Machine Learning Project

Breast Cancer Prediction Dataset

Butch Adrian Castro - 180995

CSCI 111

Overview of the Dataset

Diagnosis of breast cancer is performed when an abnormal lump is found. The doctor will then conduct a diagnosis to determine whether it is cancerous and, if so, whether it has spread to other parts of the body.

This breast cancer dataset was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg.

Source:

https://www.kaggle.com/datasets/merishnasuwal/breast-cancer-prediction-dataset

Features of Dataset

mean_perimeter -> mean size of the core tumor

mean_area -> mean area of the core tumor

mean_radius -> mean of distances from center to points on the perimeter

mean_texture -> standard deviation of gray-scale values

mean_smoothness -> mean of local variation in radius lengths

diagnosis -> the diagnosis of breast tissues where 1 indicates that the tumor is harmful and 0 if not

Dataset Preparation

Preparing the Dataset, removed label column

```
# Load breast cancer dataset
D1 = pd.read_csv("Breast_cancer_data.csv", index_col=None)
y = D1["diagnosis"]
D1 = D1.drop(columns=["diagnosis"])
```

✓ 0.0s

D1 - Original

D1 - Original Dataset

```
D1
```

✓ 0.0s

	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

569 rows × 5 columns

D2 - MinMax

D2 - MinMax Scaled Dataset

```
D2 = MinMaxScaler().fit_transform(D1)
D2 = pd.DataFrame(D2,columns=D1.columns)
D2
```

✓ 0.0s

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness
0	0.521037	0.022658	0.545989	0.363733	0.593753
1	0.643144	0.272574	0.615783	0.501591	0.289880
2	0.601496	0.390260	0.595743	0.449417	0.514309
3	0.210090	0.360839	0.233501	0.102906	0.811321
4	0.629893	0.156578	0.630986	0.489290	0.430351
...	...	...	...	...	...
564	0.690000	0.428813	0.678668	0.566490	0.526948
565	0.622320	0.626987	0.604036	0.474019	0.407782
566	0.455251	0.621238	0.445788	0.303118	0.288165
567	0.644564	0.663510	0.665538	0.475716	0.588336
568	0.036869	0.501522	0.028540	0.015907	0.000000

D3 - Standard

D3 - Standard Scaled Dataset

```
D3 = StandardScaler().fit_transform(D1)
D3 = pd.DataFrame(D3,columns=D1.columns)
D3
```

✓ 0.0s

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness
0	1.097064	-2.073335	1.269934	0.984375	1.568466
1	1.829821	-0.353632	1.685955	1.908708	-0.826962
2	1.579888	0.456187	1.566503	1.558884	0.942210
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553
4	1.750297	-1.151816	1.776573	1.826229	0.280372
...	...	...	...	...	...
564	2.110995	0.721473	2.060786	2.343856	1.041842
565	1.704854	2.085134	1.615931	1.723842	0.102458
566	0.702284	2.045574	0.672676	0.577953	-0.840484
567	1.838341	2.336457	1.982524	1.735218	1.525767
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085

569 rows × 5 columns

sklearn.preprocessing.MinMaxScaler & sklearn.preprocessing.StandardScaler to normalize the values

Classification - kNN

75% train- 25% test split used

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
X_train_d1, X_test_d1, y_train_d1, y_test_d1 = train_test_split(D1, y)
```
✓ 0.0s

kNN Accuracy and Confusion Matrix for D3

```
acc3_d3 = knn3_d3.score(X_test_d3,y_test_d3)
cm3_d3 = confusion_matrix(y_test_d3,knn3_d3_prediction)

print("kNN, k = 3 accuracy: ",acc3_d3)
print("Confusion Matrix:")
print(cm3_d3)

precision_d3 = precision_score(y_test_d3,knn3_d3_prediction)
recall_d3 = recall_score(y_test_d3,knn3_d3_prediction)
print("Precision:", precision_d3)
print("Recall:", recall_d3)
```

Sklearn.metrics.precision_score & recall_score used

D1

```
kNN, k = 3 accuracy:  0.8601398601398601
Confusion Matrix:
[[36 12]
 [ 8 87]]
Precision: 0.8787878787878788
Recall: 0.9157894736842105
```

D2

```
kNN, k = 3 Accuracy:  0.8881118881118881
Confusion Matrix:
[[45  7]
 [ 9 82]]
Precision: 0.9213483146067416
Recall: 0.9010989010989011
```

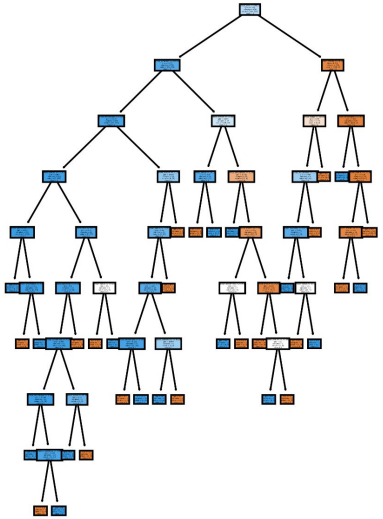
D3

```
kNN, k = 3 accuracy:  0.9090909090909091
Confusion Matrix:
[[40  8]
 [ 5 90]]
Precision: 0.9183673469387755
Recall: 0.9473684210526315
```

Also 75% train- 25% test split used

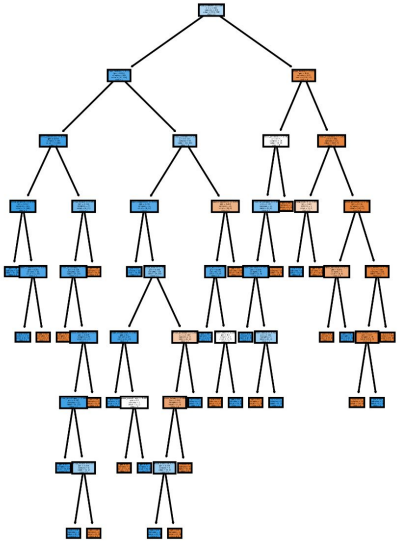
Classification - Decision Trees

D1



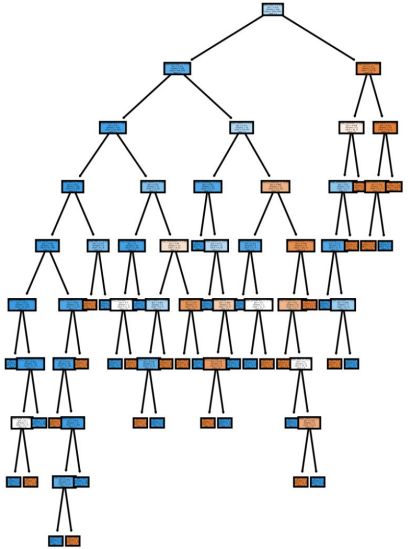
```
Accuracy:  0.8741258741258742
Confusion Matrix:
[[41  7]
 [11 84]]
Precision: 0.9230769230769231
Recall: 0.8842105263157894
```

D2



```
Accuracy:  0.8951048951048951
Confusion Matrix:
[[46  6]
 [ 9 82]]
Precision: 0.9318181818181818
Recall: 0.9010989010989011
```

D3



```
Accuracy:  0.9300699300699301
Confusion Matrix:
[[44  4]
 [ 6 89]]
Precision: 0.956989247311828
Recall: 0.9368421052631579
```

I think that the best model for this is kNN, with a k=3, because of its consistently high recall values. Having a high recall value indicates the high chance of getting a low number of false negatives, which would be much more appropriate because missing the early predictors of cancer is much more devastating to the patient than mistaking them to have cancer but when they really don't.

Clustering: Finding K

Used Silhouette Score method to find the best K value (which every set used 2)

```
k_range_d1 = range(2, 15)

highest_score_d1 = -1
highest_k_d1 = -1

for k in k_range_d1:
    km_model = KMeans(n_clusters=k,n_init='auto',random_state=0)
    km_model.fit(D1)
    km_labels = km_model.predict(D1)
    avg = silhouette_score(D1, km_labels)
    print((k, round(avg, 4)))

    if avg > highest_score_d1:
        highest_score_d1 = avg
        highest_k_d1 = k

print("K = " + str(highest_k_d1))
print("Score = " + str(highest_score_d1))
```

```
(2, 0.6991)
(3, 0.6627)
(4, 0.5588)
(5, 0.5443)
(6, 0.5334)
(7, 0.5361)
(8, 0.5515)
(9, 0.5433)
(10, 0.5274)
(11, 0.5282)
(12, 0.5279)
(13, 0.5295)
(14, 0.5428)
K = 2
Score = 0.699135645499053
```

Clustering: K-Means

Used K-Means with the best value K

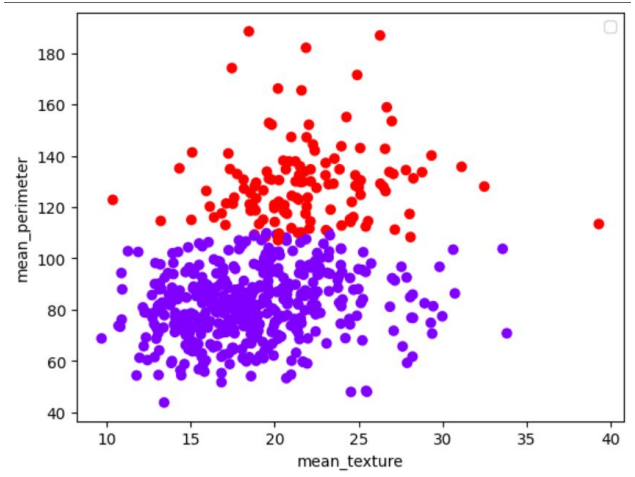
```
kmeans_d1 = KMeans(n_clusters=highest_k_d1,n_init=10,random_state=0).fit(D1)
kmeans_d1.fit(D1)
```

Chose mean_perimeter and mean_texture as the most relevant columns to showcase the clustering

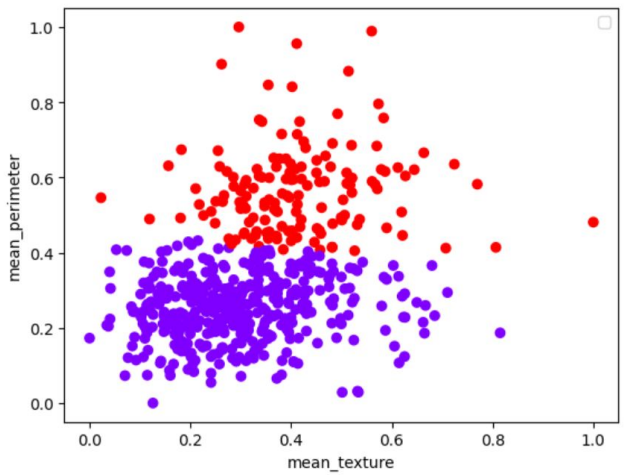
```
plt.scatter(D1["mean_texture"], D1["mean_perimeter"], c=kmeans_d1.labels_, cmap="rainbow")
plt.xlabel("mean_texture")
plt.ylabel("mean_perimeter")
plt.legend()
plt.show()
```

Scatter Plots

D1



D2



D3