KNN_class

March 22, 2024

[51]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

[52]:
```
data = pd.read_excel('cancer.xlsx')
data.head()
```

[52]:

	index	id	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	1	842302	17.99	10.38	122.80	1001.0	
1	2	842517	20.57	17.77	132.90	1326.0	
2	3	84300903	19.69	21.25	130.00	1203.0	
3	4	84348301	11.42	20.38	77.58	386.1	
4	5	84358402	20.29	14.34	135.10	1297.0	

	smoothness_mean	compactness_mean	concavity_mean	concave_points_mean	\
0	0.11840	0.27760	0.3001	0.14710	
1	0.08474	0.07864	0.0869	0.07017	
2	0.10960	0.15990	0.1974	0.12790	
3	0.14250	0.28390	0.2414	0.10520	
4	0.10030	0.13280	0.1980	0.10430	

	…	smoothness_worst	compactness_worst	concavity_worst	\
0	…	0.1622	0.6656	0.7119	
1	…	0.1238	0.1866	0.2416	
2	…	0.1444	0.4245	0.4504	
3	…	0.2098	0.8663	0.6869	
4	…	0.1374	0.2050	0.4000	

	concave points_worst	symmetry_worst	fractal_dimension_worst	N Stage	\
0	0.2654	0.4601	0.11890	N1	
1	0.1860	0.2750	0.08902	N2	
2	0.2430	0.3613	0.08758	N3	
3	0.2575	0.6638	0.17300	N1	
4	0.1625	0.2364	0.07678	N1	

	6th Stage	differentiate	diagnosis

```
0        IIA      Poorly differentiated          M
1       IIIA  Moderately differentiated          M
2       IIIC  Moderately differentiated          M
3        IIA      Poorly differentiated          M
4        IIB      Poorly differentiated          M

[5 rows x 36 columns]
```

[53]:
```
# Prepare the model
y = data["diagnosis"] # our target variable
X = data.drop(["diagnosis","index","id"], axis=1) # our predictors
X.shape
```

[53]:
```
(569, 33)
```

[54]:
```
# Taking care missing data
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X.iloc[:, 0:29])
X.iloc[:, 0:29] = imputer.transform(X.iloc[:, 0:29])
```

[55]:
```
# One hot encoding
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [30,31,32])],␣
 ↪remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

[56]:
```
# Spliting the dataset
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)
```

[57]:
```
#Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train
```

[57]:
```
array([[-1.49240501,  2.21735578, -0.40488817, …, -0.38664354,
         0.32349851, -0.7578486 ],
       [ 0.67005939, -0.45098762, -0.40488817, …, -1.48895322,
         0.62563098, -1.03071387],
       [-1.49240501,  2.21735578, -0.40488817, …,  0.71907312,
        -0.51329768, -0.96601386],
       …,
```

```
        [ 0.67005939, -0.45098762, -0.40488817, …, -1.01972052,
         -0.69995543, -0.12266325],
        [ 0.67005939, -0.45098762, -0.40488817, …, -1.80208475,
         -1.56206114, -1.00989735],
        [-1.49240501,  2.21735578, -0.40488817, …, -0.30719919,
         -1.24094654,  0.2126516 ]])
```

[58]:
```python
#Fit the values
import numpy as np

class KNNClassifier:
    def __init__(self, k):
        self.k = k

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        y_pred = []
        for sample in X_test:
            distances = []
            for i, train_sample in enumerate(self.X_train):
                distance = np.sqrt(np.sum((sample - train_sample) ** 2))
                distances.append((distance, self.y_train[i]))
            distances.sort()
            k_nearest = distances[:self.k]
            counts = {}
            for _, label in k_nearest:
                counts[label] = counts.get(label, 0) + 1
            most_common = max(counts, key=counts.get)
            y_pred.append(most_common)
        return y_pred

k = 3

knn = KNNClassifier(k)
X_train=np.array(X_train)
y_train=np.array(y_train)
knn.fit(X_train, y_train)
```

[59]:
```python
#predictions
X_test=np.array(X_test)
y_test=np.array(y_test)
y_pred = knn.predict(X_test)
y_pred_array = np.array(y_pred)
y_test_array = np.array(y_test)
```

```
# Reshape arrays
y_pred_reshaped = y_pred_array.reshape(len(y_pred_array), 1)
y_test_reshaped = y_test_array.reshape(len(y_test_array), 1)

# Concatenate arrays along the second axis
concatenated_array = np.concatenate((y_pred_reshaped, y_test_reshaped), axis=1)

print(concatenated_array)
```

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

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

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

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

[61]:
```
# Confusion metrics
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)*100
```

```
[[89  1]
 [ 9 44]]
```

[61]: 93.00699300699301