prac5

October 18, 2024

Aim: Implementing KNN for regression

Theory: ### Implementing K-Nearest Neighbors (KNN) for Regression: Theory

K-Nearest Neighbors (KNN) is a versatile algorithm commonly used for both classification and regression tasks. In the context of regression, KNN is employed to predict continuous outcomes by using the average (or weighted average) of the values of the nearest neighbors. Unlike KNN for classification, which assigns the majority class label to a new data point, KNN regression predicts a numerical value based on the surrounding data points.

0.0.1 Key Concepts in KNN Regression

1. **Basic Idea of KNN Regression**
   - **Nearest Neighbors**: The prediction for a new data point is made by finding the **k nearest neighbors** in the training dataset.
   - **Averaging**: Once the nearest neighbors are identified, the prediction is typically made by taking the **average** of the output values (dependent variables) of those neighbors.
2. **How KNN Regression Works**

   The steps for KNN regression are similar to KNN classification, but instead of class labels, the algorithm deals with continuous values:

   - **Step 1**: Choose the value of **k**, the number of nearest neighbors to consider.
   - **Step 2**: For each test data point, calculate the distance between the test point and every point in the training dataset using a distance metric such as **Euclidean distance**, **Manhattan distance**, etc.
   - **Step 3**: Identify the **k nearest neighbors** based on the calculated distances.
   - **Step 4**: Take the **average** (or weighted average) of the target variable values (continuous values) of these k neighbors. This average becomes the predicted value for the test data point.
   - **Step 5**: Assign the predicted value to the test data point.
3. **Choosing the Value of K**
   - **Low K (e.g., k=1)**: If (k = 1), the algorithm will simply predict the value of the nearest neighbor. This can lead to high variance and overfitting, especially in the presence of noise.
   - **High K**: A larger value of (k) smooths the predictions by averaging the values of many neighbors, which may help reduce variance but can lead to underfitting if (k) is too large.

- **Cross-Validation**: The optimal value of (k) is often chosen using cross-validation, balancing between bias (underfitting) and variance (overfitting).

4. **Distance Metrics**

   As with classification, the performance of KNN regression is influenced by the choice of distance metric:

   - **Euclidean Distance**: This is the most common distance metric used for continuous variables.
   - **Manhattan Distance**: Works well in some cases where the data is structured more like a grid.

5. **Weighted KNN**

   - In some cases, instead of treating all neighbors equally, KNN regression can assign **weights** to neighbors based on their distance from the query point. Closer neighbors may be given higher weight than those farther away.
   - Common weighting schemes include inversely weighting the neighbors by distance (closer neighbors have higher weight) or using other kernel-based methods.

6. **Advantages of KNN Regression**

   - **Simplicity**: KNN is easy to understand and implement since it does not involve any training phase or model parameter estimation.
   - **No Assumptions**: Unlike linear regression or other parametric models, KNN makes no assumptions about the underlying relationship between input and output variables.
   - **Flexibility**: KNN can capture complex non-linear relationships between variables, making it suitable for a wide variety of tasks.

7. **Disadvantages of KNN Regression**

   - **Computational Complexity**: KNN can be computationally expensive when predicting new data points because the algorithm must compute the distance between the test point and all training points.
   - **Sensitive to Outliers**: KNN is prone to being affected by outliers, as these can skew the prediction by pulling the average in an undesired direction.
   - **Curse of Dimensionality**: As the number of features (dimensions) increases, the distance between points becomes less meaningful. This is referred to as the "curse of dimensionality," and KNN may perform poorly in high-dimensional spaces without proper feature selection or dimensionality reduction techniques like PCA.

8. **Preprocessing and Scaling**

   - **Normalization/Standardization**: Since KNN is distance-based, it is sensitive to the scale of features. Features with larger ranges can dominate the distance calculations, so it is important to **normalize** or **standardize** the input data to ensure that each feature contributes equally.
   - **Handling Missing Data**: KNN does not inherently handle missing data well. Missing values in the training data should be imputed or removed prior to using the KNN algorithm.

0.0.2 Use Cases of KNN Regression

- **Predicting House Prices**: KNN regression can be used to predict the price of a house based on the prices of similar houses in the neighborhood.
- **Stock Market Forecasting**: KNN can predict stock prices or trends by finding patterns in past data points that are similar to current conditions.
- **Environmental Modeling**: KNN can model phenomena such as temperature or pollution levels based on historical data from nearby locations.

[ ]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
↪outside of the current session
```

```
/kaggle/input/wine-quality-dataset/WineQT.csv
```

[ ]:
```
df = pd.read_csv("/kaggle/input/wine-quality-dataset/WineQT.csv")
```

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

[ ]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
```

```
2                 15.0                    54.0    0.9970  3.26       0.65
3                 17.0                    60.0    0.9980  3.16       0.58
4                 11.0                    34.0    0.9978  3.51       0.56

   alcohol  quality  Id
0      9.4        5   0
1      9.8        5   1
2      9.8        5   2
3      9.8        6   3
4      9.4        5   4
```

[ ]:
```
df.describe()
```

[ ]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1143.000000       1143.000000  1143.000000     1143.000000
mean        8.311111          0.531339     0.268364        2.532152
std         1.747595          0.179633     0.196686        1.355917
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.392500     0.090000        1.900000
50%         7.900000          0.520000     0.250000        2.200000
75%         9.100000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1143.000000          1143.000000           1143.000000  1143.000000
mean      0.086933            15.615486             45.914698     0.996730
std       0.047267            10.250486             32.782130     0.001925
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             21.000000     0.995570
50%       0.079000            13.000000             37.000000     0.996680
75%       0.090000            21.000000             61.000000     0.997845
max       0.611000            68.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality           Id
count  1143.000000  1143.000000  1143.000000  1143.000000  1143.000000
mean      3.311015     0.657708    10.442111     5.657043   804.969379
std       0.156664     0.170399     1.082196     0.805824   463.997116
min       2.740000     0.330000     8.400000     3.000000     0.000000
25%       3.205000     0.550000     9.500000     5.000000   411.000000
50%       3.310000     0.620000    10.200000     6.000000   794.000000
75%       3.400000     0.730000    11.100000     6.000000  1209.500000
max       4.010000     2.000000    14.900000     8.000000  1597.000000
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
```

```
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64
 12  Id                    1143 non-null   int64
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

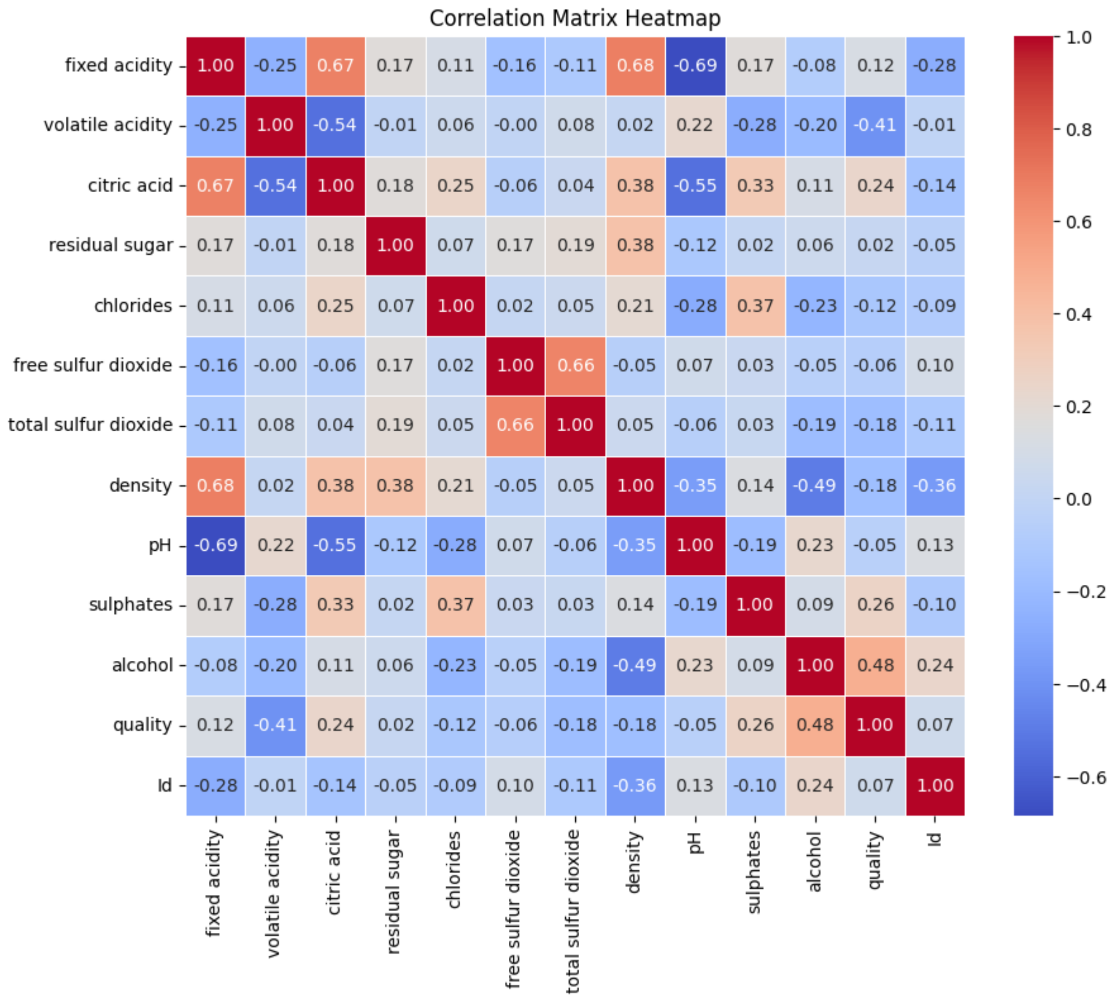
[ ]:
```
df.isnull().sum()
```

[ ]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Id                      0
dtype: int64
```

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

[ ]:
```
correlation = df.corr()
```

[ ]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm',␣
 ↪square=True, linewidths=.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



[ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

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

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

[ ]: (1143, 13)

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)
```

[ ]:
```
X_train.shape, X_test.shape
```

[ ]: ((914, 12), (229, 12))

[ ]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[ ]:
```
X_train = pd.DataFrame(X_train, columns=X.columns)
X_test = pd.DataFrame(X_test, columns=X.columns)
```

[ ]:
```
knn = KNeighborsRegressor()

param_grid = {
    'n_neighbors': range(1, 21),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'chebyshev']
}

grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,␣
 ↪n_jobs=-1, scoring='neg_mean_squared_error')

grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation MSE:", -grid_search.best_score_)

best_knn = grid_search.best_estimator_
y_pred = best_knn.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred)
print("Test Set MSE:", test_mse)
r2 = r2_score(y_test, y_pred)
print("R-squared:", r2)
```

```
Best Parameters: {'metric': 'manhattan', 'n_neighbors': 20, 'weights':
'distance'}
Best Cross-Validation MSE: 0.39812943327185785
Test Set MSE: 0.2566969620590992
R-squared: 0.5387072377718726
```

0.0.3 Conclusion

KNN for regression is a simple and flexible algorithm that can model complex relationships in data. It predicts continuous values by averaging the outputs of the nearest neighbors and is particularly useful when the relationship between the variables is non-linear. However, it requires careful tuning of the hyperparameter (k) and proper feature scaling to perform effectively. Despite its simplicity, KNN remains a valuable tool in the machine learning toolbox, particularly for small to medium-sized datasets.