SciKit-learn: KNN Implementation

Christian Rodriguez, Cole Ballard, James Miller, & Temesgen Fekadu

Import Libraries and Data Set

```
#KNN Workbook for Group6
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from matplotlib import pyplot as plt
import numpy as np
from sklearn import datasets

wine = datasets.load_wine()
```

SciKit Learn:

- Standard Scalar
- K-Neighbors Classifier
- Train Test Split
- Metric
- Grid Search CV

Pandas, Pyplot, Numpy

Cleaning the data

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  target labels                 178 non-null    int32
```

- The dataset was already optimal to work with.
- All columns were numeric types
- No columns contained any null values

Checking the Data for Outliers

```
for col in data.columns:
    if data[col].dtype == 'float64':
        low_bound = data[col].mean() - (3 * data[col].std())
        upper_bound = data[col].mean() + (3 * data[col].std())
        for value in data[col].values:
            if (value < low_bound or value > upper_bound):
                print(f'{col} has an outlier with value of {value}')
```

Outliers were defined as values that were +/- 3 standard deviations from the mean of each column.

```
malic_acid has an outlier with value of 5.8
ash has an outlier with value of 3.22
ash has an outlier with value of 1.36
ash has an outlier with value of 3.23
alcalinity_of_ash has an outlier with value of 30.0
magnesium has an outlier with value of 151.0
magnesium has an outlier with value of 162.0
flavanoids has an outlier with value of 5.08
proanthocyanins has an outlier with value of 3.58
color_intensity has an outlier with value of 13.0
hue has an outlier with value of 1.71
```

Compared to the rest of the data, these did not seem so abnormal that we should drop them. We determined that the noise would help us avoid overfitting the model.

Train-Test-Split and Scale

We trained the model on 75% of the data, leaving the remaining 25% for testing

```
#Create our train and test sets 75/25
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

Then the features were standardized using the StandardScaler utility

```
#Scaling the data after train_test split
X_train_scaled = pd.DataFrame(StandardScaler().fit_transform(X_train), columns = X_train.columns)
X_test_scaled = pd.DataFrame(StandardScaler().fit_transform(X_test), columns = X_test.columns)
```

Training the Model

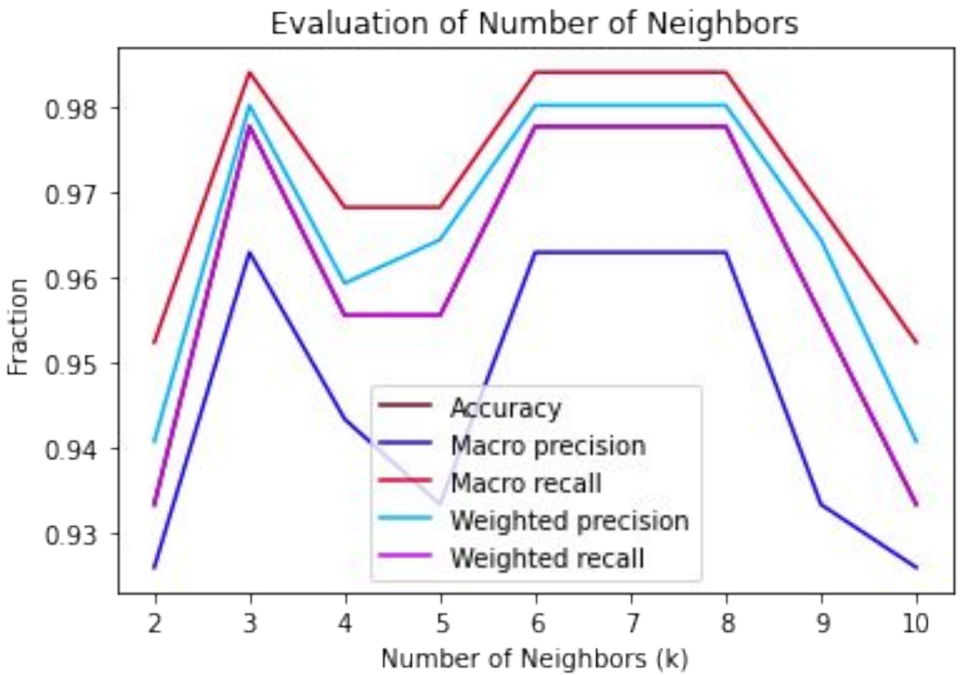
```
# Make k-NN models in list
knn = []
predictions = []
for k in range(2, 11):

    # Form the model
    model = KNeighborsClassifier(n_neighbors=k)

    # Train each model
    model.fit(X_train_scaled, y_train)

    # Predict the values
    predictions.append(model.predict(X_test_scaled))

    # Add to model list
    knn.append(model)
```

We created models using a range of k=2 to k=10

Each model was trained

Then each model was tested

Evaluating the Models



The classification report was used to plot the evaluation of the number of neighbors, k.

Evaluating the Models, cont'd

```
#Let's see what values we get for k=7 (this would be values in predictions[5])
print(metrics.classification_report(y_test, predictions[5], target_names = target_names))
```

```
              precision    recall  f1-score   support

     class 0       1.00      1.00      1.00        16
     class 1       1.00      0.95      0.98        21
     class 2       0.89      1.00      0.94         8

    accuracy                           0.98        45
   macro avg       0.96      0.98      0.97        45
weighted avg       0.98      0.98      0.98        45
```

Question: Can this model be optimized via other hyperparameter tuning?

Tuning the Hyperparameters

Tuning hyperparameters with GridSearchCV

```python
# Source: https://medium.datadriveninvestor.com/k-nearest-neighbors-in-python-hyperparameters-tuning-716734bc557f

# List Hyperparameters that we want to tune.
leaf_size = list(range(1,50))
n_neighbors = list(range(2, 11))
p=[1, 2]

# Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)

# Create new KNN object
knn_hp = KNeighborsClassifier()

# Use grid search to find the ideal hyperparamters
clf = GridSearchCV(knn_hp, hyperparameters, cv=5)
```

Tuning the Hyperparameters, cont'd

```
# Fit the model
best_model = clf.fit(X_train_scaled, y_train)
```

```
Best leaf_size: 1
Best p: 1
Best n_neighbors: 5
```

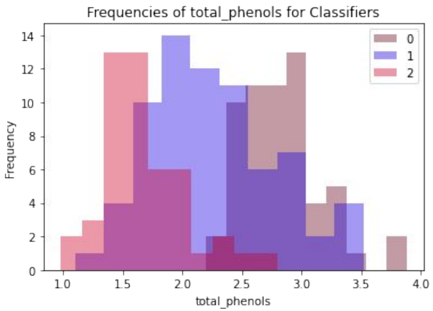
	precision	recall	f1-score	support
class 0	1.00	1.00	1.00	16
class 1	1.00	0.95	0.98	21
class 2	0.89	1.00	0.94	8
accuracy			0.98	45
macro avg	0.96	0.98	0.97	45
weighted avg	0.98	0.98	0.98	45

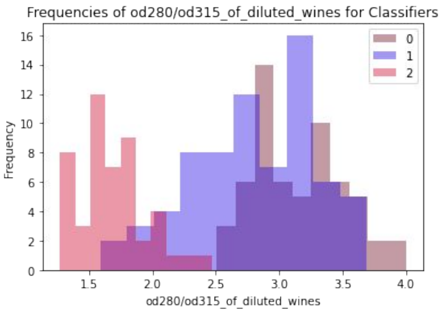
Can this model be optimized via feature engineering?

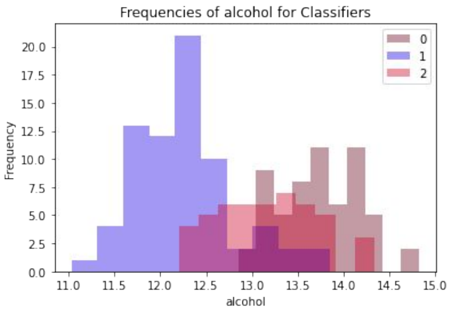
Removing Columns to Improve Performance

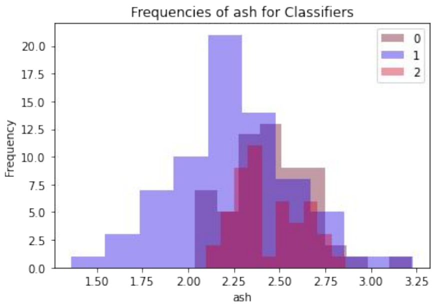
```
# Plot classification map
for column in data.drop(columns = 'target labels').columns:
    fig, ax = plt.subplots()
    i = 0
    for classifier in data['target labels'].unique():
        this_data = data[data['target labels'] == classifier]
        ax.hist(this_data[column], color=colors[i], alpha=0.4, label=classifier)
        i += 1
    ax.set_xlabel(f'{column}')
    ax.set_ylabel('Frequency')
    ax.set_title(f'Frequencies of {column} for Classifiers')
    ax.legend(facecolor='white')
    plt.show()
```

Creating histograms for each column to determine which columns should be removed to improve the model's performance









Feature Engineering

```
for col in data.columns:
    drop_columns = [col]
    new_data = data.drop(columns=drop_columns).copy()

    # Separate out classfiers
    X_d = new_data.copy().drop(columns=['target labels'])
    y_d = new_data.copy()['target labels']

    # Generate training and testing set: 25%
    X_d_train, X_d_test, y_d_train, y_d_test = train_test_split(X_d, y_d, test_size=0.25, random_state=0)

    #Scaling the data after train_test split
    X_d_train_scaled = pd.DataFrame(StandardScaler().fit_transform(X_d_train), columns = X_d_train.columns)
    X_d_test_scaled = pd.DataFrame(StandardScaler().fit_transform(X_d_test), columns = X_d_test.columns)

    # Create new KNN object
    knn_d_hp = KNeighborsClassifier()

    # Use grid search to find the ideal hyperparamters
    clf_d = GridSearchCV(knn_d_hp, hyperparameters, cv=5)

    # Fit the model
    best_model_d = clf_d.fit(X_d_train_scaled, y_d_train)

    #Print The value of best Hyperparameters
    print('Best leaf_size:', best_model_d.best_estimator_.get_params()['leaf_size'])
    print('Best p:', best_model_d.best_estimator_.get_params()['p'])
    print('Best n_neighbors:', best_model_d.best_estimator_.get_params()['n_neighbors'])

    # Show the metrics of the best model
    prediction_d = best_model_d.predict(X_d_test_scaled)
    print(f'Removal of column {col}.')
    print(metrics.classification_report(y_d_test, prediction_d))
```

Looping through each column in the dataframe and seeing how the model is affected by removing that column

Improvements to the Model

The model became less accurate or didn't change at all after removing each column except the od280/od315_of_diluted_wines column. After removing that column the model reached 100% accuracy.

```
Best leaf_size: 1
Best p: 1
Best n_neighbors: 9
Removal of column malic_acid.
              precision    recall  f1-score   support

           0       0.89      1.00      0.94        16
           1       1.00      0.81      0.89        21
           2       0.80      1.00      0.89         8

    accuracy                           0.91        45
   macro avg       0.90      0.94      0.91        45
weighted avg       0.92      0.91      0.91        45
```

```
Best leaf_size: 1
Best p: 1
Best n_neighbors: 6
Removal of column od280/od315_of_diluted_wines.
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        21
           2       1.00      1.00      1.00         8

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
Best leaf_size: 1
Best p: 2
Best n_neighbors: 7
Removal of column proanthocyanins.
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.95      0.98        21
           2       0.89      1.00      0.94         8

    accuracy                           0.98        45
   macro avg       0.96      0.98      0.97        45
weighted avg       0.98      0.98      0.98        45
```

Conclusion

- K (number of neighbors) was the hyperparameter with the largest impact
- The removal of one column resulted in a perfect model performance on the test data

	precision	recall	f1-score	support
class 0	1.00	1.00	1.00	16
class 1	1.00	0.95	0.98	21
class 2	0.89	1.00	0.94	8
accuracy			0.98	45
macro avg	0.96	0.98	0.97	45
weighted avg	0.98	0.98	0.98	45

Removal of column od280/od315_of_diluted_wines.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	16
1	1.00	1.00	1.00	21
2	1.00	1.00	1.00	8
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

Questions

Resources

https://github.com/jackrlynn3/ml-k-nearest-neighbors

https://medium.datadriveninvestor.com/k-nearest-neighbors-in-python-hyperparameters-tuning-716734bc557f

[Link2ColabNB]