ABALONE AGE PREDICTION

INTRODUCTION

Abalone is a shellfish considered a delicacy in many parts of the world. Farmers usually cut the shells and count the rings through microscopes to estimate the abalones age. Telling the age of abalone is therefore difficult mainly because their size depends not only on their age, but on the availability of food as well. Moreover, abalone sometimes form the so-called 'stunted' populations which have their growth characteristics very different from other abalone populations This complex method increases the cost and limits its popularity.

OBJECTIVE OF RESEARCH

A dataset provided by the University of California Irvine Machine Learning Repository consists of physical characteristics of abalones and their ages. This study is a classification problem that aims to predict the age range of abalones using their physical characteristics. The data set is pre-processed to transform the problem of predicting the age to a classification problem. Two clustering algorithms, the k-means algorithm and a hierarchical clustering algorithm are and run to generate a large number of cluster centers. An web application is integrated to the model built.

LITERATURE REVIEW

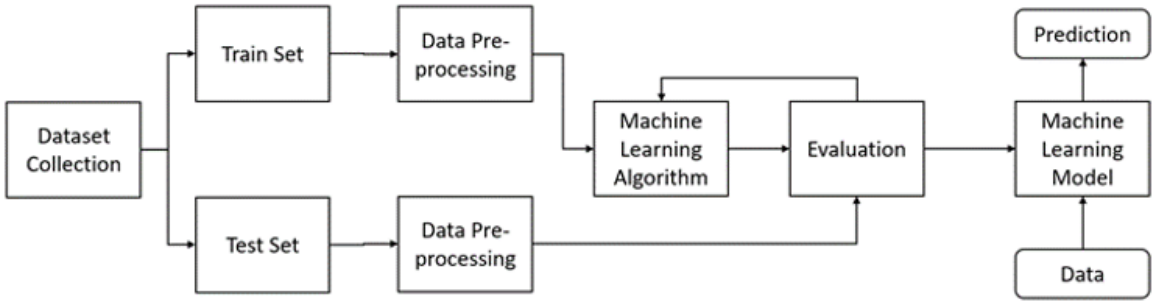
The abalone dataset was first published in 1995. Since then, copious amounts of research using many different algorithms and methods has been done, first among them being decision trees. In 1999, CLOUDS, a decision tree based algorithm was used to achieve a 26.4% accuracy on the abalone test dataset. [6] Typically in classification problems, the algorithm for selecting a split point for the dataset at

each internal node involves sorting the values of each numeric attribute, calculating the gini index at each possible split point and selecting the split point with the minimum gini value. This brute force method was found to be computationally expensive and challenging [6]. Thus, CLOUDS used a better approach called SSE [6] in which the range of each attribute in the dataset was divided into intervals using quantiling techniques, an estimation of gini values at the boundaries of these intervals was made and compared with the minimum of the actual gini values at the boundaries. Thereafter, the brute force method was applied only in the intervals where the estimated gini value was less than the minimum gini value. However, using SSE did not lead to any significant improvement in classification accuracy or tree size for abalone dataset over the sorting technique, which obtained an accuracy of 26.3%[6]. C4.5 is another decision tree technique which achieved only 21.5% accuracy.[4]

K-means clustering algorithm run on a preprocessed dataset with reduced classes (8 classes), all numeric attributes and with one-fourth of the dataset left out for testing results in an accuracy of 61.78% [7] The experiment also helps in determining the relative contribution of different attributes to the classification accuracy (in increasing order of importance): Sex, Length, Height, Whole weight, Shell weight, Viscera weight, Diameter, Shucked weight. [7]

THEORITICAL ANALYSIS

Machine Learning Workflow:



Project Flow :

- User interacts with the UI (User Interface) to enter the current attrition data.
- Entered data are analyzed and predictions are made based on interpretation of descriptive features.
- Predictions are popped onto the UI.

DATA COLLECTION

The dataset is related to abalones. It was taken from the below website:
https://archive.ics.uci.edu/ml/datasets/abalone
The UCI Learning Repository provides one dataset abalone data, it contains 4177 observations, 8 descriptive features and 1 target feature.

Let's now see the type and name of the features:

Name	Data Type	Measurement	Description
Sex	categorical (factor)		M, F, and I (Infant)
Length	continuous	mm	Longest shell measurement
Diameter	continuous	mm	perpendicular to length
Height	continuous	mm	with meat in shell
Whole weight	continuous	grams	whole abalone
Shucked weight	continuous	grams	weight of meat
Viscera weight	continuous	grams	gut weight (after bleeding)
Shell weight	continuous	grams	after being dried
Rings	continuous		+1.5 gives the age in years

DATA PRE-PROCESSING

Importing required Libraries:

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn.preprocessing import LabelEncoder
        from sklearn.preprocessing import OneHotEncoder
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import r2_score
        from sklearn.metrics import accuracy_score
        import joblib
        from sklearn.svm import SVC
        from sklearn.metrics import confusion_matrix
```

Pandas: It is a python library mainly used for data manipulation.

NumPy: This python library is used for numerical analysis.

Matplotlib and Seaborn: Both are the data visualization library used for plotting graph which will help us for understanding the data.

Accuracy score: used in classification type problem and for finding accuracy it is used.

R2 Score: Coefficient of Determination or R^2 is another metric used for evaluating the performance of a regression model. The metric helps us to compare our current model with a constant baseline and tells us how much our model is better.

Train_test_split: used for splitting data arrays into training data and for testing data.

joblib: to serialize your machine learning algorithms and save the serialized format to a file.

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality

reduction via a consistence interface in Python

Importing the dataset:

```
In [2]: data=pd.read_csv("abalone.csv")

In [3]: data.head()
```

Out[3]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program) and read it using a method called read_csv which can be found in the library called pandas. Here we are using a data set which you can find in the below link:

https://archive.ics.uci.edu/ml/datasets/abalone

Taking Care of missing Data:

Sometimes you may find some data are missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously, you could remove the entire line of data but what if you are unknowingly removing crucial information? Of course we would not want to do that. One of the most common ideas to handle the problem is to take a mean of all the values for continuous and for categorical we make use of mode values and replace the missing data.

We will be using **isnull().any()** method to see which column has missing values.

```
In [4]: data.isnull().any()

Out[4]: Sex               False
        Length            False
        Diameter          False
        Height            False
        Whole weight      False
        Shucked weight    False
        Viscera weight    False
        Shell weight      False
        Rings             False
        dtype: bool
```

Since there are no missing values in the dataset, no need to execute this step.

Label Encoding

Sometimes in the dataset we will find textual data like Sex, then the machine cannot do mathematical operations or cannot understand the textual data. So the textual data are to be converted in to numerical format which is called as label encoding. we make use of label Encoder class to convert textual data in to Numerical data. In the given dataset Sex has textual data so we will be converting that particular columns textual data to numerical values.
You have to apply this for every column which has textual data.

```
In [16]: data['Sex'].unique()

Out[16]: array(['M', 'F', 'I'], dtype=object)
```

```
In [17]: from sklearn.preprocessing import LabelEncoder
         le=LabelEncoder()
         data['Sex']=le.fit_transform(data['Sex'])
```

```
In [18]: data
```

Out[18]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	2	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	15
1	2	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9
3	2	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10
4	1	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7
...	...	...	...	...	...	...	...	...	...

Now the Sex column with names will be converted in to num.

Splitting Dataset in to Independent variable and Dependent variable:

To read the columns, we will use iloc of pandas (used to fix the indexes for selection) which takes two parameters — [row selection, column selection].

```
In [52]: x=data.iloc[:,0:8]
         y=data.iloc[:,8:]
```

From the above piece of code ":" indicates you are considering all the rows and "0:8" indicates we are considering column 0 to 8 as input values and assigning them to variable x. in the same way in second line ":" indicates you are considering all the rows and "8" indicates we are considering only one column 8 as output value and assigning them to variable y

Feature Scaling:

- It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.
- For feature scaling, we will import ***StandardScaler*** class of ***sklearn.preprocessing***
- Now, we will create the object of **StandardScaler** class for independent variables or features. And then we will fit and transform the training dataset.

```
In [69]: from sklearn.preprocessing import StandardScaler
         sc=StandardScaler()

In [70]: x=sc.fit_transform(x)

In [71]: x

Out[71]: array([[ 1.15198011, -0.57455813, -0.43214879, ..., -0.60768536,
                 -0.72621157, -0.63821689],
                [ 1.15198011, -1.44898585, -1.439929  , ..., -1.17090984,
                 -1.20522124, -1.21298732],
                [-1.28068972,  0.05003309,  0.12213032, ..., -0.4634999 ,
                 -0.35668983, -0.20713907],
                ...,
                [ 1.15198011,  0.6329849 ,  0.67640943, ...,  0.74855917,
                  0.97541324,  0.49695471],
                [-1.28068972,  0.84118198,  0.77718745, ...,  0.77334105,
                  0.73362741,  0.41073914],
                [ 1.15198011,  1.54905203,  1.48263359, ...,  2.64099341,
                  1.78744868,  1.84048058]])
```

Splitting The dataset in to Train set and Testing set:

● When you are working on a model and you want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset. For this, you will a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing.

● Now split our dataset into train set and test using train_test_split class from scikit learn library

```
In [72]: from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)

In [74]: x_train.shape,x_test.shape

Out[74]: ((3341, 8), (836, 8))

In [75]: y_train.shape,y_test.shape

Out[75]: ((3341, 1), (836, 1))
```

Model Building

Training and testing the model:

Now, comes the fun part where we finally get to use the meticulously prepared data for model building. Depending on the data type (qualitative or quantitative) of the target variable (commonly referred to as the **Y** variable) we are either going to be building a classification (if **Y** is qualitative) or regression (if **Y** is quantitative) model.

1. Linear Regression
2. Decision Tree Regression
3. Random Forest Regression
4. Support Vector Regression

➢ Linear Regression:

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

```
In [75]: from sklearn.linear_model import LinearRegression

         lr=LinearRegression()
         lr.fit(x_train,y_train)
         #y_test
```

```
Out[75]: LinearRegression()
```

```
In [76]: y_pred_lr=lr.predict(x_test)
         #y_pred_lr
```

```
In [77]: from sklearn.metrics import r2_score
         r2_score(y_test,y_pred_lr)
```

```
Out[77]: 0.5300147524184928
```

➢ Decision Tree Regression:

Decision tree builds **regression** or **classification** models in the form of a **tree** structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated **decision tree** is incrementally developed. **Decision trees** can handle both categorical and numerical data.

```
In [78]: from sklearn.tree import DecisionTreeRegressor
         dtr=DecisionTreeRegressor()
         dtr.fit(x_train,y_train)
         #y_test

Out[78]: DecisionTreeRegressor()

In [79]: y_pred_dtr=dtr.predict(x_test)
         #y_pred_dtr

In [80]: from sklearn.metrics import accuracy_score,confusion_matrix
         acc_dtr=accuracy_score(y_test,y_pred_dtr)

In [81]: acc_dtr

Out[81]: 0.18899521531100477
```

➢ Random Forest Regression:

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

```
In [83]: from sklearn.ensemble import RandomForestRegressor
         rfr=RandomForestRegressor()
         rfr.fit(x_train,y_train)
         #y_train

         <ipython-input-83-7bb52d863d2f>:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
         nge the shape of y to (n_samples,), for example using ravel().
           rfr.fit(x_train,y_train)

Out[83]: RandomForestRegressor()

In [84]: y_pred_rfr=rfr.predict(x_test)
         #y_pred_rfr

In [85]: acc_rfr=r2_score(y_test,y_pred_rfr)

In [86]: acc_rfr

Out[86]: 0.5658084220394961
```

- Support Vector Regression:

Supervised **Machine Learning** Models with associated **learning** algorithms that analyze data for classification and regression analysis are known as **Support Vector Regression**. **SVR** is built based on the concept of Support Vector **Machine** or SVM.

```
In [87]: from sklearn.svm import SVR
         svm=SVR()
         svm.fit(x_train,y_train)

         C:\Users\umesh\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning: A column-vector y was passed
         when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
           return f(**kwargs)
Out[87]: SVR()

In [88]: y_pred_svr=svm.predict(x_test)

In [89]: #y_pred_svr

In [90]: acc_svr=r2_score(y_test,y_pred_svr)

In [91]: acc_svr
Out[91]: 0.5559957143349785
```

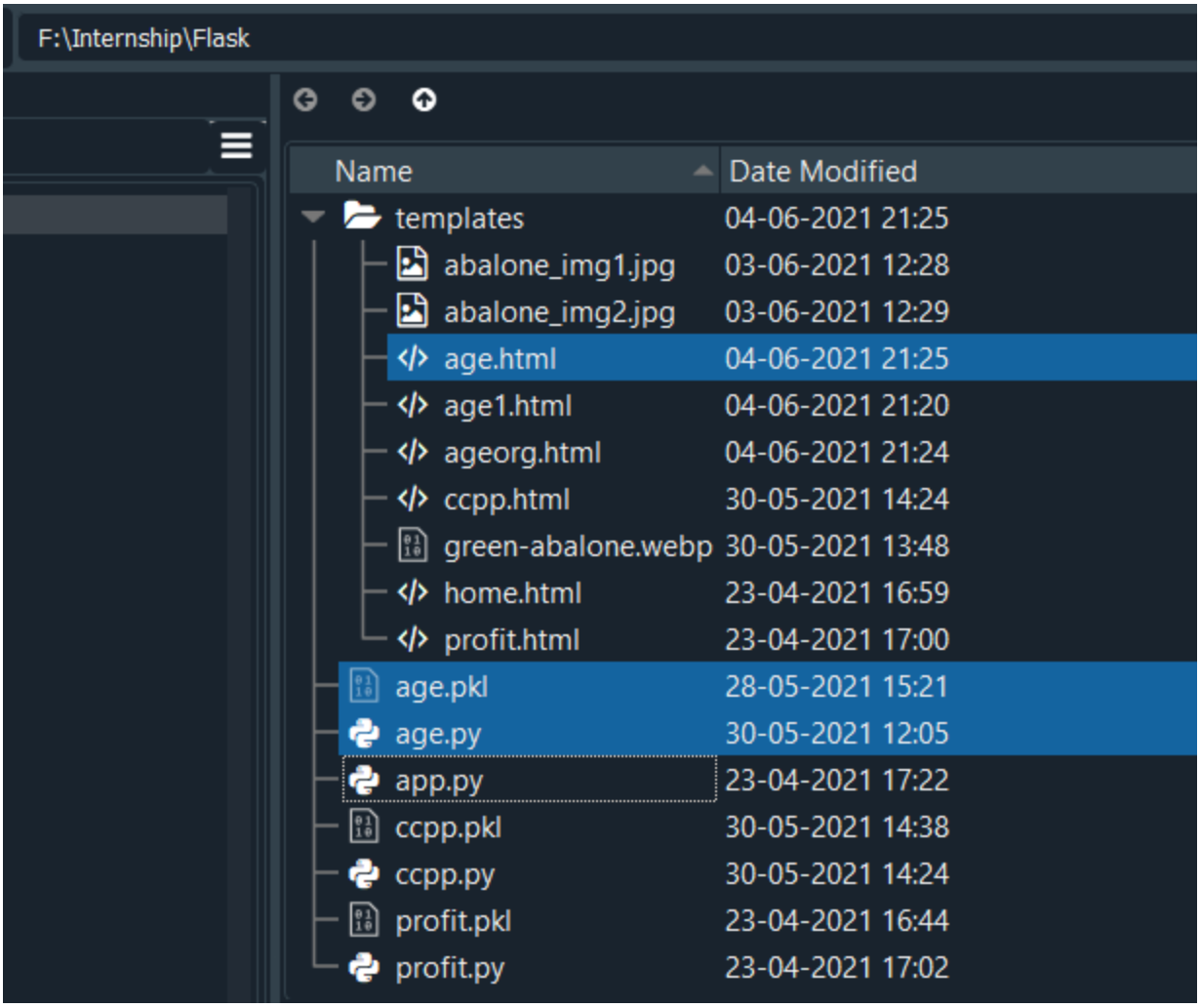
Saving a model:

Model is saved so it can be used in future and no need to train it again

```
In [92]: import pickle
         pickle.dump(rfr,open('age.pkl','wb'))
```

IV.Application Building

Creating a HTML File, flask application.



- We are building a Flask Application that needs HTML pages stored in the templates folder
- Templates folder contains index.html

Build python code

- **Importing Libraries**

```
from flask import Flask,render_template,request
import pickle
import numpy as np

app=Flask(__name__) #your application
rfr=pickle.load(open('age.pkl','rb'))
```

Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument Pickle library to load the model file.

- **Routing to the html Page**

Here, declared constructor is used to route to the HTML page created earlier.

```
@app.route('/') # default route
def home():
    return render_template("age.html")


@app.route('/predict',methods=['post'])
def predict():
    Sex=float(request.form['Sex'])
    Length=float(request.form['Length'])
    Diameter=float(request.form['Diameter'])
    Height=float(request.form['Height'])
    Whole_weight=float(request.form['Whole weight'])
    Shucked_weight=float(request.form['Shucked weight'])
    Viscera_weight=float(request.form['Viscera weight'])
    Shell_weight=float(request.form['Shell weight'])
    #Rings=float(request.form['Rings'])

    print(Sex,Length,Diameter,Height,Whole_weight,Shucked_weight,Viscera_weight,Shell_weight)
    a=np.array([[Sex,Length,Diameter,Height,Whole_weight,Shucked_weight,Viscera_weight,Shell_weight]])

    result=rfr.predict(a)

    return render_template('age.html',x=result)
```

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page is rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.
Here, "age.html" is rendered when home button is clicked on the UI

- **Main Function**

  This is used to run the application in a local host.

```
if __name__ == '__main__':
    app.run(port=8000) # you are running your app
```

- **Run The app in local browser**

  Open the *spyder* from the start menu.
  Navigate to the folder where your *age.py* resides.
  Now run "age.py" file.
  It will show the local host where your app is running on **http://127.0.0.1:8000/**
  Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
  Enter the values, click on the predict button and see the result/prediction on the web page.

```
In [1]: runfile('F:/Internship/Flask/age.py', wdir='F:/Internship/Flask')
 * Serving Flask app "age" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
```

Output Screen:

Findings and Suggestions

Through Exploratory Data Analysis,

- The Accuracy for Linear regression is 53.0%.
- The Accuracy for Random forest regression is 56.5%.
- The Accuracy for svm regressor is 55.5%.

Conclusion

On the source of this study it appears the future regression systems effort well to forecast the age of abalone. The study directs that we do not prerequisite to count the quantity of rings consuming microscopic test. In other disputes, we do not need any laboratory experiment to predict the age of abalones. We can predict the age and price of abalone using the very simple physical individualities like weight, height, diameter, and length.

Reference

1. www.kaggle.com
2. www.quora.com
3. www.wikkipedia.com