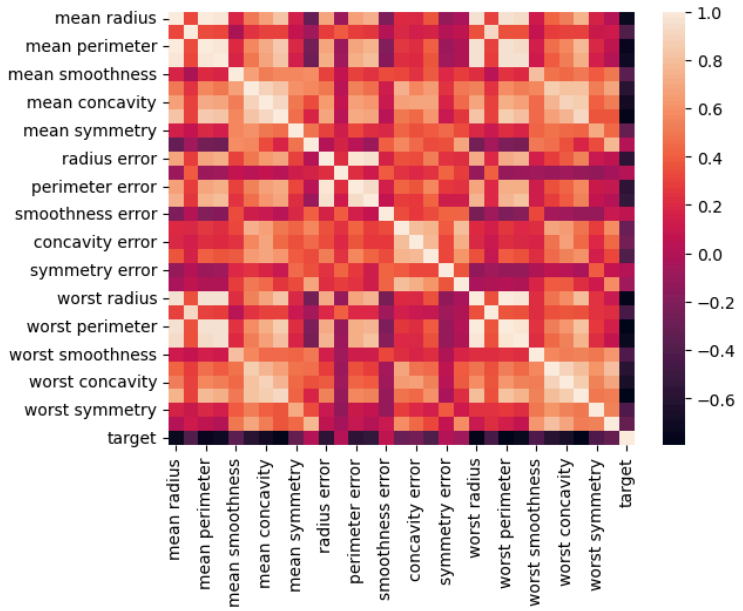
```
15]: AriDF.corr()
```

15]:

	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	worst area	sm
mean radius	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529	0.147741	-0.311631	...	0.297008	0.965137	0.941082	
mean texture	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.293464	0.071401	-0.076437	...	0.912045	0.358040	0.343546	
mean perimeter	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.850977	0.183027	-0.261477	...	0.303038	0.970387	0.941550	
mean area	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823269	0.151293	-0.283110	...	0.287489	0.959120	0.959213	
mean smoothness	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.553695	0.557775	0.584792	...	0.036072	0.238853	0.206718	
mean compactness	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135	0.602641	0.565369	...	0.248133	0.590210	0.509604	
mean concavity	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.921391	0.500667	0.336783	...	0.299879	0.729565	0.675987	
mean concave points	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.000000	0.462497	0.166917	...	0.292752	0.855923	0.809630	
mean symmetry	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.462497	1.000000	0.479921	...	0.090651	0.219169	0.177193	
mean fractal dimension	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.166917	0.479921	1.000000	...	-0.051269	-0.205151	-0.231854	
radius error	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925	0.698050	0.303379	0.000111	...	0.194799	0.719684	0.751548	
texture error	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218	0.021480	0.128053	0.164174	...	0.409003	-0.102242	-0.083195	
perimeter error	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391	0.710650	0.313893	0.039830	...	0.200371	0.721031	0.730713	
area error	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427	0.690299	0.223970	-0.090170	...	0.196497	0.761213	0.811408	
smoothness error	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564	0.027653	0.187321	0.401964	...	-0.074743	-0.217304	-0.182195	
compactness error	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279	0.490424	0.421659	0.559837	...	0.143003	0.260516	0.199371	
concavity error	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270	0.439167	0.342627	0.446630	...	0.100241	0.226680	0.188353	
concave points error	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260	0.615634	0.393298	0.341198	...	0.086741	0.394999	0.342271	
symmetry error	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009	0.095351	0.449137	0.345007	...	-0.077473	-0.103753	-0.110343	
fractal dimension error	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301	0.257584	0.331786	0.688132	...	-0.003195	-0.001000	-0.022736	
worst radius	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688236	0.830318	0.185728	-0.253691	...	0.359921	0.993708	0.984015	
worst texture	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299879	0.292752	0.090651	-0.051269	...	1.000000	0.365098	0.345842	
worst perimeter	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729565	0.855923	0.219169	-0.205151	...	0.365098	1.000000	0.977578	
worst area	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675987	0.809630	0.177193	-0.231854	...	0.345842	0.977578	1.000000	
worst															

Plot the heatmap for the result

```
[19]: sb.heatmap(AriDF.corr())
      plt.show()
```




```
[ ]:
```

Plot mean radius versus mean perimeter attributes and then mean radius versus worst symmetry attributes

Task2:

Implement the Decision Tree Classifier

```
|: df_classsifier=DecisionTreeClassifier()
```

```
df_classsifier.fit(x_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Calculate the accuracy for the Decision Tree Classifier

```
accuracy_score(y_test, pre1)
```

```
0.9230769230769231
```

```
df_classsifier.score(x_test,y_test)
```

```
0.9230769230769231
```

Show the confusion matrix

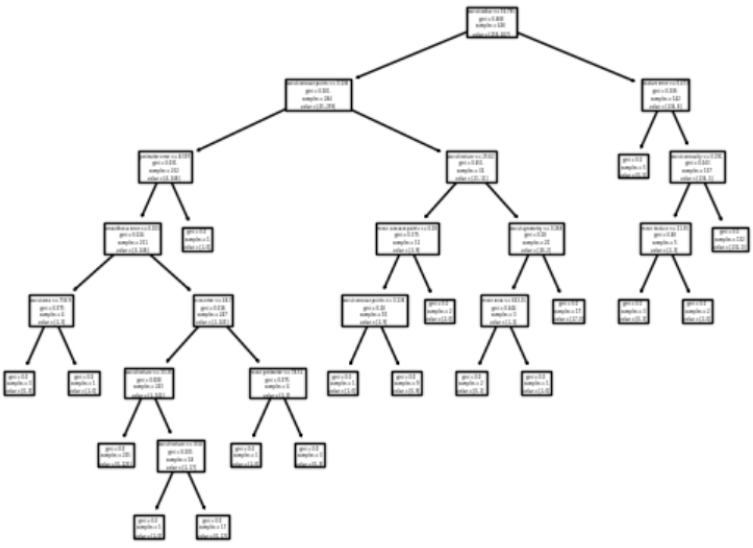
```
pre1=df_classsifier.predict(x_test)
```

```
confusion_matrix(y_test,pre1)
```

```
array([[49,  4],
       [ 7, 83]], dtype=int64)
```

Draw the tree

```
tree.plot_tree(df_classsifier, feature_names=f1)
plt.show()
```



Perform the classification with KNN
Assign value=2

```
knn1=KNeighborsClassifier(n_neighbors=2, metric='minkowski' , p=2)
```

```
knn1.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=2)
```

```
pre2=knn1.predict(x_test)
```

Calculate the accuracy

```
accuracy_score(y_test,pre2)
```

```
0.9300699300699301
```

```
knn1.score(x_test,y_test)
```

```
0.9300699300699301
```

Show the confusion matrix

```
confusion_matrix(y_test,pre2)
```

```
array([[49,  4],
       [ 6, 84]], dtype=int64)
```

Assign value=10

```
knn2=KNeighborsClassifier(n_neighbors=10, metric='minkowski' , p=2)
```

```
knn2.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=10)
```

```
pre3=knn2.predict(x_test)
```

Calculate the accuracy

```
knn2.score(x_test,y_test)
```

```
0.9440559440559441
```

```
accuracy_score(y_test,pre3)
```

```
0.9440559440559441
```

Show the confusion matrix

```
confusion_matrix(y_test,pre3)
```

```
array([[48,  5],
       [ 3, 87]], dtype=int64)
```

Task3:
Implement the Random Forest Classifier

```
RF1=RandomForestRegressor(n_estimators=100, random_state=0)
```

```
RF1.fit(x_train,y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=0)
```

```
preRF1=RF1.predict(x_test)
```

```
RF1.score(x_test,y_test)
```

```
0.8491065199161426
```

Max depth of the tree to 4
Calculate the accuracy

```
RF2=RandomForestClassifier(max_depth=4, oob_score=True)
RF2.fit(x_train,y_train)
preRF2=RF2.predict(x_test)
print(RF2.score(x_test,y_test))
print(RF2.oob_score_)
```

```
0.958041958041958
0.9577464788732394
```

Show the confusion matrix

```
confusion_matrix(y_test,preRF2 )
```

```
array([[49,  4],
       [ 2, 88]], dtype=int64)
```

Max depth of the tree to 6
Calculate the accuracy

```
RF3=RandomForestClassifier(max_depth=6, oob_score=True)
RF3.fit(x_train,y_train)
preRF3=RF3.predict(x_test)
print(RF3.score(x_test,y_test))
print(RF3.oob_score_)
```

```
0.958041958041958
0.960093896713615
```

Show the confusion matrix

```
confusion_matrix(y_test,preRF3 )
```

```
array([[49,  4],
       [ 2, 88]], dtype=int64)
```

Implement the Ada Boost Classifier

```
Ada1=AdaBoostClassifier(n_estimators=50, learning_rate=0.2)
```

```
Ada1.fit(x_train,y_train)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:5
will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

```
▾ AdaBoostClassifier
AdaBoostClassifier(learning_rate=0.2)
```

Calculate the accuracy

```
score=Ada1.score(x_test,y_test)
```

```
score
```

```
0.965034965034965
```

Show the confusion matrix

```
confusion_matrix(y_test,preAda1)
```

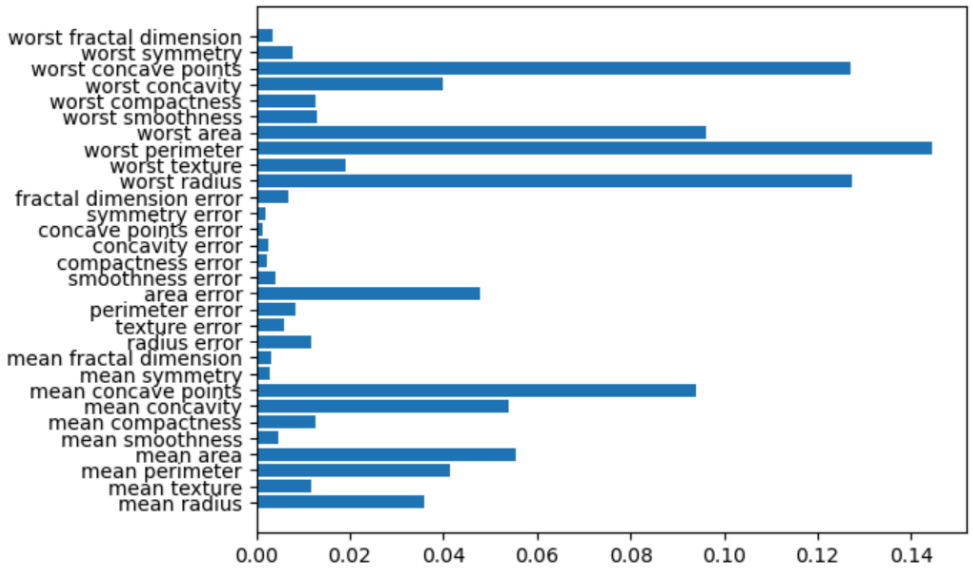
```
array([[49,  4],
       [ 1, 89]], dtype=int64)
```

Calculate the importance of features for Random Forest Classifier and plot the importance for Random Forest Classifier
For max_depth=4

```
print(RF2.feature_importances_)
```

```
[0.03602236 0.01181593 0.04136062 0.05550365 0.00484291 0.01251437
 0.05393251 0.09398849 0.00301376 0.00305514 0.01174606 0.00584955
 0.00822853 0.04784856 0.0042107  0.00212588 0.00251804 0.00147469
 0.00193384 0.0067148  0.12729295 0.01922196 0.1444315  0.09622372
 0.01304152 0.01278856 0.03972711 0.12715638 0.00782665 0.00358923]
```

```
plt.barh(data1.feature_names, RF2.feature_importances_)
plt.show()
```

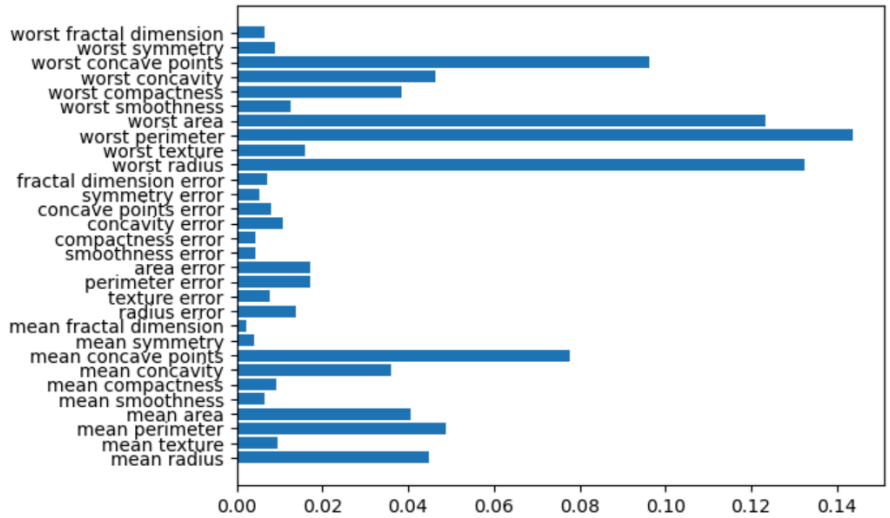


For max_depth=6

```
print(RF3.feature_importances_)
```

```
[0.04496842 0.00951554 0.04886318 0.0405767  0.00649184 0.00918764
 0.03587758 0.077769   0.00393998 0.00218062 0.01378557 0.0076613
 0.01722924 0.01720311 0.00442925 0.00444889 0.01085879 0.00816564
 0.0052033  0.00701655 0.13232353 0.01607473 0.14370419 0.12345218
 0.01248335 0.03843556 0.04637209 0.09610915 0.00903902 0.00663405]
```

```
plt.barh(data1.feature_names, RF3.feature_importances_)
plt.show()
```



Calculate the importance of features for AdaBoost Classifier and plot the importance for AdaBoost Classifier

```
importances=Ada1.feature_importances_
```

```
Sorted=pd.Series(importances).sort_values()
```

```
print(Sorted)
```

```
0     0.00
25    0.00
19    0.00
18    0.00
17    0.00
16    0.00
12    0.00
11    0.00
9     0.00
29    0.00
2     0.00
6     0.00
8     0.00
5     0.00
26    0.02
24    0.02
20    0.02
3     0.02
14    0.02
28    0.02
15    0.06
1     0.06
7     0.06
10    0.06
22    0.08
13    0.08
27    0.10
4     0.10
21    0.14
23    0.14
dtype: float64
```

```
dtype: float64
```

```
plt.barh(data1.feature_names, Ada1.feature_importances_)
plt.show()
```