Allen Ben Philipose - 18BIS0043

Lab FAT, L57+L58

ECE3047 - Machine Learning Fundamentals

Submitted to: Prof. Sankar Ganesh

Aim

- Analyze the performance of KNN by choosing 2 different data sets.
- Train and Test KNN classifier using the cancer dataset for K=3,4,5. Calculate the result using three performance metrics.

Code and Result

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
from sklearn import preprocessing
from sklearn.metrics import accuracy_score, log_loss, confusion_matrix, f1_score
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import StratifiedKFold, GridSearchCV, KFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
```

In [3]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
```

In [4]:
```
import warnings
warnings.filterwarnings('ignore')
```

Dataset

In [5]:
```
allen = pd.read_csv("Dataset.csv") # The assigned Dataset - Cancer
```

In [6]:
```
compare = pd.read_csv("Comparison.csv") # Dataset downloaded for comparison - Diabetes
```

In [7]:
```
allen.head()
```

Out[7]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 33 columns

In [8]:
```
allen.info
```

Out[8]:
```
<bound method DataFrame.info of           id diagnosis  radius_mean  texture_mean  perimeter_mean  a
rea_mean  \
0      842302         M        17.99         10.38          122.80     1001.0
1      842517         M        20.57         17.77          132.90     1326.0
2    84300903         M        19.69         21.25          130.00     1203.0
3    84348301         M        11.42         20.38           77.58      386.1
4    84358402         M        20.29         14.34          135.10     1297.0
..        ...       ...          ...           ...             ...        ...
564    926424         M        21.56         22.39          142.00     1479.0
565    926682         M        20.13         28.25          131.20     1261.0
566    926954         M        16.60         28.08          108.30      858.1
567    927241         M        20.60         29.33          140.10     1265.0
568     92751         B         7.76         24.54           47.92      181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710
1            0.08474           0.07864         0.08690              0.07017
2            0.10960           0.15990         0.19740              0.12790
3            0.14250           0.28390         0.24140              0.10520
4            0.10030           0.13280         0.19800              0.10430
..               ...               ...             ...                  ...
564          0.11100           0.11590         0.24390              0.13890
565          0.09780           0.10340         0.14400              0.09791
566          0.08455           0.10230         0.09251              0.05302
567          0.11780           0.27700         0.35140              0.15200
568          0.05263           0.04362         0.00000              0.00000

     ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0    ...          17.33           184.60      2019.0           0.16220
1    ...          23.41           158.80      1956.0           0.12380
2    ...          25.53           152.50      1709.0           0.14440
3    ...          26.50            98.87       567.7           0.20980
4    ...          16.67           152.20      1575.0           0.13740
..   ...            ...              ...         ...               ...
564  ...          26.40           166.10      2027.0           0.14100
565  ...          38.25           155.00      1731.0           0.11660
566  ...          34.12           126.70      1124.0           0.11390
567  ...          39.42           184.60      1821.0           0.16500
568  ...          30.37            59.16       268.6           0.08996

     compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0              0.66560           0.7119                0.2654          0.4601
1              0.18660           0.2416                0.1860          0.2750
2              0.42450           0.4504                0.2430          0.3613
3              0.86630           0.6869                0.2575          0.6638
4              0.20500           0.4000                0.1625          0.2364
..                 ...              ...                   ...             ...
564            0.21130           0.4107                0.2216          0.2060
565            0.19220           0.3215                0.1628          0.2572
566            0.30940           0.3403                0.1418          0.2218
567            0.86810           0.9387                0.2650          0.4087
568            0.06444           0.0000                0.0000          0.2871

     fractal_dimension_worst  Unnamed: 32
0                    0.11890          NaN
1                    0.08902          NaN
2                    0.08758          NaN
3                    0.17300          NaN
4                    0.07678          NaN
..                       ...          ...
564                  0.07115          NaN
565                  0.06637          NaN
566                  0.07820          NaN
567                  0.12400          NaN
568                  0.07039          NaN

[569 rows x 33 columns]>
```

In [9]:
```
allen.describe()
```

Out[9]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800

8 rows × 32 columns

In [10]:
```
compare.head()
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [12]:
```
compare.info
```

Out[12]:
```
<bound method DataFrame.info of      Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BM
I  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]>
```

In [11]:
```
compare.describe()
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.00000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.34895
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.47695
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.00000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.00000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.00000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.00000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.00000

Pre-processing

Label Encoding

In [14]:
```
l1 = preprocessing.LabelEncoder()
f1 = l1.fit_transform(allen['diagnosis'])
f1 = pd.DataFrame(data=f1, columns=['diagnosis'])
allen['diagnosis'] = f1['diagnosis']
```

In [16]:
```
print(list(l1.inverse_transform([0,1])))
```

```
['B', 'M']
```

Dropping unwanted Columns

In [19]:
```
allen = allen.drop(['id','Unnamed: 32'],axis=1)
```

Final Look

In [23]:
```
allen.head()
```

Out[23]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	conc points_m
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10

5 rows × 31 columns

In [24]:
```
compare.head()
```

Out[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Train-test-split - I

In [25]:
```
x1 = allen.drop(['diagnosis'],axis=1)
```

In [26]:
```
y1 = allen['diagnosis']
```

In [27]:
```
xtrain1, xtest1, ytrain1, ytest1 = train_test_split(x1,y1,test_size=0.1,random_state=21)
```

In [28]:
```
x2 = compare.drop(['Outcome'],axis=1)
```

In [29]:
```
y2 = compare['Outcome']
```

In [30]:
```
xtrain2, xtest2, ytrain2, ytest2 = train_test_split(x2,y2,test_size=0.1,random_state=21)
```

Train-test-split - II

In [68]:
```
x = allen.drop(['diagnosis'],axis=1)
```

In [69]:
```
y = allen['diagnosis']
```

In [70]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x,y,test_size=0.1,random_state=21)
```

Model - I

In [52]:
```
modelA = KNeighborsClassifier()
modelA.fit(xtrain1,ytrain1)
pA = modelA.predict(xtest1)
```

In [53]:
```
modelB = KNeighborsClassifier()
modelB.fit(xtrain2,ytrain2)
pB = modelB.predict(xtest2)
```

Evaluation I - Dataset I (Cancer)

In [54]:
```
print(classification_report(ytest1.values.ravel(),pA,target_names=['0','1']))
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        41
           1       0.81      0.81      0.81        16

    accuracy                           0.89        57
   macro avg       0.87      0.87      0.87        57
weighted avg       0.89      0.89      0.89        57
```

In [56]:
```
print(confusion_matrix(ytest1,pA))
```

```
[[38  3]
 [ 3 13]]
```

In [57]:
```
cA = confusion_matrix(ytest1,pA)
print('Specificity: ', cA[1,1]/(cA[1,0]+cA[1,1]))
print('Sensitivity: ', cA[0,0]/(cA[0,0]+cA[0,1]))
```

```
Specificity:  0.8125
Sensitivity:  0.926829268292683
```

In [58]:
```
print('Accuracy: {:.2%}'.format((cA[0,0]+cA[1,1])/sum(sum(cA))))
```

```
Accuracy: 89.47%
```

In [60]:
```
f1A = f1_score(ytest1,pA)
print('F1 Score: {:.2%}'.format(f1A))
```

```
F1 Score: 81.25%
```

Evaluation I - Dataset II (Diabetes)

In [62]:
```
print(classification_report(ytest2.values.ravel(),pB,target_names=['0','1']))
```

```
              precision    recall  f1-score   support

           0       0.71      0.85      0.78        47
           1       0.67      0.47      0.55        30

    accuracy                           0.70        77
   macro avg       0.69      0.66      0.66        77
weighted avg       0.70      0.70      0.69        77
```

In [63]:
```
print(confusion_matrix(ytest2,pB))
```

```
[[40  7]
 [16 14]]
```

In [64]:
```
cB = confusion_matrix(ytest2,pB)
print('Specificity: ', cB[1,1]/(cB[1,0]+cB[1,1]))
print('Sensitivity: ', cB[0,0]/(cB[0,0]+cB[0,1]))
```

```
Specificity:  0.4666666666666667
Sensitivity:  0.851063829787234
```

In [65]:
```
print('Accuracy: {:.2%}'.format((cB[0,0]+cB[1,1])/sum(sum(cB))))
```

```
Accuracy: 70.13%
```

In [66]:
```
f1B = f1_score(ytest2,pB)
print('F1 Score: {:.2%}'.format(f1B))
```

```
F1 Score: 54.90%
```

Inference - I

We can understand that the model from the Dataset 1 is giving **much** better performance values that Dataset 2.

- Accuracy decreased in the second dataset because of the variance of values which causes underfitting of the model.
- F1 score, specificity and sensitivity can be derived from the confusion matrix of each, and even all those parameters show lesser value in the second dataset.

Model - II

In [102]:
```
modelC = KNeighborsClassifier(n_neighbors=3)
modelC.fit(xtrain1,ytrain1)
pC = modelC.predict(xtest1)
```

In [103]:
```
modelD = KNeighborsClassifier(n_neighbors=4)
modelD.fit(xtrain1,ytrain1)
pD = modelD.predict(xtest1)
```

In [104]:
```
modelE = KNeighborsClassifier(n_neighbors=5)
modelE.fit(xtrain1,ytrain1)
pE = modelE.predict(xtest1)
```

Evaluation - II

In [109]:
```
cC = confusion_matrix(ytest,pC)
k1 = pd.DataFrame({'Model':'KNN (n=3)',
                   'Accuracy':[(cC[0,0]+cC[1,1])/sum(sum(cC))],
                   'Specificity':[cC[1,1]/(cC[1,0]+cC[1,1])],
                   'Sensitivity':[cC[0,0]/(cC[0,0]+cC[0,1])],
                   'F1 Score':[f1_score(ytest,pC)]
})
```

In [111]:
```
cD = confusion_matrix(ytest,pD)
k2 = pd.DataFrame({'Model':'KNN (n=4)',
                   'Accuracy':[(cD[0,0]+cD[1,1])/sum(sum(cD))],
                   'Specificity':[cD[1,1]/(cD[1,0]+cD[1,1])],
                   'Sensitivity':[cD[0,0]/(cD[0,0]+cD[0,1])],
                   'F1 Score':[f1_score(ytest,pD)]
})
```

In [112]:
```
cE = confusion_matrix(ytest,pE)
k3 = pd.DataFrame({'Model':'KNN (n=5)',
                   'Accuracy':[(cE[0,0]+cE[1,1])/sum(sum(cE))],
                   'Specificity':[cE[1,1]/(cE[1,0]+cE[1,1])],
                   'Sensitivity':[cE[0,0]/(cE[0,0]+cE[0,1])],
                   'F1 Score':[f1_score(ytest,pE)]
})
```

In [114]:
```
k = pd.concat([k1,k2,k3],axis=0).reset_index()
k = k.drop('index',axis=1)
k
```

Out[114]:

	Model	Accuracy	Specificity	Sensitivity	F1 Score
0	KNN (n=3)	0.894737	0.7500	0.951220	0.800000
1	KNN (n=4)	0.929825	0.7500	1.000000	0.857143
2	KNN (n=5)	0.894737	0.8125	0.926829	0.812500

Inference - II

Hence from the experiment we have figured out that the highest performance is received when the value of K = 4 by analysing the performance metrics such as Accuracy, sensitivity which are derived from the confusion matrix and F1 score giving an overall analysed score of the model performance.

Evaluation - Additional

How are other algorithms doing when compared to KNN

Random Forest

In [71]:
```
model1 = RandomForestClassifier()
model1.fit(xtrain,ytrain)
p1 = model1.predict(xtest)
```

In [72]:
```
print(confusion_matrix(ytest,p1))
```

```
[[40  1]
 [ 2 14]]
```

In [73]:
```
print(classification_report(ytest.values.ravel(),p1,target_names=['0','1']))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96        41
           1       0.93      0.88      0.90        16

    accuracy                           0.95        57
   macro avg       0.94      0.93      0.93        57
weighted avg       0.95      0.95      0.95        57
```

Decision Tree

In [74]:
```
model2 = DecisionTreeClassifier()
model2.fit(xtrain,ytrain)
p2 = model2.predict(xtest)
```

In [75]:
```
print(confusion_matrix(ytest,p2))
```

```
[[40  1]
 [ 2 14]]
```

In [76]:
```
print(classification_report(ytest.values.ravel(),p2,target_names=['0','1']))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96        41
           1       0.93      0.88      0.90        16

    accuracy                           0.95        57
   macro avg       0.94      0.93      0.93        57
weighted avg       0.95      0.95      0.95        57
```

Naive Bayes

In [77]:
```
model3 = GaussianNB()
model3.fit(xtrain,ytrain)
p3 = model3.predict(xtest)
```

In [78]:
```
print(confusion_matrix(ytest,p3))
```

```
[[39  2]
 [ 3 13]]
```

In [79]:
```
print(classification_report(ytest.values.ravel(),p3,target_names=['0','1']))
```

```
              precision    recall  f1-score   support

           0       0.93      0.95      0.94        41
           1       0.87      0.81      0.84        16

    accuracy                           0.91        57
   macro avg       0.90      0.88      0.89        57
weighted avg       0.91      0.91      0.91        57
```

K-Nearest Neighbour

In [80]:
```
model4 = KNeighborsClassifier()
model4.fit(xtrain,ytrain)
p4 = model4.predict(xtest)
```

In [81]:
```
print(confusion_matrix(ytest,p4))
```

```
[[38  3]
 [ 3 13]]
```

In [82]:
```
print(classification_report(ytest.values.ravel(),p4,target_names=['0','1']))
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        41
           1       0.81      0.81      0.81        16

    accuracy                           0.89        57
   macro avg       0.87      0.87      0.87        57
weighted avg       0.89      0.89      0.89        57
```

Logistic Regression

In [83]:
```
model5 = LogisticRegression()
model5.fit(xtrain,ytrain)
p5 = model5.predict(xtest)
```

In [84]:
```
print(confusion_matrix(ytest,p5))
```

```
[[40  1]
 [ 4 12]]
```

In [85]:
```
print(classification_report(ytest.values.ravel(),p5,target_names=['0','1']))
```

```
              precision    recall  f1-score   support

           0       0.91      0.98      0.94        41
           1       0.92      0.75      0.83        16

    accuracy                           0.91        57
   macro avg       0.92      0.86      0.88        57
weighted avg       0.91      0.91      0.91        57
```

K-Fold Analysis - Additional

In [129]:
```
kfold = StratifiedKFold(n_splits=4, shuffle=True, random_state=21)
```

Random Forest

In [130]:
```
acc = np.mean(cross_val_score(model1, xtrain, ytrain, scoring='accuracy', cv=kfold))
```

In [131]:
```
allen11 = pd.DataFrame({'Model':'Random Forest',
                        'Accuracy':[acc]})
```

Decision Tree

In [132]:
```
acc = np.mean(cross_val_score(model2, xtrain, ytrain, scoring='accuracy', cv=kfold))
```

In [133]:
```
allen12 = pd.DataFrame({'Model':'Decision Tree',
                        'Accuracy':[acc]})
```

Naive Bayes

In [134]:
```
acc = np.mean(cross_val_score(model3, xtrain, ytrain, scoring='accuracy', cv=kfold))
```

In [135]:
```
allen13 = pd.DataFrame({'Model':'Naive Bayes',
                        'Accuracy':[acc]})
```

K-Nearest Neighbour

In [136]:
```
acc = np.mean(cross_val_score(model4, xtrain, ytrain, scoring='accuracy', cv=kfold))
```

In [137]:
```
allen14 = pd.DataFrame({'Model':'K-Nearest Neighbour',
                        'Accuracy':[acc]})
```

Logistic Regression

In [138]:
```
acc = np.mean(cross_val_score(model5, xtrain, ytrain, scoring='accuracy', cv=kfold))
```

In [139]:
```
allen15 = pd.DataFrame({'Model':'Logistic Regression',
                        'Accuracy':[acc]})
```

Concat

In [140]:
```
al = pd.concat([allen11,allen12,allen13,allen14,allen15],axis=0).reset_index()
al = al.drop('index',axis=1)
al
```

Out[140]:

	Model	Accuracy
0	Random Forest	0.955078
1	Decision Tree	0.941406
2	Naive Bayes	0.939453
3	K-Nearest Neighbour	0.927734
4	Logistic Regression	0.939453

Inference - Additional

Random Forest Classifier is giving the best predictions for the given cancer dataset considering the high values of accuracy and F1 scores received.