Dohun Pyun - Abalone Classification

dp23r@fsu.edu , Electrical engineering

Abalone- EDA, Classification project

Data Description

Anticipating the age of abalone from physical estimations. The age of abalone is dictated by carving the shell through the cone, recoloring it, and tallying the number of rings through a magnifying instrument - an exhausting and tedious task. Different estimations, which are simpler to acquire, are utilized to anticipate the age. Additional data, for example, climate examples and area (thus food accessibility) might be required to take care of the issue.

Machine Learning Methods Used:

1. K- Nearest Neighbors
2. Support Vector Machines
3. Naive Bayes
4. CNN and NN

Importing Data

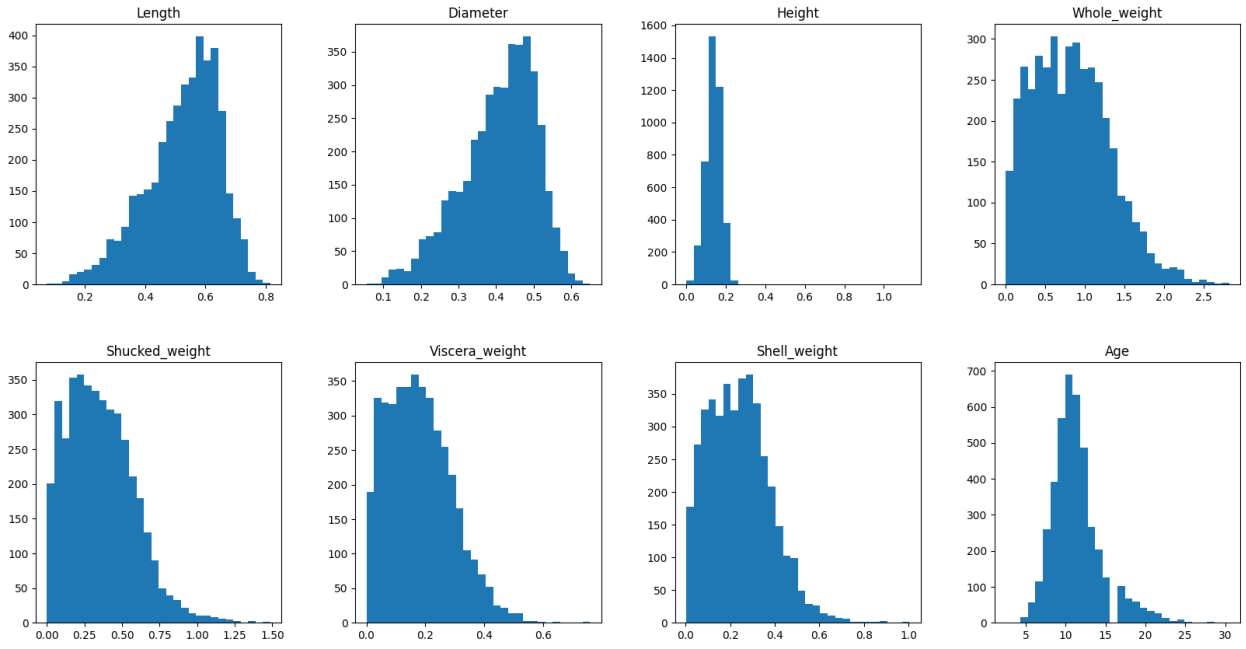
```
from ucimlrepo import fetch_ucirepo

# fetch dataset
```

```
abalone = fetch_ucirepo(id=1)

# data (as pandas dataframes)
x = abalone.data.features
y = abalone.data.targets
```

Data Visualization

1. **Histogram**

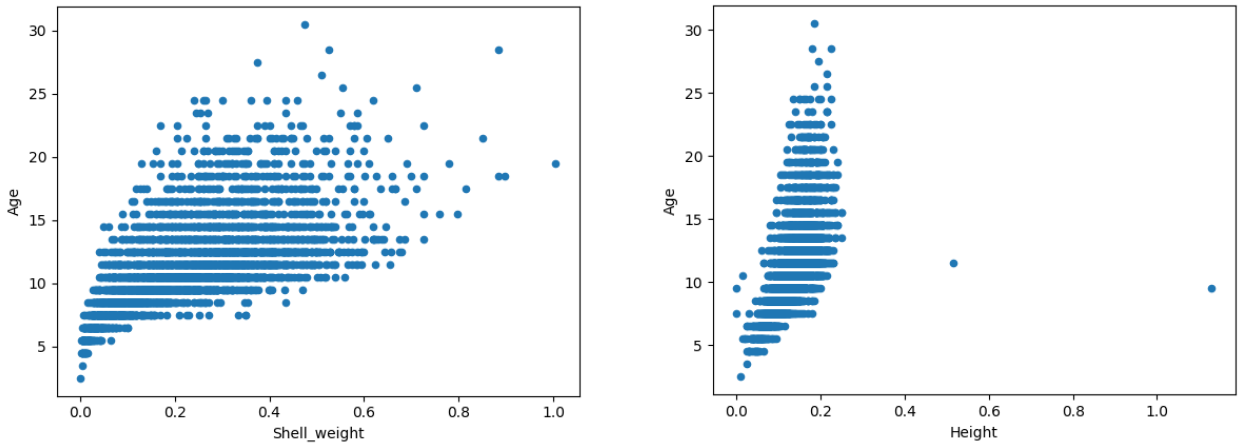
```
x.hist(figsize=(20,10), grid=False, layout=(2, 4), bins = 30)
```



2. **Analyzing the correlations with output and each input attribute and find outliers**

```
# Histogram visualisation for age output and Longest shell measurement input attributes
data_plot=pd.concat([x['Age'],x['Length']],axis=1)
```

```
data_plot.plot.scatter(x='Length',y='Age')

# Histogram visualisation for age output and Diameter input attributes.
data_plot=pd.concat([x['Diameter'],x['Age']],axis=1)
data_plot.plot.scatter(x='Diameter',y='Age')

# Histogram visualisation for age output and Height input attributes.
data_plot=pd.concat([x['Height'],x['Age']],axis=1)
data_plot.plot.scatter(x='Height',y='Age')

# Histogram visualisation for age output and Whole weight input attributes.
data=pd.concat([x['Whole_weight'],x['Age']],axis=1)
data.plot.scatter(x='Whole_weight',y='Age')

# Histogram visualisation for age output and Shucked weight input attributes.
data=pd.concat([x['Shucked_weight'],x['Age']],axis=1)
data.plot.scatter(x='Shucked_weight',y='Age')

# Histogram Visualisation for Shell weight input attribute and age output attribute.
data=pd.concat([x['Shell_weight'],x['Age']],axis=1)
data.plot.scatter(x='Shell_weight',y='Age')
```

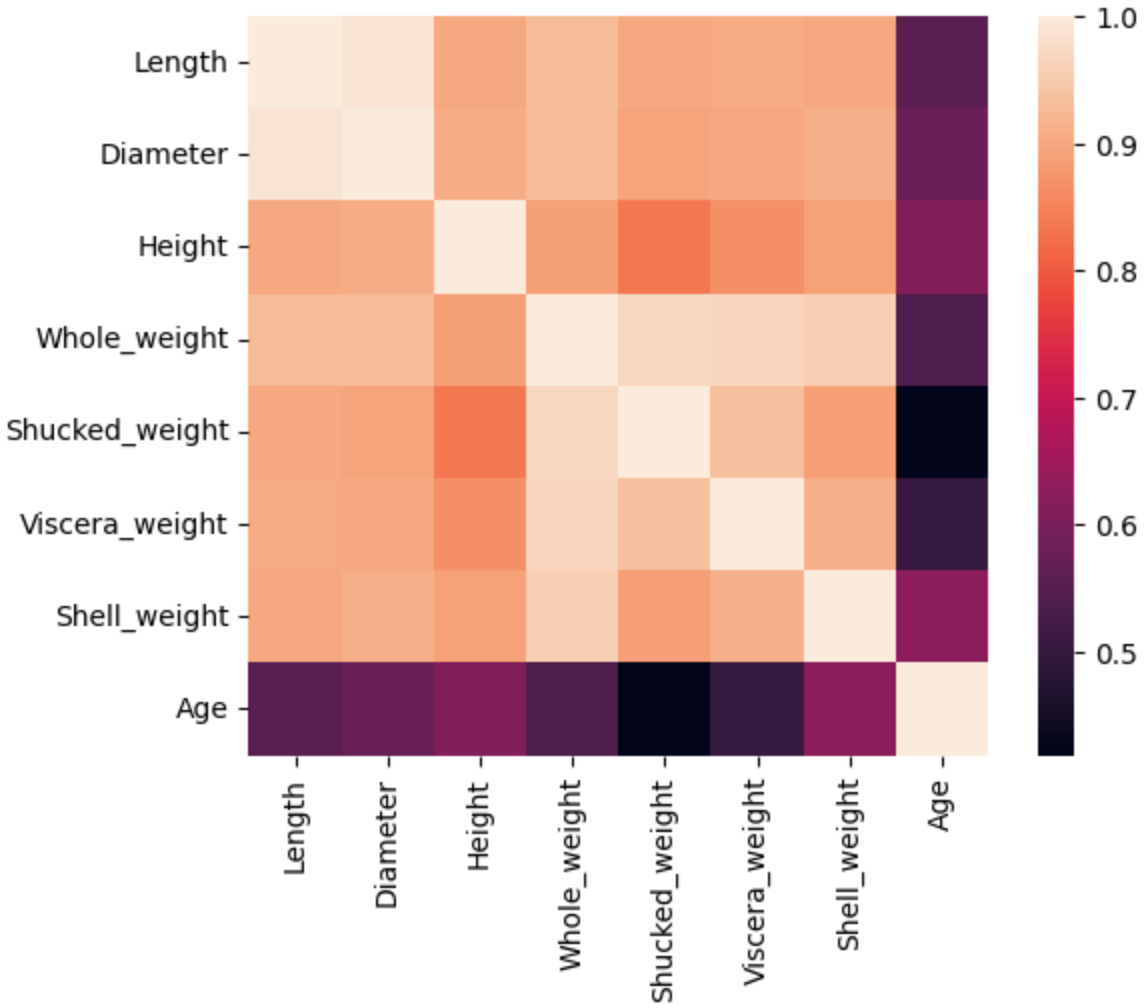


There are outliers between Age with Shell_weight and Height.

Removing outlier for age and two attributes.

3. **Correlation values between each attributes using heatmap**

```
# Correlation value with each attribute using heatmap
import seaborn as sb
correlation_values=x.corr()
sb.heatmap(correlation_values,square=True)
```



Data preprocessing

```
# Change Data for classification
bins = [0,8,10,y['Rings'].max()]
group_names = ['young','medium','old']
y['Rings'] = pd.cut(y['Rings'],bins, labels = group_names)
dictionary = {'young':0, 'medium':1, 'old':2}
x['age'] = y['Rings'].map(dictionary)
x.drop('Age', axis = 1, inplace = True)
```

```
x.head()

#LabelEncoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
x['Sex'] = le.fit_transform(x['Sex'])

#Train Test Split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.2)
```

Time to Train Model.

KNN

The k-Nearest-Neighbors method of classification it is essentially classification by finding the most similar data points in the training data, and making an educated guess based on their classifications. This method is used in areas like recommendation systems, semantic searching, and anomaly detection.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    ConfusionMatrixDisplay,
    f1_score,
)
paramsKn = {'n_neighbors':range(1,40)}
Kneighbours = GridSearchCV(KNeighborsClassifier(),paramsKn, cv=10)

Kneighbours.fit(X=X_train,y=y_train)
Kmodel = Kneighbours.best_estimator_
print(Kneighbours.best_score_, Kneighbours.best_params_)

y_pred = Kmodel.predict(X_test)
knn_accuray = accuracy_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test, average="weighted")

print("Accuracy:", knn_accuray)
```

```
print("F1 Score:", f1)
print('Classification Report: \n', classification_report(y_test, y_pred))
```

{'n_neighbors': 29}
Accuracy: 0.8023952095808383
F1 Score: 0.8156604961744769
Classification Report:

	precision	recall	f1-score	support
0	0.70	0.60	0.64	82
1	0.60	0.44	0.51	180
2	0.85	0.95	0.90	573
accuracy			0.80	835
macro avg	0.72	0.66	0.68	835
weighted avg	0.79	0.80	0.79	835

Using the KNN library contained in sklearn.

Using GridSearch to select the parameter (n_neighbors) in KNN.

The trained model has an accuracy of about 80.2% and an F1 score of 0.816.

The F1 score is a measure of a model's accuracy in statistical analysis, especially in the context of binary classification and other machine learning models. It considers both the precision and the recall of the test to compute the score. An F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

Support Vector Machine

The Support Vector Machine is a discriminative classifier formally defined by a separating hyperplane. The goal of the model is to output optimal hyperplane that will categorize the data into categories. There are many hyperplanes dividing data possible so the object is to find one that will maximize the distance from the line to the classes.

```
from sklearn.svm import SVC
paramsSvm = {'kernel':['linear', 'poly', 'rbf', 'sigmoid'],
                  'C':[0.1,1,10],'gamma':[0.01,0.1,0.5,1,2]}

Svm = GridSearchCV(SVC(),paramsSvm,cv=5)

Svm.fit(X_train,y_train)
model_svm = Svm.best_estimator_
print(Svm.best_score_,Svm.best_params_)

y_pred = model_svm.predict(X_test)
svm_accuray = accuracy_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test, average="weighted")

print("Accuracy:", svm_accuray)
print("F1 Score:", f1)
print('Classification Report: \n', classification_report(y_test, y_pred))
```

{'C': 1, 'gamma': 2, 'kernel': 'poly'}
Accuracy: 0.7952095808383234
F1 Score: 0.805440493904367
Classification Report:

	precision	recall	f1-score	support
0	0.69	0.65	0.67	82
1	0.58	0.44	0.50	180
2	0.86	0.93	0.89	573
accuracy			0.80	835
macro avg	0.71	0.67	0.69	835
weighted avg	0.78	0.80	0.78	835

Using the SVC library contained in sklearn.

Using GridSearch to select the parameter (Gamma, Kernel, Poly) in SVM.

Set parameter of Svm as {'kernel':['linear', 'poly', 'rbf', 'sigmoid'], 'C':[0.1,1,10],'gamma' [0.01,0.1,0.5,1,2]} Select best performance parameter by using Gridsearch.

The trained model has an accuracy of about 79.5% and an F1 score of 0.805.

Naive Baues Classifier

```
from sklearn.naive_bayes import GaussianNB

# Build a Gaussian Classifier
model_NB = GaussianNB()

# Model training
model_NB.fit(X_train, y_train)

from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    ConfusionMatrixDisplay,
    f1_score,
)

y_pred = model_NB.predict(X_test)
NB_accuray = accuracy_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test, average="weighted")

print("Accuracy:", NB_accuray)
print("F1 Score:", f1)
print('Classification Report: \n', classification_report(y_test, y_pred))
```

Accuracy: 0.7005988023952096
F1 Score: 0.6824100781526722
Classification Report:

	precision	recall	f1-score	support
0	0.51	0.85	0.64	82
1	0.40	0.54	0.46	180
2	0.91	0.73	0.81	573
accuracy			0.70	835
macro avg	0.61	0.71	0.64	835
weighted avg	0.76	0.70	0.72	835

CNN (Convolutional Neural Network)

CNN, or Convolutional Neural Network, is a type of deep learning algorithm primarily used for processing data with a grid-like topology, such as images. CNNs are particularly effective in tasks like image recognition, classification, and analysis, as well as in other areas like natural language processing and video analysis. The distinctive feature of CNNs lies in their ability to automatically and adaptively learn spatial hierarchies of features from input images.

```
# data preprocessing

# Labelencoding Sex to number
data['Sex'] = LabelEncoder().fit_transform(data['Sex'])

#  Rings + 1.5 -> Age
data['Age'] = data['Rings'] + 1.5
data['Age'] = pd.cut(data['Age'], bins=[0, 8, 10, 12, 15, 100], labels=[0, 1, 2, 3, 4])

# One hot encoding
y = pd.get_dummies(data['Age'])

# Data preprocessing
X = data.drop(['Rings', 'Age'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Change the Age data into the range (0, 4) for classification. Then, perform one-hot encoding for training in a neural network.

```
# Reshape Data
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
```

```
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Reshape the dataset into a three-dimensional format because CNNs use filters, and for filtering, we need to change the dataset into a three-dimensional structure.

```
# 1D-CNN model build
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape
[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax')) # 출력층

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)
```

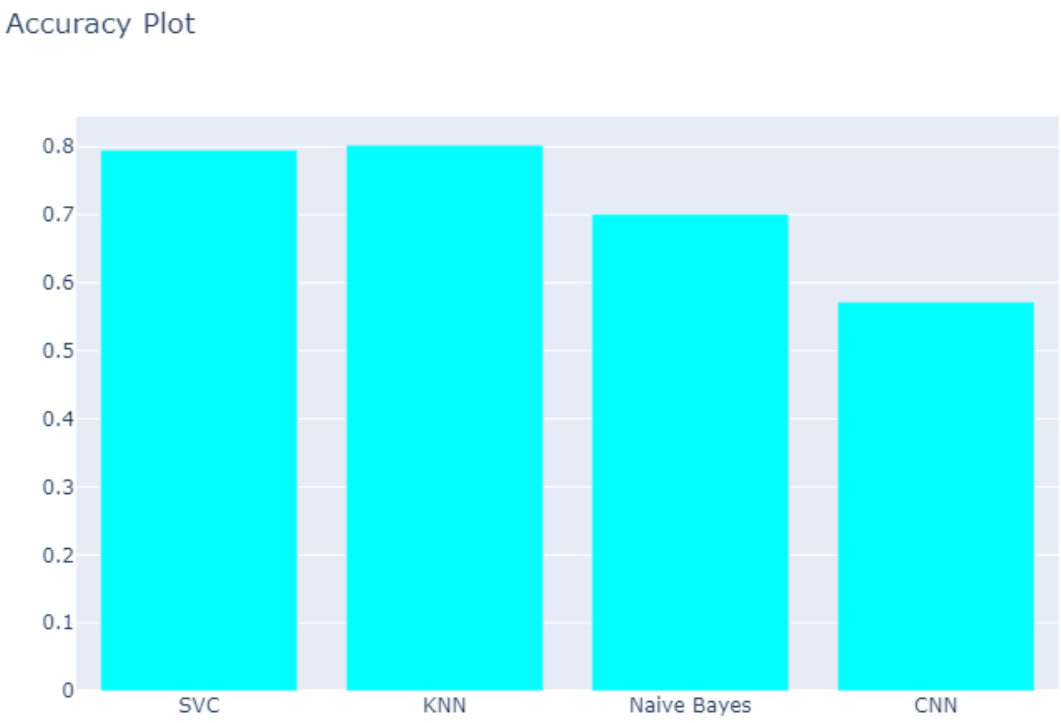
To begin, apply a filter size of (1,1) and then perform Maxpooling with a size of 2. Afterward, proceed to train the neural network.

Epoch 94/100
84/84 [==============================] - 0s 4ms/step - loss: 0.9351 - accuracy: 0.5719 - val_loss: 1.0442 - val_accuracy: 0.5142
Epoch 95/100
84/84 [==============================] - 0s 4ms/step - loss: 0.9332 - accuracy: 0.5752 - val_loss: 1.0555 - val_accuracy: 0.4993
Epoch 96/100
84/84 [==============================] - 0s 5ms/step - loss: 0.9362 - accuracy: 0.5696 - val_loss: 1.0338 - val_accuracy: 0.5306
Epoch 97/100
84/84 [==============================] - 0s 5ms/step - loss: 0.9332 - accuracy: 0.5752 - val_loss: 1.0383 - val_accuracy: 0.5262
Epoch 98/100
84/84 [==============================] - 0s 5ms/step - loss: 0.9358 - accuracy: 0.5741 - val_loss: 1.0511 - val_accuracy: 0.5157
Epoch 99/100

84/84 [==============================] - 0s 5ms/step - loss: 0.9298 - accuracy: 0.5786 - val_loss: 1.0588 - val_accuracy: 0.5037
Epoch 100/100
84/84 [==============================] - 0s 5ms/step - loss: 0.9332 - accuracy: 0.5715 - val_loss: 1.0423 - val_accuracy: 0.5366

Accuracy = 57,15%

Model summary



Neural Network

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from keras.models import Sequential
from keras.layers import Dense


data['Sex'] = LabelEncoder().fit_transform(data['Sex'])

: Rings + 1.5 -> Age)
data['Age'] = data['Rings'] + 1.5
data['Age'] = pd.cut(data['Age'], bins=[0, 8, 10, 12, 15, 100], labels=[0, 1, 2, 3, 4])

X = data.drop(['Rings', 'Age'], axis=1)
y = pd.get_dummies(data['Age'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)
```

Conclusion

KNN is the best performing model, while CNN is the worst model. Initially, I expected CNN to have the best performance. However, the performance of CNN is worse because it is a model designed for image data, while we are using a 2-dimensional dataset and artificially transforming it into a 3-dimensional format. As a result, I tried using a simple neural network without changing the dimension of the dataset, but the performance also decreased. I personally believe that the process of 'One hot encoding' is causing the decrease in performance. To better understand the exact reasons for the low performance of CNN and DNN, further study is needed. Additionally, I personally think that the reason for the good performance of KNN is because we are labeling age into 0, 1, 2. However, we still need to study to determine the exact reasons why KNN is the best model.