Experiment No. 6

Experiment Title: KNN

Student Name: YANA SRIVASTAVA **UID:** 20BCS2279
Branch: CSE **Section/Group:** 20BCS-WM-906/B
Semester: 5th **Date of Performance:** 28/10/22
Subject Name: Machine Learning Lab **Subject Code:** 21CSP-317

1. Aim/Overview of the practical:

Implementation of KNN (k Nearest Neighbors) Algorithm.

2. Steps of Experiment:

- Import all the required library.
- Import the dataset which you want to implement.
- Split data into x and y and perform some task.
- Split data into training set and testing set.
- Apply KNN Algorithm.

3. Source Code/Result/Output:

jupyter Untitled2 Last Checkpoint: an hour ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [14]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

In [17]:
```python
dataset = pd.read_csv(r'C:\Users\G.K.Computer Service\Downloads\Breast_cancer_data.csv')
dataset.head()
```

Out[17]:

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0

In [18]:
```python
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [19]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [22]:
```python
print(y_train)
```

```
[1 1 0 1 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1 1 1 1 0 1 1 1 1 0
 0 1 1 0 1 1 1 1 0 1 1 0 0 1 1 0 0 1 1 0 1 1 0 0 0 1 1 1 0 1 1 1 1 1 0 1 0
 1 0 1 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0 1 0 1
 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 0 1 1
 0 1 0 0 1 0 0 1 1 0 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 0 1 0 1 1 1 1 0 0 0 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0 1 1 0
 1 1 1 0 1 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 0 1 1 0 0 0 0 1 1 0 1 1
 1 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 0 1 1 1
 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 0
 0 0 1 0 1 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1
 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0
 0 1 1 1 0 1 1 0 1 1 1 1 1 0 0 0 1 1 1]
```

In [23]:
```python
print(X_test)
```

```
 [2.020e+01 2.683e+01 1.337e+02 1.234e+03 9.905e-02]
 [1.162e+01 1.818e+01 7.638e+01 4.088e+02 1.175e-01]
 [1.434e+01 1.347e+01 9.251e+01 6.412e+02 9.906e-02]
 [1.129e+01 1.304e+01 7.223e+01 3.880e+02 9.834e-02]
 [1.026e+01 1.658e+01 6.585e+01 3.208e+02 8.877e-02]
 [1.086e+01 2.148e+01 6.851e+01 3.605e+02 7.431e-02]
 [1.801e+01 2.056e+01 1.184e+02 1.007e+03 1.001e-01]
 [1.008e+01 1.511e+01 6.376e+01 3.175e+02 9.267e-02]
 [1.320e+01 1.582e+01 8.407e+01 5.373e+02 8.511e-02]
 [1.311e+01 2.254e+01 8.702e+01 5.294e+02 1.002e-01]
 [1.130e+01 1.819e+01 7.393e+01 3.894e+02 9.592e-02]
 [1.154e+01 1.072e+01 7.373e+01 4.091e+02 8.597e-02]
 [1.474e+01 2.542e+01 9.470e+01 6.686e+02 8.275e-02]
 [1.505e+01 1.907e+01 9.726e+01 7.019e+02 9.215e-02]
```

In [24]:
```
print(y_test)
```

```
[0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 1 0 1 0 1 0 1 0 1
 0 1 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0
 0 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 1
 0 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0]
```

In [25]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [26]:
```
print(X_test.dtype)
```

```
float64
```

In [27]:
```
print(X_train)
```

```
[[-0.65079907 -0.43057322 -0.68024847 -0.62698309 -0.91381897]
 [-0.82835341  0.15226547 -0.82773762 -0.75309358  0.65281216]
 [ 1.68277234  2.18977235  1.60009756  1.67383892  0.10362413]
 ...
 [-1.33114223 -0.22172269 -1.3242844  -1.05503654  0.32763504]
 [-1.25110186 -0.24600763 -1.28700242 -1.02864778 -1.94137868]
 [-0.74662205  1.14066273 -0.72203706 -0.7080938  -0.27141349]]
```

In [28]:
```
from math import sqrt
class KNN():
  def __init__(self,k):
    self.k=k
    print(self.k)
  def fit(self,X_train,y_train):
    self.x_train=X_train
    self.y_train=y_train
  def calculate_euclidean(self,sample1,sample2):
    distance=0.0
    for i in range(len(sample1)):
      distance+=(sample1[i]-sample2[i])**2 #Euclidean Distance = sqrt(sum i to N (x1_i - x2_i)^2)
    return sqrt(distance)
  def nearest_neighbors(self,test_sample):
    distances=[]#calculate distances from a test sample to every sample in a training set
    for i in range(len(self.x_train)):
      distances.append((self.y_train[i],self.calculate_euclidean(self.x_train[i],test_sample)))
    distances.sort(key=lambda x:x[1])#sort in ascending order, based on a distance value
    neighbors=[]
    for i in range(self.k): #get first k samples
      neighbors.append(distances[i][0])
    return neighbors
  def predict(self,test_set):
    predictions=[]
    for test_sample in test_set:
      neighbors=self.nearest_neighbors(test_sample)
      labels=[sample for sample in neighbors]
      prediction=max(labels,key=labels.count)
      predictions.append(prediction)
    return predictions
```

In [29]:
```
model=KNN(5)
model.fit(X_train,y_train)
```

```
5
```

In [30]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)S
classifier.fit(X_train, y_train)
```

Out[30]: `KNeighborsClassifier()`

In [31]:
```
y_pred = classifier.predict(X_test)
```

In [32]:
```
predictions=model.predict(X_test)#our model's predictions
```

In [33]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[45  8]
 [ 8 82]]
```

Out[33]: 0.8881118881118881

[[45 8]
 [8 82]]

Out[33]: 0.8881118881118881

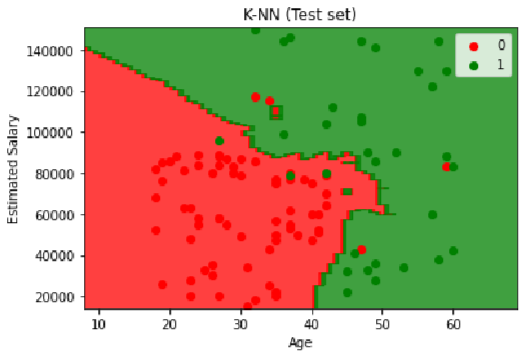
In [34]:
```
cm = confusion_matrix(y_test, predictions) #our model
print(cm)
accuracy_score(y_test, predictions)
```

```
[[45  8]
 [ 8 82]]
```

Out[34]: 0.8881118881118881

In [35]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 1),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matche
s with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA valu
e for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matche
s with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA valu
e for all points.
```



Learning outcomes (What I have learnt):

1. **Learnt to analyze the data.**
2. **Learnt to import various libraries.**
3. **Learnt to read csv files.**
4. **Learnt to train and test the data.**
5. **Learnt the concept of KNN (K – Nearest Neighbors).**

Evaluation Grid:

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Student Performance (Conduct of experiment) objectives/Outcomes.		12
2.	Viva Voce		10
3.	Submission of Work Sheet (Record)		8
	Total		30