KNN-Project

May 20, 2023

1 KNN Data Science Project

1.1 The Sonar Data

1.1.1 Detecting a Rock or a Mine

Sonar (sound navigation ranging) is a technique that uses sound propagation (usually underwater, as in submarine navigation) to navigate, communicate with or detect objects on or under the surface of the water, such as other vessels.

The data set contains the response metrics for 60 separate sonar frequencies sent out against a known mine field (and known rocks). These frequencies are then labeled with the known object they were beaming the sound at (either a rock or a mine).

Our main goal is to create a machine learning model capable of detecting the difference between a rock or a mine based on the response of the 60 separate sonar frequencies.

Data Source: https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks)

1.1.2 Complete the Tasks in bold

TASK: Run the cells below to load the data.

1.2 Dados Sonar

1.2.1 Detectando uma Rocha ou uma Mina

Sonar (variação de navegação por som) é uma técnica que usa a propagação do som (geralmente debaixo d'água, como na navegação submarina) para navegar, comunicar-se ou detectar objetos na superfície da água ou sob ela, como outras embarcações.

O conjunto de dados contém as métricas de resposta para 60 frequências de sonar separadas enviadas contra um campo minado conhecido (e rochas conhecidas). Essas frequências são então rotuladas com o objeto conhecido para o qual estavam transmitindo o som (uma rocha ou uma mina).

Nosso principal objetivo é criar um modelo de aprendizado de máquina capaz de detectar a diferença entre uma rocha ou uma mina com base na resposta das 60 frequências de sonar separadas.

Fonte de dados: https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks)

Importing Libraries

Importando as Bibliotecas

[5]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[6]:
```
df = pd.read_csv('../DATA/sonar.all-data.csv')
```

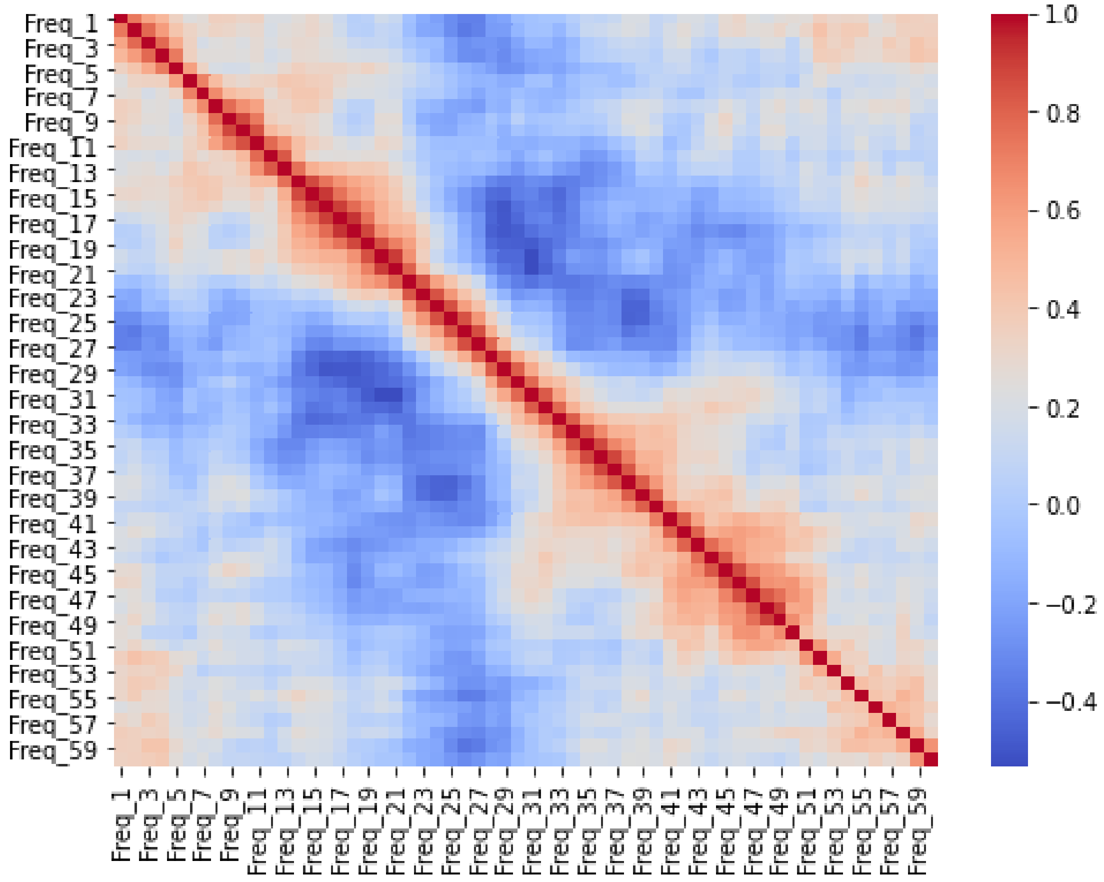
[7]:
```
df.head()
```

[7]:
```
   Freq_1  Freq_2  Freq_3  Freq_4  Freq_5  Freq_6  Freq_7  Freq_8  Freq_9  \
0  0.0200  0.0371  0.0428  0.0207  0.0954  0.0986  0.1539  0.1601  0.3109
1  0.0453  0.0523  0.0843  0.0689  0.1183  0.2583  0.2156  0.3481  0.3337
2  0.0262  0.0582  0.1099  0.1083  0.0974  0.2280  0.2431  0.3771  0.5598
3  0.0100  0.0171  0.0623  0.0205  0.0205  0.0368  0.1098  0.1276  0.0598
4  0.0762  0.0666  0.0481  0.0394  0.0590  0.0649  0.1209  0.2467  0.3564

   Freq_10  …  Freq_52  Freq_53  Freq_54  Freq_55  Freq_56  Freq_57  \
0   0.2111  …   0.0027   0.0065   0.0159   0.0072   0.0167   0.0180
1   0.2872  …   0.0084   0.0089   0.0048   0.0094   0.0191   0.0140
2   0.6194  …   0.0232   0.0166   0.0095   0.0180   0.0244   0.0316
3   0.1264  …   0.0121   0.0036   0.0150   0.0085   0.0073   0.0050
4   0.4459  …   0.0031   0.0054   0.0105   0.0110   0.0015   0.0072

   Freq_58  Freq_59  Freq_60  Label
0   0.0084   0.0090   0.0032      R
1   0.0049   0.0052   0.0044      R
2   0.0164   0.0095   0.0078      R
3   0.0044   0.0040   0.0117      R
4   0.0048   0.0107   0.0094      R

[5 rows x 61 columns]
```

1.3 Data Exploration

[98]:
```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(),cmap='coolwarm')
```

[98]:
```
<AxesSubplot:>
```



```
[3]: **Changing Target Variable**
```

```
  Cell In[3], line 1
    **Changing Target Variable**
    ^
SyntaxError: invalid syntax
```

```
[8]: df['Target'] = df['Label'].map({'R':0,'M':1})
```

```
[101]: np.abs(df.corr()['Target']).sort_values().tail(6)
```

```
[101]: Freq_45    0.339406
       Freq_10    0.341142
       Freq_49    0.351312
       Freq_12    0.392245
       Freq_11    0.432855
       Target     1.000000
```

```
Name: Target, dtype: float64
```

1.4 Train | Test Split

Our approach here will be one of using Cross Validation on 90% of the dataset, and then judging our results on a final test set of 10% to evaluate our model.

Nossa abordagem aqui será usar validação cruzada em 90% do conjunto de dados e, em seguida, julgar nossos resultados em um conjunto de teste final de 10% para avaliar nosso modelo.

[103]:
```
from sklearn.model_selection import train_test_split
```

[104]:
```
X = df.drop(['Target','Label'],axis=1)
y = df['Label']
```

[105]:
```
X_cv, X_test, y_cv, y_test = train_test_split(X, y, test_size=0.1,␣
↪random_state=42)
```

TASK: Create a PipeLine that contains both a StandardScaler and a KNN model

TAREFA: Crie um PipeLine que contenha um StandardScaler e um modelo KNN

[106]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
```

[107]:
```
scaler = StandardScaler()
```

[108]:
```
knn = KNeighborsClassifier()
```

[109]:
```
operations = [('scaler',scaler),('knn',knn)]
```

[110]:
```
from sklearn.pipeline import Pipeline
```

[111]:
```
pipe = Pipeline(operations)
```

TASK: Perform a grid-search with the pipeline to test various values of k and report back the best performing parameters.

TAREFA: execute uma pesquisa de grade com o pipeline para testar vários valores de k e relatar os parâmetros de melhor desempenho.

[112]:
```
from sklearn.model_selection import GridSearchCV
```

[79]:
```
k_values = list(range(1,30))
```

[80]:
```
param_grid = {'knn__n_neighbors': k_values}
```

[81]:
```
full_cv_classifier = GridSearchCV(pipe,param_grid,cv=5,scoring='accuracy')
```

[82]:
```
full_cv_classifier.fit(X_cv,y_cv)
```

```
[82]: GridSearchCV(cv=5,
                   estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                             ('knn', KNeighborsClassifier())]),
                   param_grid={'knn__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
                                                    12, 13, 14, 15, 16, 17, 18, 19,
                                                    20, 21, 22, 23, 24, 25, 26, 27,
                                                    28, 29]},
                   scoring='accuracy')
```

```
[83]: full_cv_classifier.best_estimator_.get_params()
```

```
[83]: {'memory': None,
       'steps': [('scaler', StandardScaler()),
        ('knn', KNeighborsClassifier(n_neighbors=1))],
       'verbose': False,
       'scaler': StandardScaler(),
       'knn': KNeighborsClassifier(n_neighbors=1),
       'scaler__copy': True,
       'scaler__with_mean': True,
       'scaler__with_std': True,
       'knn__algorithm': 'auto',
       'knn__leaf_size': 30,
       'knn__metric': 'minkowski',
       'knn__metric_params': None,
       'knn__n_jobs': None,
       'knn__n_neighbors': 1,
       'knn__p': 2,
       'knn__weights': 'uniform'}
```

** TASK: Using the .cv_results_ dictionary, see if you can create a plot of the mean test scores per K value.**

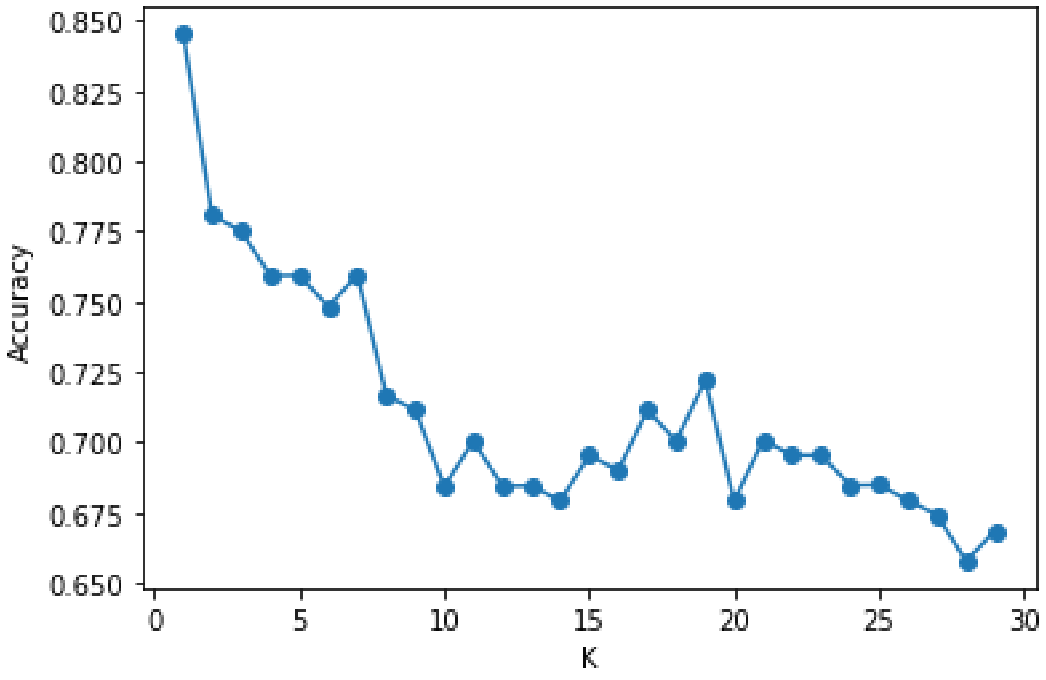
** TAREFA: Usando o dicionário .cv_results_, veja se você pode criar um gráfico das pontuações médias do teste por valor K.**

```
[114]: full_cv_classifier.cv_results_['mean_test_score']
```

```
[114]: array([0.84537696, 0.78065434, 0.77524893, 0.75917496, 0.75931721,
              0.74822191, 0.75945946, 0.71664296, 0.7113798 , 0.68421053,
              0.70042674, 0.68435277, 0.68449502, 0.67908962, 0.69530583,
              0.68990043, 0.7113798 , 0.70042674, 0.72204836, 0.67908962,
              0.70071124, 0.69530583, 0.69530583, 0.68463727, 0.68477952,
              0.67923186, 0.67411095, 0.65775249, 0.6685633 ])
```

```
[115]: scores = full_cv_classifier.cv_results_['mean_test_score']
       plt.plot(k_values,scores,'o-')
       plt.xlabel("K")
       plt.ylabel("Accuracy")
```

[115]: Text(0, 0.5, 'Accuracy')



1.4.1 Final Model Evaluation

TASK: Using the grid classifier object from the previous step, get a final performance classification report and confusion matrix.

TAREFA: Usando o objeto classificador de grade da etapa anterior, obtenha um relatório final de classificação de desempenho e uma matriz de confusão.

[119]:
```
pred = full_cv_classifier.predict(X_test)
```

[120]:
```
from sklearn.metrics import␣
↪classification_report,confusion_matrix,accuracy_score
```

[121]:
```
confusion_matrix(y_test,pred)
```

[121]:
```
array([[12,  1],
       [ 1,  7]], dtype=int64)
```

[122]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           M       0.92      0.92      0.92        13
           R       0.88      0.88      0.88         8
```

```
    accuracy                           0.90        21
   macro avg       0.90      0.90      0.90        21
weighted avg       0.90      0.90      0.90        21
```

1.4.2 End!

1.4.3 Fim!