Proj6_resolvido

March 19, 2025

Prever se uma célula é Benigna ou Maligna usando KNN Observe as ações/instruções abaixo. Justifique cada passo e apresente a conclusão final da análise.

Bibliotecas

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

[2]:
```
# df = pd.read_csv('/kaggle/input/breast-cancer-wisconsin-data/data.csv')
df = pd.read_csv('data.csv')
df.head()
```

[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_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	

	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	

	…	texture_worst	perimeter_worst	area_worst	smoothness_worst	\
0	…	17.33	184.60	2019.0	0.1622	
1	…	23.41	158.80	1956.0	0.1238	
2	…	25.53	152.50	1709.0	0.1444	
3	…	26.50	98.87	567.7	0.2098	
4	…	16.67	152.20	1575.0	0.1374	

	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	\
0	0.6656	0.7119	0.2654	0.4601	
1	0.1866	0.2416	0.1860	0.2750	

```
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   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

[5 rows x 33 columns]
```

[3]:
```
df.drop(['id', 'Unnamed: 32'], axis=1, inplace=True)
```

[4]:
```
df.shape
```

[4]:
```
(569, 31)
```

[5]:
```
df.isna().sum()
```

[5]:
```
diagnosis                0
radius_mean              0
texture_mean             0
perimeter_mean           0
area_mean                0
smoothness_mean          0
compactness_mean         0
concavity_mean           0
concave points_mean      0
symmetry_mean            0
fractal_dimension_mean   0
radius_se                0
texture_se               0
perimeter_se             0
area_se                  0
smoothness_se            0
compactness_se           0
concavity_se             0
concave points_se        0
symmetry_se              0
fractal_dimension_se     0
radius_worst             0
texture_worst            0
perimeter_worst          0
area_worst               0
smoothness_worst         0
```

```
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

[6]:
```
df.dtypes
```

[6]:
```
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
dtype: object
```

[7]:
```
df['diagnosis'] = df['diagnosis'].astype('category')
```

[8]:
```
df.dtypes
```

[8]:
```
diagnosis                 category
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object
```

Quanto regular é a distribuição da categoria

[9]:
```
df['diagnosis'].value_counts()
```

[9]:
```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

É uma boa distribuição

Divisão dos dados

[10]:
```
y = df['diagnosis']
x = df.drop('diagnosis', axis=1)
```

Normalização dos dados Opcional...

[11]:
```
from scipy.stats import zscore
xscaled = x.apply(zscore)
```

[12]:
```
xscaled.describe().T
# All will have the same std
```

[12]:

	count	mean	std	min	25%	\
radius_mean	569.0	-1.373633e-16	1.00088	-2.029648	-0.689385	
texture_mean	569.0	6.868164e-17	1.00088	-2.229249	-0.725963	
perimeter_mean	569.0	-1.248757e-16	1.00088	-1.984504	-0.691956	
area_mean	569.0	-2.185325e-16	1.00088	-1.454443	-0.667195	
smoothness_mean	569.0	-8.366672e-16	1.00088	-3.112085	-0.710963	
compactness_mean	569.0	1.873136e-16	1.00088	-1.610136	-0.747086	
concavity_mean	569.0	4.995028e-17	1.00088	-1.114873	-0.743748	
concave points_mean	569.0	-4.995028e-17	1.00088	-1.261820	-0.737944	
symmetry_mean	569.0	1.748260e-16	1.00088	-2.744117	-0.703240	
fractal_dimension_mean	569.0	4.745277e-16	1.00088	-1.819865	-0.722639	
radius_se	569.0	2.372638e-16	1.00088	-1.059924	-0.623571	
texture_se	569.0	-1.123881e-16	1.00088	-1.554264	-0.694809	
perimeter_se	569.0	-1.123881e-16	1.00088	-1.044049	-0.623768	
area_se	569.0	-1.311195e-16	1.00088	-0.737829	-0.494754	
smoothness_se	569.0	-1.529727e-16	1.00088	-1.776065	-0.624018	
compactness_se	569.0	1.748260e-16	1.00088	-1.298098	-0.692926	
concavity_se	569.0	1.623384e-16	1.00088	-1.057501	-0.557161	
concave points_se	569.0	0.000000e+00	1.00088	-1.913447	-0.674490	
symmetry_se	569.0	8.741299e-17	1.00088	-1.532890	-0.651681	
fractal_dimension_se	569.0	-6.243785e-18	1.00088	-1.096968	-0.585118	
radius_worst	569.0	-8.241796e-16	1.00088	-1.726901	-0.674921	
texture_worst	569.0	1.248757e-17	1.00088	-2.223994	-0.748629	
perimeter_worst	569.0	-3.746271e-16	1.00088	-1.693361	-0.689578	
area_worst	569.0	0.000000e+00	1.00088	-1.222423	-0.642136	
smoothness_worst	569.0	-2.372638e-16	1.00088	-2.682695	-0.691230	
compactness_worst	569.0	-3.371644e-16	1.00088	-1.443878	-0.681083	
concavity_worst	569.0	7.492542e-17	1.00088	-1.305831	-0.756514	
concave points_worst	569.0	2.247763e-16	1.00088	-1.745063	-0.756400	
symmetry_worst	569.0	2.622390e-16	1.00088	-2.160960	-0.641864	
fractal_dimension_worst	569.0	-5.744282e-16	1.00088	-1.601839	-0.691912	

	50%	75%	max
radius_mean	-0.215082	0.469393	3.971288
texture_mean	-0.104636	0.584176	4.651889
perimeter_mean	-0.235980	0.499677	3.976130
area_mean	-0.295187	0.363507	5.250529
smoothness_mean	-0.034891	0.636199	4.770911
compactness_mean	-0.221940	0.493857	4.568425

```
concavity_mean           -0.342240  0.526062   4.243589
concave points_mean      -0.397721  0.646935   3.927930
symmetry_mean            -0.071627  0.530779   4.484751
fractal_dimension_mean   -0.178279  0.470983   4.910919
radius_se                -0.292245  0.266100   8.906909
texture_se               -0.197498  0.466552   6.655279
perimeter_se             -0.286652  0.243031   9.461986
area_se                  -0.347783  0.106773  11.041842
smoothness_se            -0.220335  0.368355   8.029999
compactness_se           -0.281020  0.389654   6.143482
concavity_se             -0.199065  0.336752  12.072680
concave points_se        -0.140496  0.472657   6.649601
symmetry_se              -0.219430  0.355692   7.071917
fractal_dimension_se     -0.229940  0.288642   9.851593
radius_worst             -0.269040  0.522016   4.094189
texture_worst            -0.043516  0.658341   3.885905
perimeter_worst          -0.285980  0.540279   4.287337
area_worst               -0.341181  0.357589   5.930172
smoothness_worst         -0.046843  0.597545   3.955374
compactness_worst        -0.269501  0.539669   5.112877
concavity_worst          -0.218232  0.531141   4.700669
concave points_worst     -0.223469  0.712510   2.685877
symmetry_worst           -0.127409  0.450138   6.046041
fractal_dimension_worst -0.216444  0.450762   6.846856
```

Modelo KNN

A divisão dos dados

[13]:
```
from sklearn.model_selection import train_test_split
```

[14]:
```
x_train, x_test, y_train, y_test = train_test_split(xscaled,y, test_size=0.3,␣
↪random_state=1)
```

[15]:
```
from sklearn.neighbors import  KNeighborsClassifier
KNN = KNeighborsClassifier(n_neighbors=5, weights='distance')
# Utilizamos weights='distance' para dar prioridade ao vizinho mais próximo
# também se pode usar pesos='uniforme', trataria todos os vizinhos de forma␣
↪igual sem considerar a distância
```

[16]:
```
KNN.fit(x_train, y_train)
```

[16]:
```
KNeighborsClassifier(weights='distance')
```

[17]:
```
KNN_predict = KNN.predict(x_test)
KNN_predict
```

[17]:
```
array(['B', 'M', 'B', 'M', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'M',
       'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'B',
       'B', 'M', 'M', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'M', 'M',
       'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B',
       'B', 'M', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'B', 'B',
       'M', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'B',
       'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'M', 'M', 'M', 'M', 'M',
       'B', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B',
       'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B',
       'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M', 'M',
       'B', 'B'], dtype=object)
```

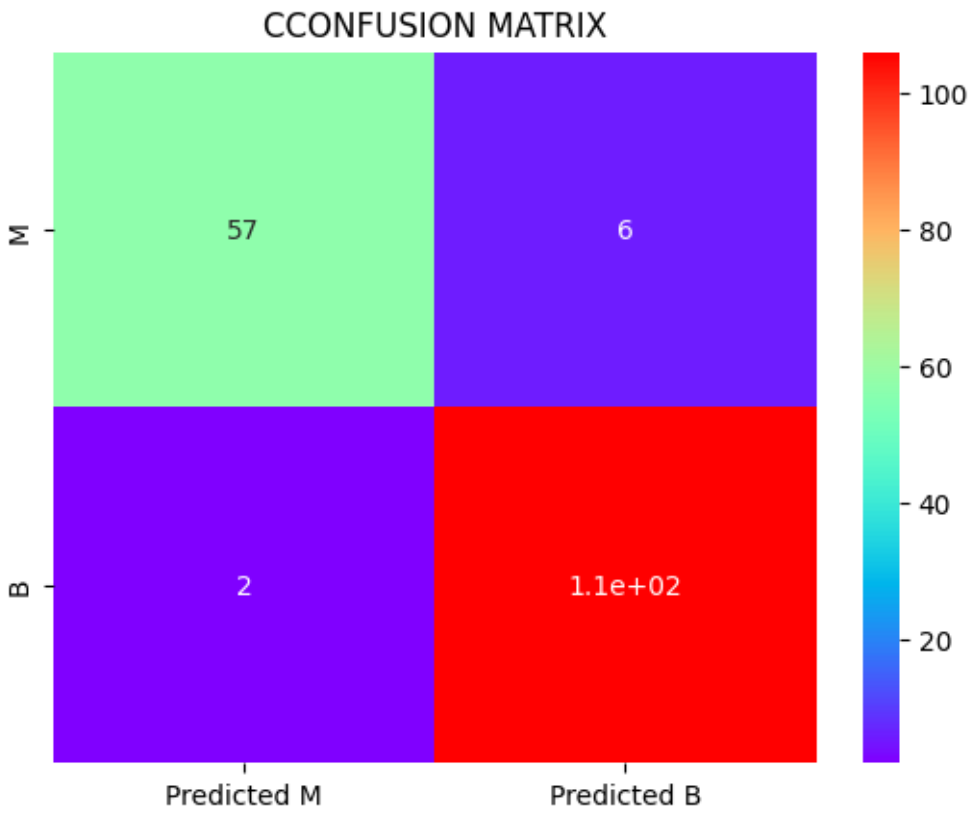
[18]:
```
KNN.score(x_test, y_test)
```

[18]:
```
0.9532163742690059
```

[19]:
```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, KNN_predict, labels=['M', 'B'])

df_cm = pd.DataFrame(cm, index =[i for i in ['M', 'B']],
                     columns = [i for i in ['Predicted M', 'Predicted B']])

sns.heatmap(df_cm, annot=True, cmap='rainbow')
plt.title('CCONFUSION MATRIX')
plt.show()
```



Como escolher os K(n_neighbors)

[20]:
```
from sklearn.model_selection import cross_val_score

score_1 = []

for i in range(1,50):
    KNN_2 = KNeighborsClassifier(n_neighbors=i)
    score_2 = cross_val_score(KNN_2,xscaled, y, cv=10) # cv=10 significa que␣
 ↪para cada pontuação, esta será feita 10 vezes
    score_1.append(score_2.mean()) # a média de cada pontuação (recorde-se que␣
 ↪cada pontuação foi calculada 10 vezes)
```

[21]:
```
score_1
```

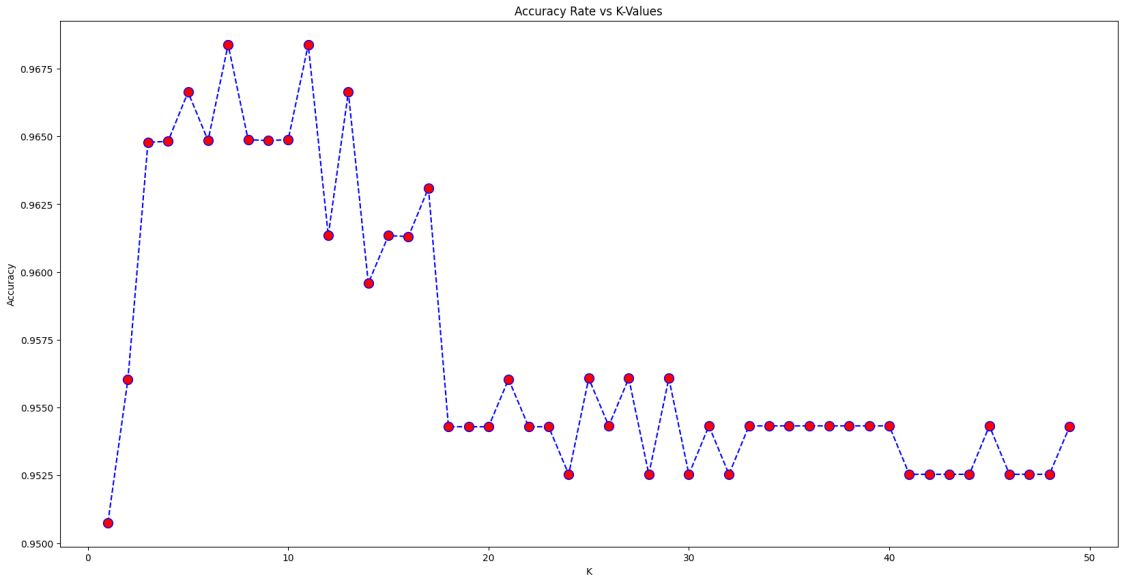
[21]:
```
[np.float64(0.9507518796992482),
 np.float64(0.9560463659147869),
 np.float64(0.9647869674185465),
 np.float64(0.9648182957393484),
 np.float64(0.9666353383458647),
 np.float64(0.9648496240601503),
```

```
np.float64(0.9683897243107771),
np.float64(0.9648809523809524),
np.float64(0.9648496240601503),
np.float64(0.9648809523809524),
np.float64(0.9683897243107769),
np.float64(0.9613408521303258),
np.float64(0.9666353383458647),
np.float64(0.9595864661654134),
np.float64(0.9613408521303256),
np.float64(0.9613095238095237),
np.float64(0.963095238095238),
np.float64(0.9542919799498746),
np.float64(0.9542919799498746),
np.float64(0.9542919799498746),
np.float64(0.9560463659147869),
np.float64(0.9542919799498746),
np.float64(0.9542919799498746),
np.float64(0.9525375939849624),
np.float64(0.9560776942355889),
np.float64(0.9543233082706767),
np.float64(0.9560776942355889),
np.float64(0.9525375939849623),
np.float64(0.9560776942355889),
np.float64(0.9525375939849623),
np.float64(0.9543233082706765),
np.float64(0.9525375939849623),
np.float64(0.9543233082706765),
np.float64(0.9543233082706765),
np.float64(0.9543233082706765),
np.float64(0.9543233082706765),
np.float64(0.9543233082706765),
np.float64(0.9543233082706765),
np.float64(0.9543233082706765),
np.float64(0.9525375939849623),
np.float64(0.9525375939849623),
np.float64(0.9525375939849623),
np.float64(0.9525375939849623),
np.float64(0.9543233082706765),
np.float64(0.9525375939849623),
np.float64(0.9525375939849623),
np.float64(0.9525375939849623),
np.float64(0.9542919799498746)]
```

[22]:
```
plt.figure(figsize=(20,10))
```

```
plt.plot(range(1,50), score_1, color='blue', linestyle='dashed', marker='o',␣
 ↪markerfacecolor='red', markersize=10)

plt.title('Accuracy Rate vs K-Values')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.show()
```



Usar o melhor k

[23]:
```
model = KNeighborsClassifier(n_neighbors=9, weights='distance')
# I used weights='distance' so that it will give priority to the nearest␣
 ↪neighbor
# you can also use weights='uniform', it owuld treated all neighbors equally␣
 ↪without considering the distance
```

[24]:
```
model.fit(x_train,y_train)
```

[24]:
```
KNeighborsClassifier(n_neighbors=9, weights='distance')
```

[25]:
```
model.score(x_test, y_test)
```

[25]:
```
0.9649122807017544
```

0.0.1 *Conclusão*

Um bom nível de previsão para 9 vizinhos nos agrupamentos.