KNN regressor

2024-12-05

Contents

In this report, I perform K Nearest Neighbour (KNN) model to predict saturation vapour pressure. KNN predicts the value of the target variable based on the average of the values of its k-nearest neighbors in the training set. Since KNN does not accept missing values in the dataset, I run the model on complete-case train data.

Initial KKN model with varying number of k

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score, KFold


# read complete-case train data
train = pd.read_csv('../data/train_complete_cases.csv')

X_train = train.iloc[:, 2:]
y_train = train['log_pSat_Pa']

mses = []
ks = range(1, X_train.shape[1] + 1)
for k in ks :
  model = KNeighborsRegressor(n_neighbors=k)
  kf = KFold(n_splits=10, shuffle=True, random_state=100)
  fold_scores = -cross_val_score(model, X_train, y_train, cv=kf, scoring='neg_mean_squared_error')
  mse = round(fold_scores.mean(), 3)
  mses.append(mse)

result = pd.DataFrame({'k': ks, 'MSE': mses})
print (result)

# find the lowest MSE and MSE with 1% margin of error
```

```
##       k    MSE
## 0     1  8.375
## 1     2  6.499
## 2     3  5.856
## 3     4  5.561
## 4     5  5.419
## 5     6  5.319
## 6     7  5.270
## 7     8  5.231
## 8     9  5.210
## 9    10  5.194
## 10   11  5.185
## 11   12  5.176
## 12   13  5.166
## 13   14  5.155
## 14   15  5.158
## 15   16  5.152
## 16   17  5.151
## 17   18  5.154
## 18   19  5.154
## 19   20  5.159
## 20   21  5.161
## 21   22  5.165
## 22   23  5.169
## 23   24  5.178
## 24   25  5.181
```

```
min_mse = result['MSE'].min()
print(min_mse)
```

```
## 5.151
```

```
print(min_mse * 0.01 + min_mse)
```

```
## 5.20251
```

Based on the table, the lowest MSE correspond to k=17 with MSE 5.151. Considering 1% margin of error, a safer number of k is 10 with MSE 5.195.

KNN model improvement

Features scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)

mses = []
```

```
ks = range(1, X_train_scaled.shape[1] + 1)
for k in ks :
  model = KNeighborsRegressor(n_neighbors=k)
  kf = KFold(n_splits=10, shuffle=True, random_state=100)
  fold_scores = -cross_val_score(model, X_train_scaled, y_train, cv=kf, scoring='neg_mean_squared_error
  mse = round(fold_scores.mean(), 4)
  mses.append(mse)

result = pd.DataFrame({'k': ks, 'MSE': mses})
print (result)

# find the lowest MSE and MSE with 1% margin of error
```

```
##       k     MSE
## 0     1  4.9879
## 1     2  3.8653
## 2     3  3.5187
## 3     4  3.3571
## 4     5  3.2593
## 5     6  3.2063
## 6     7  3.1658
## 7     8  3.1424
## 8     9  3.1230
## 9    10  3.1077
## 10   11  3.0946
## 11   12  3.0801
## 12   13  3.0760
## 13   14  3.0719
## 14   15  3.0692
## 15   16  3.0694
## 16   17  3.0723
## 17   18  3.0728
## 18   19  3.0713
## 19   20  3.0735
## 20   21  3.0795
## 21   22  3.0804
## 22   23  3.0845
## 23   24  3.0854
## 24   25  3.0885
```

```
min_mse = result['MSE'].min()
print(min_mse)
```

```
## 3.0692
```

```
print(min_mse * 0.01 + min_mse)
```

```
## 3.099892
```

Based on the table, we see that scaled data achieves lower overall MSE. The lowest MSE which is 3.0690 corresponds to k=15. Considering 1% margin of error, the next best number of k is 10 with MSE 3.099.

The optimal number of k=10 aligns with the result obtained from unscaled data; however, the scaled data achieved a significantly lower MSE.

Dimentionality reduction (PCA)

As the optimal number of k, achieved from both scaled and unscaled data is 10, I specify n_neighbors=10 in KNeighborsRegressor().

```
from sklearn.decomposition import PCA

mses = []
for component in range(1, X_train_scaled.shape[1] + 1):
    pca = PCA(n_components=component)
    X_train_new = pca.fit_transform(X_train_scaled)

    model = KNeighborsRegressor(n_neighbors=10)
    kf = KFold(n_splits=10, shuffle=True, random_state=100)
    fold_scores = -cross_val_score(model, X_train_new, y_train, cv=kf, scoring='neg_mean_squared_error')
    mse = round(fold_scores.mean(), 3)
    mses.append(mse)

result = pd.DataFrame({'PC': range(1, X_train_scaled.shape[1] + 1), 'MSE': mses})
print(result)


# find the lowest MSE and MSE with 1% margin of error
```

```
##     PC     MSE
## 0    1  10.361
## 1    2   8.386
## 2    3   4.145
## 3    4   3.819
## 4    5   3.561
## 5    6   3.401
## 6    7   3.267
## 7    8   3.084
## 8    9   3.057
## 9   10   3.112
## 10  11   3.108
## 11  12   3.109
## 12  13   3.109
## 13  14   3.113
## 14  15   3.117
## 15  16   3.118
## 16  17   3.120
## 17  18   3.113
## 18  19   3.111
## 19  20   3.107
## 20  21   3.106
## 21  22   3.108
## 22  23   3.108
## 23  24   3.108
## 24  25   3.108
```

```
min_mse = result['MSE'].min()
print(min_mse)
```

```
## 3.057
```

```
print(min_mse * 0.01 + min_mse)
```

```
## 3.08757
```

As indicated by the table, the dimensionality with minimum MSE is 9 with MSE 3.057. Considering 1% margin of error, the optimal dimensionality is 8 with MSE 3.087.

In conclusion, we have achieved the best performance so far with the model using scaled data, k=10, and pc=8.