Noah Caulfield

CS 460

Dr. Edgar Eduardo Ceh Varela

3/27/2024

HW4: Classification Search Optimal Parameters

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

```
(4898, 12)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847	10.514267	5.877909
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126	1.230621	0.885639
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000	5.000000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000	6.000000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000	6.000000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000	9.000000

```
There are no missing values in the DataFrame.
```

```
First few rows of features (X):
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9

```
display(y.head())
```

```
0    6
1    6
2    6
3    6
4    6
Name: quality, dtype: int64
```

Split train and test

```
(3918, 11)
(980, 11)
(3918,)
(980,)
```

```
[[5.15119310e-01 -1.07623315e+00 2.27730764e-01 3.40419470e-01
  -8.13688488e-01 5.34064605e-01 -6.41932319e-01 -4.47040725e-01
  -3.28261014e-01 -7.02444738e-01 1.54037099e+00]
 [-6.69188091e-01 -2.88776731e-01 8.95831948e-01 1.00207124e+00
  -2.17211567e-01 7.73947112e-01 1.35510550e+00 9.03369755e-01
  -6.18856911e-02 2.66074147e-01 -8.21711966e-01]
 [-1.49820327e+00 4.00247639e-01 -2.28071805e-02 1.84736700e-01
  -4.00742927e-01 -6.05377303e-01 -1.02232048e+00 -4.60280044e-01
  4.04271124e-01 1.93263316e-03 4.81506217e-01]
 [4.13963498e-02 -8.79369048e-01 1.44218115e-01 -9.24503038e-01
  -4.46625767e-01 -1.25612289e-01 -8.79674917e-01 -3.04718052e-01
  1.37895801e-01 4.42168490e-01 2.37152807e-01]
 [9.88842271e-01 2.03383533e-01 -6.07395717e-01 2.43240669e+00
  3.33382515e-01 5.42995912e-02 8.55846045e-01 1.88307932e+00
  7.13019704e-02 8.99798045e-02 -8.86517384e-02]]
```

```
array([0.47959184, 0.4744898 , 0.45535714, 0.45849298, 0.47254151])
```

```
Average accuracy: 0.46809465165376496
```

This is the accuracy of our model using 5 neigh

```
[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, 30, 31]
```

```
{'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, 30, 31]}
```

```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

Let's train the grid with our X_train and y

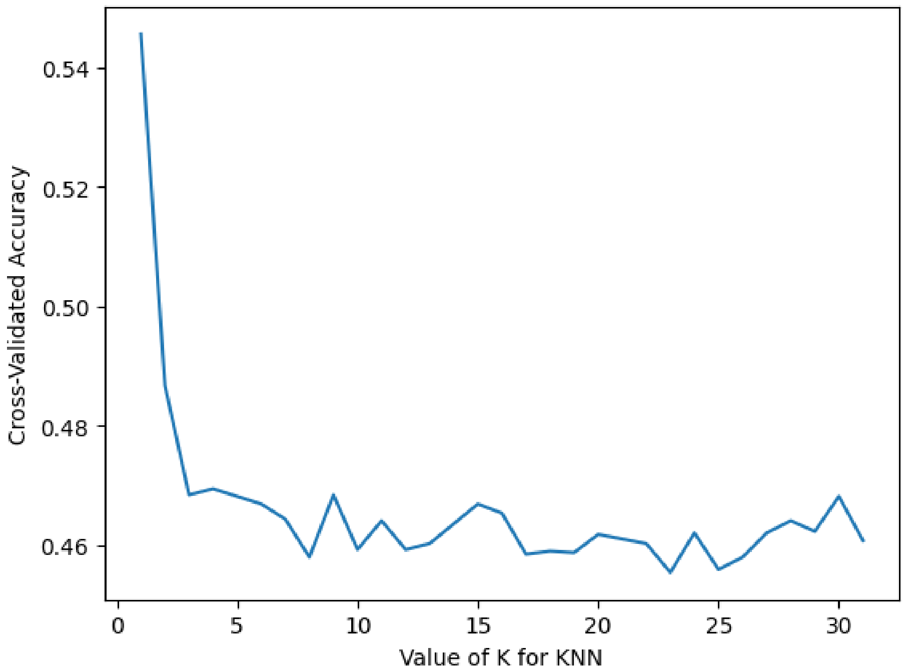
```
GridSearchCV
GridSearchCV(cv=5, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                         13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                                         23, 24, 25, 26, 27, 28, 29, 30, ...]},
             scoring='accuracy')
estimator: KNeighborsClassifier
KNeighborsClassifier
KNeighborsClassifier()
```

	mean_test_score	std_test_score	params
0	0.545685	0.008051	{'n_neighbors': 1}
1	0.486729	0.018125	{'n_neighbors': 2}
2	0.468358	0.016629	{'n_neighbors': 3}
3	0.469377	0.009407	{'n_neighbors': 4}
4	0.468095	0.009458	{'n_neighbors': 5}
5	0.466822	0.005355	{'n_neighbors': 6}
6	0.464265	0.007483	{'n_neighbors': 7}
7	0.457890	0.008705	{'n_neighbors': 8}
8	0.468353	0.011294	{'n_neighbors': 9}
9	0.459169	0.013004	{'n_neighbors': 10}
10	0.464011	0.011681	{'n_neighbors': 11}
11	0.459161	0.008404	{'n_neighbors': 12}
12	0.460180	0.010164	{'n_neighbors': 13}
13	0.463498	0.010682	{'n_neighbors': 14}
14	0.466817	0.016665	{'n_neighbors': 15}
15	0.465283	0.012302	{'n_neighbors': 16}
16	0.458394	0.010249	{'n_neighbors': 17}
17	0.458905	0.008230	{'n_neighbors': 18}
18	0.458653	0.007690	{'n_neighbors': 19}
19	0.461712	0.007131	{'n_neighbors': 20}
20	0.460946	0.011894	{'n_neighbors': 21}

```
[0.54568542 0.48672916 0.4683579  0.46937668 0.46809465 0.4668224
 0.46426519 0.45788959 0.46835334 0.45916934 0.46401139 0.45916087
 0.46017965 0.46349825 0.46681654 0.46528266 0.45839426 0.45890511
 0.45865262 0.46171221 0.46094593 0.46017899 0.45533075 0.46196569
 0.45583802 0.45787884 0.46196504 0.46400553 0.4622234  0.46809205
 0.46069213]
```

Let's visualize the scores

```
Text(0, 0.5, 'Cross-Validated Accuracy')
```




```
Best Score: 0.5456854197617744
Best Parameters: {'n_neighbors': 1}
Best Estimator: KNeighborsClassifier(n_neighbors=1)
```

```
{'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, 30, 31], 'weights': ['uniform', 'distance']}
```

GridSearchCV

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'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, 30, ...],
                         'weights': ['uniform', 'distance']},
             scoring='accuracy')
```

estimator: KNeighborsClassifier

```
KNeighborsClassifier()
```

KNeighborsClassifier

```
KNeighborsClassifier()
```

	mean_test_score	std_test_score	params
0	0.545685	0.008051	{'n_neighbors': 1, 'weights': 'uniform'}
1	0.545685	0.008051	{'n_neighbors': 1, 'weights': 'distance'}
2	0.486729	0.018125	{'n_neighbors': 2, 'weights': 'uniform'}
3	0.545685	0.008051	{'n_neighbors': 2, 'weights': 'distance'}
4	0.468358	0.016629	{'n_neighbors': 3, 'weights': 'uniform'}
...	...	...	...
57	0.591881	0.008306	{'n_neighbors': 29, 'weights': 'distance'}
58	0.468092	0.020879	{'n_neighbors': 30, 'weights': 'uniform'}
59	0.593667	0.011521	{'n_neighbors': 30, 'weights': 'distance'}
60	0.460692	0.020216	{'n_neighbors': 31, 'weights': 'uniform'}
61	0.593410	0.015712	{'n_neighbors': 31, 'weights': 'distance'}

62 rows × 3 columns

```
0.5936667578908958
{'n_neighbors': 30, 'weights': 'distance'}
```

```
Accuracy of the final model on the test dataset: 0.4826530612244898
```

What is the accuracy of your final model using the test dataset?

Write the code for getting that result and answer the following questions

- Does the model prsents overfitting?

Yes, the model does present some overfitting, which is specifically demonstrated by the accuracy of the final model within the test dataset, compared to the best score found above. If the model performs significantly better on the training dataset than on the test dataset, it may indicate overfitting. Overfitting occurs when the model learns the training data too well, including its noise and outliers, making it perform poorly on unseen data. In total the model is relatively in line, however I do feel as though there are some instances of overfitting.

- What could you test to improve your model?

Additional data can always help to strengthen and improve a model- including this one. Hyperparameter tuning can be helpful. Also further addressment of the complexity of the problem/model along with the implementation of different algorithms may be helpful.

- Is accuracy the best metric for this problem? Yes/No and why?

Accuracy is a valid and acceptable metric for this sort of problem, as the "Correctness" of said problem is based on the analysis of the 1-9 input variables per wine type/name. However, the validity of these input variables may be seen as subjective, and the accuracy of identification is contingent concluding with the correct figures/metrics and them being accurately compared. Also, the larger the dataset gets, the harder this level of "Accuracy" will be able to predict- due to crossings within certain identifiable thresholds.

Note: There is another method called `RandomizedSearchCV` which randomly searches a subset of parameter. You can play with this method to see the differences