ML PROJECT-1

REPORT

Name:Paladi Navya Sree

SRN:PES1UG19EC192

Section:C

Algorithm explanation:

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Accuracy:

Test=92.397

Train=94.723

CODE AND OUTPUT SCREENSHOTS:

```
[25] from google.colab import drive
     drive.mount('/content/drive')

     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[26] import warnings
     warnings.filterwarnings('ignore')
     import numpy as np
     import pandas as pd
     import math
     from sklearn.metrics import accuracy_score
     from sklearn.model_selection import train_test_split, GridSearchCV
     from sklearn.neighbors import KNeighborsClassifier
     import matplotlib.pyplot as plt
     import seaborn as sns

[27] from sklearn.datasets import load_breast_cancer

     df=pd.read_csv('/content/drive/MyDrive/data.csv')

[29] df.head()
```

[29]

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	pe
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	

5 rows × 33 columns

```
[30] diagnosis=df['diagnosis'].map(lambda row: 1 if row=='M' else 0)

[31] df.drop(['id', 'diagnosis', 'Unnamed: 32'], axis=1, inplace=True)

     df.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...

```
[33] df.isnull().sum().sum()
     0

[34] df['diagnosis']=diagnosis

     df.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	texture_worst	p
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	

5 rows × 31 columns

```
[36] df.shape
     (569, 31)

[37] train_set=df.iloc[:456, :]

[38] test_set=df.iloc[456:, :]

[39] train_data=train_set.values.tolist()

     train_data[:1]
[[17.99,
  10.38,
  122.8,
  1001.0,
  0.1184,
  0.2776,
  0.3001,
  0.1471,
  0.2419,
  0.07871,
  1.095,
  0.9053,
  8.589,
  153.4,
  0.006399,
```

✓ 0s

```
train_data[:1]
```

```
[[17.99,
  10.38,
  122.8,
  1001.0,
  0.1184,
  0.2776,
  0.3001,
  0.1471,
  0.2419,
  0.07871,
  1.095,
  0.9053,
  8.589,
  153.4,
  0.006399,
  0.04904,
  0.05373,
  0.01587,
  0.03003,
  0.006193,
  25.38,
  17.33,
  184.6,
  2019.0,
  0.1622,
  0.6656,
  0.7119,
  0.2654,
  0.4601,
  0.1189,
  1.0]]
```

```
[41] test_data=test_set.values.tolist()

[42] X_train=[row[:-1] for row in train_data]
     Y_train=[row[-1] for row in train_data]

[43] X_test=[row[:-1] for row in test_data]
     Y_test=[row[-1] for row in test_data]

[44] def euclidean_distance(train_row, test_row):
         distance=0

         for t in range(len(train_row)-1):
             distance+=(train_row[t]-test_row[t])**2

         return math.sqrt(distance)

     print(euclidean_distance([180, 80, 1], [180, 70]))
```

```
10.0
```

```
def get_distances(train_data, test_row):
    distances=[]
```

```
    distances=[]

    for train_row in train_data:
        distance=euclidean_distance(train_row, test_row)
        distances.append( (train_row, distance) )

        """
    for row in range(len(distances)):
        for i in range (len(distances)-1):
            if distances[i][-1]>distances[i+1][-1]:

                distances[i], distances[i+1]=distances[i+1], distances[i]


    print(distances)
    """
    distances.sort(key=lambda x: x[1])

    return distances
```

```
[47] print(get_distances([ [180, 80, 1], [170, 70, 1], [150, 65, 0], [160, 55, 0] ], [180, 70]))
```

```
[([180, 80, 1], 10.0), ([170, 70, 1], 10.0), ([160, 55, 0], 25.0), ([150, 65, 0], 30.4138126514911)]
```

```
def get_neighbors(train_data, test_row, num_of_neighbors):
    distances=get_distances(train_data, test_row)
    neighbors=[]
    for i in range(num_of_neighbors):
        neighbors.append(distances[i][0])

    return neighbors
```

```
[49] print(get_neighbors([ [180, 80, 1], [170, 70, 1], [150, 65, 0], [160, 55, 0] ], [180, 70], 2))
```

```
[[180, 80, 1], [170, 70, 1]]
```

```
def prediction(train_data, test_row, num_of_neighbors):
    neighbors=get_neighbors(train_data, test_row, num_of_neighbors)

    y_values=[ row[-1] for row in neighbors]

    prediction=max(y_values, key=y_values.count)

    return prediction
```

```
Y_preds=[]
for test_row in test_data:
```

```
[51] Y_preds=[]
     for test_row in test_data:
         res=prediction(train_data, test_row, 3)
         Y_preds.append(res)
```

```
[52] print(accuracy_score(Y_test, Y_preds))
```

```
0.9292035398230089
```

```
df.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	texture_worst
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67

5 rows × 31 columns

```
[54] X=df.drop('diagnosis', axis=1)
     Y=df['diagnosis']
```

```
[55] X_train, X_test, Y_train, Y_test=train_test_split(X, Y, random_state=1, test_size=0.3)
```

```
[56] knn=KNeighborsClassifier()
     knn.fit(X_train, Y_train)
     Y_pred=knn.predict(X_test)
     print(accuracy_score(Y_test, Y_pred))
```

```
0.9298245614035088
```

```
res={'neighbors':[], 'train_scores':[], 'test_scores':[]}
for each in range(1, 15):
    knn=KNeighborsClassifier(n_neighbors=each)
    knn.fit(X_train, Y_train)

    Y_train_pred=knn.predict(X_train)
    Y_test_pred=knn.predict(X_test)

    train_score=accuracy_score(Y_train, Y_train_pred)
    test_score=accuracy_score(Y_test, Y_test_pred)
```

✓ 0s completed at 23:07

```
[57] res={'neighbors':[], 'train_scores':[], 'test_scores':[]}
     for each in range(1, 15):
         knn=KNeighborsClassifier(n_neighbors=each)
         knn.fit(X_train, Y_train)

         Y_train_pred=knn.predict(X_train)
         Y_test_pred=knn.predict(X_test)

         train_score=accuracy_score(Y_train, Y_train_pred)
         test_score=accuracy_score(Y_test, Y_test_pred)

         res['neighbors'].append(each)
         res['train_scores'].append(train_score)
         res['test_scores'].append(test_score)
```

```
[58] res=pd.DataFrame(res)
```

```
[59] res
```

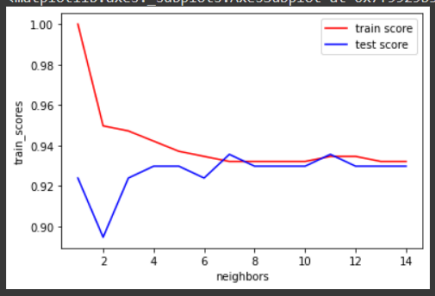
```
res
```

	neighbors	train_scores	test_scores
0	1	1.000000	0.923977
1	2	0.949749	0.894737
2	3	0.947236	0.923977
3	4	0.942211	0.929825
4	5	0.937186	0.929825
5	6	0.934673	0.923977
6	7	0.932161	0.935673
7	8	0.932161	0.929825
8	9	0.932161	0.929825
9	10	0.932161	0.929825
10	11	0.934673	0.935673
11	12	0.934673	0.929825
12	13	0.932161	0.929825
13	14	0.932161	0.929825

```
[60] sns.lineplot(res['neighbors'], res['train_scores'], label='train score', color='red')
     sns.lineplot(res['neighbors'], res['test_scores'], label='test score', color='blue')
```

```
[60] sns.lineplot(res['neighbors'], res['train_scores'], label='train score', color='red')
     sns.lineplot(res['neighbors'], res['test_scores'], label='test score', color='blue')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9929b38250>
```




```
knn_params={'n_neighbors':np.arange(1, 50)}

knn=KNeighborsClassifier()

knn_cv=GridSearchCV(knn, knn_params, cv=10)
knn_cv.fit(X_train, Y_train)
```

```
61] knn_params={'n_neighbors':np.arange(1, 50)}

    knn=KNeighborsClassifier()

    knn_cv=GridSearchCV(knn, knn_params, cv=10)
    knn_cv.fit(X_train, Y_train)

    print(knn_cv.best_params_)
```

```
{'n_neighbors': 14}
```

```
63] knn_tuned=KNeighborsClassifier(3)
    knn_tuned.fit(X_train, Y_train)
```

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

```
64] train_score=knn_tuned.score(X_train, Y_train)
    test_score=knn_tuned.score(X_test, Y_test)
```

```
print(train_score)
print(test_score)
```

```
0.9472361809045227
0.9239766081871345
```

CLASSIFICATION REPORT USING SCIKIT LEARN:

```
from sklearn.metrics import classification_report
target_names = ['Benign', 'Malignant']
print(classification_report(Y_test, Y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      Benign       0.94      0.95      0.94       108
   Malignant       0.92      0.89      0.90        63

    accuracy                           0.93       171
   macro avg       0.93      0.92      0.92       171
weighted avg       0.93      0.93      0.93       171
```

Varying parameters:

We should choose K in K - Nearest Neighbour Algorithms wisely.If we choose our K = 1 , then our algorithm behaves as over fitting and it gives a non - smooth decision surface.As K increases, our decision surface gets smoother.And, if we choose K = n, then our algorithm behaves as underfitting and it gives a smooth decision surface and everything becomes one class which is the majority class in our dataset.So, we should choose K wisely such that it should neither be overfitting nor be underfitting.