K-Nearest Neighbor Algorithm (KNN) Practicum

Abstract

The purpose of this assignment is to design a KNN algorithm from scratch to classify the age of the Abalone using the abalone dataset provided by the UCI Machine Learning repository.

In one residual Rings, we have observed an accuracy prediction of 23.0% (with the model the Train Split method and Five Cross validation method. We have also compared the impact of the Ring value at 5,10,15 and the relation of the Rings value at 10. In terms of accuracy improvements, we could certainly improve the accuracy by using the Rings value as a feature in the model. Although, we have found that the Sex feature contributes negatively if it is not the dependent variable for the model accuracy.

Introduction

The UCI Machine Learning Learning repository has unique Rings, simply since the training data is messy and not well-balanced data is provided, and lack of better measures, we have decided to use the Rings variable to classify the sex query instance.

In the abalone dataset, there are a total of 8177 data observations with 8 input attributes and 1 output variable. The input attributes are as follows:

1. Sex (Male, Female, I) - refers to
2. Length - mm
3. Diameter - mm
4. Height - mm
5. Whole weight - g
6. Shucked weight - g
7. Viscera weight - g
8. Shell weight - g
9. Rings (output)

RUNS of Train-TEST SPLIT CODEBASE

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=5 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.23044444444444444
Predicted age: 10
```

RUNS of Train-TEST SPLIT CODEBASE

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=10 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```

PART 2: Tabulation of accuracy values of K-neighbors, with Train-Test-Split and CV methods

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=15 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=15)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```

RUNS of CROSS VALIDATION CODEBASE

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=5 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```

PART 3: Tabulation of accuracy values of K-neighbors, with Train-Test-Split and CV methods

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=10 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```

PART 4: Tabulation of computational time values of K-neighbors, with Train-Test-Split and CV methods

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=15 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=15)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```

Comment on Graph of Accuracy against the range of K-neighbors

From the graph, we observed that the number of neighbors has a negative correlation with the prediction accuracy.

Intuitively, this means that the model performs better when the accuracy gives from an average of 5-10 neighbors. Although, we observed that the accuracy of the model with the Train-Test-Split method is more gradually and consistent in almost linear fashion, all the way to 15.

From the graph, we observed that the prediction time for K-neighbors with k-value = 5 is averaged-value greater or equal to 3 in comparison to the constant and possibly applicable to other case studies too.

Although, we observed that the computational time for the Cross Validation method is constantly higher than the Train-Test-Split method, we observed that the computational time for both methods increases. Conversely, the slopes of the train-test-split time are steeper than those of the cross validation ones. Thus, the K-nn is a lazy algorithm, the model has to take time to predict a query instance.

PART 5: Tabulation of computational time values of K-neighbors, with Train-Test-Split and CV methods

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=10 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```

Comment on Graph of Computational Time against the range of K-neighbors

From the graph, we observed and understand that the number of neighbors has a negative correlation relationship with computation time taken for cross validation time. On comparison, with the computation time taken for train-test-split time, we observed that the computational time for cross validation is more than the train-test-split time. Although, we observed that the computational time for both methods increases.

Intuitively, we know that it is inappropriate to assume that cross validation is not recommended over train-test-split because it takes more time to train the model. Although, cross validation takes more time to train the model, it is more robust and reliable. Cross validation is appropriate in certain scenarios where the different blocks of data are working on.

Cross validation is useful up-front very little data because splitting it into training and test set might leave us with a very small set.

Cross validation is useful when we are able to get more metrics and draw important conclusion (which about our algorithm and data). The train-test-split is fast and accuracy is similar to all our folds, but 5-10 folds (CV), 0.21 and 0.23. This means that our algorithm and our data is consistent and we can be confident that the training set is representative of the data and generalizes well and is not overfitting.

PART 6: Classification report on 5-fold cross validation and K=15

Pandas informs us how accurate the predictions were correct. Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true positives and false positives. Recall is the ability of a classifier to find all positive instances.

Recall is the fraction of positives that were correctly identified. It is the ability of a classifier to find all positive samples. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives. Recall = TP/(TP+FN).

The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0. With F-beta score, beta = 1 gives equal weight to precision and recall. The support is the number of samples of each class in the given data.

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=15 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=15)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```

PART 7: continue ... Classification summary report on 5-fold cross-validation and K=15

The weighted average of the precision, recall and F1 score of the 5-fold cross validation K=15 are between 0.21 and 0.23. The confusion matrix is shown below and and we will interpret it.

Interestingly, we can see from the table that the higher numbers of the precision, recall and F1 score are associated with the rings value of 10 and the lowest numbers are associated with the rings value of 5. Although, the proportion of the support means that is not the cause. The middle part of the class support is where the mean lies.

Scatterplot of test and predicted data of the Rings

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=15 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=15)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```

Comment on the Scatter Plot (as above)

The plot shows that the test data of the Rings is centered at 9.87% The green line shows the mean of the y pred value at 9.7 for

Interestingly, we can observe that the actual test data (blue) is slightly more dispersed northwards on the y-axis compared to the predicted data.

They are concludes that the mean of the actual data is correspondingly slightly higher than the mean of the predicted value.

Bar plot of Frequency count against the Rings number (age)

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=15 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=15)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```


Plot of Confusion Matrix Display

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=15 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=15)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```

Confusion Matrix Display

As above, we have generated a confusion matrix plot here. We can observe that the highest' general' predicted value is generally located in the diagonal. This means that our model has generally been accurate in predicting the rings.

We can compare with the standard confusion matrix table shown above. It points out that the highest' general' predicted value is in the Pred Rings PRED 10 field. In fact, it has an 1999 digit in classifying our test predicted values with

Heatmap of Pairwise Correlation among the features and label

```
# Loading data and cleaning it
df = pd.read_csv('abalone.csv')
df.shape, df.isnull().sum()
df.iloc[0:5, 0:5]
# Initialize the model with k=15 neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=15)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predicting the test set
y_pred = knn.predict(X_test)
# Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Predicting the age of a new abalone
new_abalone = [[0, 0.455, 0.365, 0.095, 0.514]]
predicted_age = knn.predict(new_abalone)
print("Predicted age:", predicted_age[0])
```

```
(8177, 9) Unnamed: 0    0
Length     0
Diameter   0
Height     0
Whole     0
dtype: int64
   Unnamed: 0  Length  Diameter  Height  Whole
0           0   0.455     0.365   0.095  0.514
1           1   0.350     0.265   0.090  0.226
2           2   0.530     0.420   0.135  0.677
3           3   0.440     0.365   0.125  0.516
4           4   0.330     0.255   0.080  0.205
Accuracy: 0.2304444444444444
Predicted age: 10
```

Heatmap of Pairwise Correlation among the features and label

The correlation heatmap shown above is useful enough to give us a meaningful insight on the correlation relationship between the input features and the output. Rings. Sex is a categorical variable.

From the heatmap above, we can observe that the highest correlation factor of 0.41 with Rings is the Shell weight. Although, we observed that the Sex feature has a negative correlation factor of -0.04 with Rings with a correlation factor of 0.04 with the Length and Diameter features, we observed that the Length and Diameter are almost identical, therefore sex is a very demeaning feature. Although, we observed that the Length and Diameter are almost identical, therefore sex is a very demeaning feature and therefore it contributes negatively to the model accuracy.

Let's look at the features correlation with one another. Among the features, we can observe that Whole weight has a correlation factor of 0.97 with Viscera, 0.96 with Shucked and 0.93 with Shell. Although, we observed that Whole weight is highly correlated with the other features, we decided to keep the Whole weight feature, we can interpret feature selection to eliminate highly correlated features for further improve the model.

From the heatmap, we can observe that Whole Weight, Viscera, Shucked Weight and Viscera Weight have a perfect positive correlation of 1.0 with one another. Shell Weight comes closely behind and with them to range of correlation factor of 0.99 with Length and Diameter. Although, we observed that Whole weight is highly correlated with the other features, we decided to keep the Whole weight feature, we can interpret feature selection to eliminate Whole Weight, Shucked Weight, Viscera Weight, Viscera Weight and Shell Weight as the school correlation factor is very high with the Whole weight.

Final Thoughts

As predicted in case of 2D-CNN, I think we can do much more to improve the KNN algorithm and performance between sections to improve the performance.

ADDITIONAL CLEANING & FEATURE SELECTION

Machine Learning models are as good or as bad as the data you have. Thus, pre-processing and cleaning the data is an important step in the process. Although, the KNN algorithm is suitable for classification and prediction problems, it is also important to pre-process the data before implementing the algorithm. Although, the algorithm is robust to noisy data, it is important to take the necessary steps to ensure that the data is suitable for the algorithm. We could have used an outlier detection method to remove outliers from the dataset. Although, we could have used a feature selection method to select the most relevant features for the model, we could also have used a dimension reduction algorithm such as Principle Component Analysis (PCA).

KNN ALGORITHM IMPROVEMENT

The KNN algorithm is a non-parametric method used for classification in the current case. It comes by attraction that in the algorithm we used. In particular all, in the algorithm, it stores all instances in the correct. Given a distance or similarity function, it then will find. For example, K Rings of integer values and K Rings of integer values that fit in the 10 Rings. The algorithm then computes the distance between the query instance and the training.

The algorithm's primary drawback is that it calculates the distance of every all instance. Thus, it might be classified as Slow Distant prediction. We can go one step further to refine the algorithm. We could have used a weighted variation of the KNN algorithm. The improved version of the KNN algorithm is a hybrid method known as adaptive or weighted KNN. In the hybrid method, the distance function utilizes information from the sample which classifies the complete data points accurate as it learns other target.