Classification

Analysis of Data

8 input features and 1 column ("Rings") to be predicted. Column 'Sex' have values as {M, F, I} and every other column has a numerical value. Therefore column 'Sex' needs to be one hot encoded. Column 'Rings' also need to be one hot encoded as this is a classification problem

Preprocessing Steps

Checked for duplicate and null values.
Used one-hot encoding on 'Ring' and 'Sex' column for classification. Therefore Ring is encoded into Ring_1 - Ring_29 and Sex into Sex_M, Sex_F, Sex_I.

Information about code

I have used an object-oriented approach by creating classes. To create a particular model, I am creating an object for that respective class and then applying fit function on it to improve the model.

Code Approach:

Logistic Regression: Used softmax function because of multiclass classification.
Naive Bayes: Gaussian fitting and then applied Bayes theorem assuming all features are independent.

Test and train errors

Algorithm	Features	Prediction	Accuracy (Train)	Accuracy (Test)
Univariate Logistic Regression	Length	Rings (20 vs not 20)	0.9916201117318436	0.9960095770151636
Multivariate Logistic Regression	All except (Rings)	Ring_1 - Ring_29	0.22825219473264166	0.24421388667198723
Univariate Naive Bayes	Length	Rings (20 vs not 20)	0.9916201117318436	0.9960095770151636
Multivariate Naive Bayes	All except (Rings)	Ring_1 - Ring_29	0.25591140377132593	0.29571984435797666

Preprocessing

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

Loading data

In [ ]:
```
main_df = pd.read_csv("./abalone.data",names=["Sex","Length","Diameter","
main_df.head()
```

Out[ ]:

	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

Data description

In [ ]:
```
main_df.describe()
```

Out[ ]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell w
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.0
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	0.2
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.1
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.0
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.1
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.2
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.3
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.0

Since Column 'Sex' has categorical value we have encoded it.

In [ ]:
```
main_df = pd.get_dummies(main_df, columns=['Sex'])
main_df.head()
```

Out[ ]:

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

Checking for null/empty values

In [ ]:
```
main_df.isna().sum()
```

Out[ ]:
```
Length            0
Diameter          0
Height            0
Whole weight      0
Shucked weight    0
Viscera weight    0
Shell weight      0
Rings             0
Sex_F             0
Sex_I             0
Sex_M             0
dtype: int64
```

In [ ]:
```
(main_df == "?").sum()
```

Out[ ]:
```
Length            0
Diameter          0
Height            0
Whole weight      0
Shucked weight    0
Viscera weight    0
Shell weight      0
Rings             0
Sex_F             0
Sex_I             0
Sex_M             0
dtype: int64
```

In [ ]:
```
main_df['Rings'].unique().shape
```

Out[ ]:
```
(28,)
```

In [ ]:
```
main_df['Rings'].describe()
```

Out[ ]:
```
count    4177.000000
mean        9.933684
std         3.224169
min         1.000000
25%         8.000000
50%         9.000000
75%        11.000000
max        29.000000
Name: Rings, dtype: float64
```

This is classification problem so we need to encode our prediction column

In [ ]:
```
main_df = pd.get_dummies(main_df, columns=['Rings'])
```

In [ ]:
```
print(main_df.sum())
```

```
Length            2188.7150
Diameter          1703.7200
Height             582.7600
Whole weight      3461.6560
Shucked weight    1501.0780
Viscera weight     754.3395
Shell weight       997.5965
Sex_F             1307.0000
Sex_I             1342.0000
Sex_M             1528.0000
Rings_1              1.0000
Rings_2              1.0000
Rings_3             15.0000
Rings_4             57.0000
Rings_5            115.0000
Rings_6            259.0000
Rings_7            391.0000
Rings_8            568.0000
Rings_9            689.0000
Rings_10           634.0000
Rings_11           487.0000
Rings_12           267.0000
Rings_13           203.0000
Rings_14           126.0000
Rings_15           103.0000
Rings_16            67.0000
Rings_17            58.0000
Rings_18            42.0000
Rings_19            32.0000
Rings_20            26.0000
Rings_21            14.0000
Rings_22             6.0000
Rings_23             9.0000
Rings_24             2.0000
Rings_25             1.0000
Rings_26             1.0000
Rings_27             2.0000
Rings_29             1.0000
dtype: float64
```

Common functions

Below softmax functions its softmax values for input vector lst

In [ ]:
```
from math import exp


def softmax(lst):
    for x in range(len(lst)):
        lst[x] = exp(lst[x])
    e_sum = sum(lst)
    for x in range(len(lst)):
        lst[x] = lst[x]/e_sum
    return lst
```

In [ ]:
```
print(sum(softmax([1,2,3])))
```

```
1.0
```

Accuracy function

In [ ]:
```
def accuracy(a,b):
    x,y = a.shape
    count = 0
    for i in range(x):
        if(a[i][b[i]] == 1):
            count = count + 1
    return count*1.0/x
```

Function to split data into training and testing

In [ ]:
```
def train_test_split(X,y,train_size):
    a = int(X.shape[0]*train_size)
    b = X.shape[0]-a
    return X[:a],X[b:],y[:a],y[b:]
```

Gaussian probability function

In [ ]:
```
def GPF(mu,sg,x):
    if(sg==0):
        return 1
    a = abs((x-mu)*1.0/sg)
    a = exp(-(a**2)/2)
    b = math.sqrt(2*math.pi)*sg
    return a/b
```

In [ ]:
```
GPF(0,1,0)
```

Out[ ]:
```
0.3989422804014327
```

Logistic Regression

In [ ]:
```
class LogReg:
    w = np.ones((1,1))
    def apply_softmax(self,X):
        a,b = X.shape
        for x in range(a):
            X[x] = softmax(X[x])
        return X

    def fit(self,w_ini,x,y,lr,itrs):
        global w
        a,b = x.shape
        x0 = np.ones((a,1))
        X = np.hstack((x0,x))
        for i in range(itrs):
            Z = np.matmul(X,w_ini)
            Z = self.apply_softmax(Z)
            diff = Z - y
            grad = diff.T @ X
            w = w_ini - grad.T/X.shape[0]
            w_ini = w
        return w

    def predict(self,x):
        global w
        a,b = x.shape
        x0 = np.ones((a,1))
        X = np.hstack((x0,x))
        sf = X @ w
        sf = self.apply_softmax(sf)
        return np.argmax(sf,axis=1)
```

Univariate Logistic Regression

In [ ]:
```
X = main_df['Length'].to_numpy().astype(np.float64)
y = main_df['Rings_20'].to_numpy().astype(np.float64)
X = np.reshape(X,(X.shape[0],1))

y = np.vstack((y,(y+1)%2)).T
train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.6)
```

Univariate Logistic model

In [ ]:
```
ulr_model = LogReg()
w = ulr_model.fit(np.zeros((train_X.shape[1]+1,train_y.shape[1])),train_X
```

Train accuracy

In [ ]:
```
print(accuracy(train_y,ulr_model.predict(train_X)))
```

```
0.9916201117318436
```

Test accuracy

In [ ]:
```
print(accuracy(test_y,ulr_model.predict(test_X)))
```

```
0.9960095770151636
```

Multivariate Logistic Regression

In [ ]:
```
X = main_df.iloc[:,:10].to_numpy().astype(np.float64)
y = main_df.iloc[:,10:].to_numpy().astype(np.float64)

train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.6)
```

Multivariate Logistic Regression model

In [ ]:
```
mlr_model = LogReg()
w = mlr_model.fit(np.zeros((train_X.shape[1]+1,train_y.shape[1])),train_X
```

Training accuracy

In [ ]:
```
print(accuracy(train_y,mlr_model.predict(train_X)))
```

```
0.22825219473264166
```

Testing accuracy

In [ ]:
```
print(accuracy(test_y,mlr_model.predict(test_X)))
```

```
0.24421388667198723
```

Naive Bayes

In [ ]:
```
class NB:
    gauss_fit_mean = np.ones((1,1))
    gauss_fit_std = np.ones((1,1))
    y_prob = np.ones((1,1))
    b = 0
    def gauss_fit_class(self,X,y,c):
        a = np.zeros((1,X.shape[1]))
        a = np.delete(a,obj=0,axis=0)
        for i in range(y.shape[0]):
            if(y[i][c]==1):
                a = np.vstack((a,X[i]))
        return np.mean(a,axis=0), np.std(a,axis=0)

    def bayesThm(self,mu,sg,x,p):
        ans = 1
        for xi in range(len(x)):
            ans = GPF(mu[xi],sg[xi],x[xi])*ans
        return ans*p

    def fit(self,X,y):
        global y_prob,gauss_fit_mean,gauss_fit_std,b
        a = X.shape[0]
        b = y.shape[1]
        gauss_fit_mean = np.zeros((b,X.shape[1]))
        gauss_fit_std = np.zeros((b,X.shape[1]))
        for i in range(b):
            gauss_fit_mean[i],gauss_fit_std[i] = self.gauss_fit_class(X,y
        y_prob = (np.sum(y,axis=0))/a
        prob = np.ones((a,b))
        for i in range(a):
            for j in range(b):
                prob[i][j] = self.bayesThm(gauss_fit_mean[j],gauss_fit_st
        return np.argmax(prob,axis=1)

    def predict(self,X):
        global y_prob,gauss_fit_mean,gauss_fit_std,b
        a = X.shape[0]
        prob = np.ones((a,b))
        for i in range(a):
            for j in range(b):
                prob[i][j] = self.bayesThm(gauss_fit_mean[j],gauss_fit_st
        return np.argmax(prob,axis=1)
```

Univariate Naive Bayes

In [ ]:
```
X = main_df.iloc[:,:1].to_numpy().astype(np.float64)
y = main_df['Rings_20'].to_numpy().astype(np.float64)
y = np.vstack((y,(y+1)%2)).T

train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.6)
```

Univariate naive bayes model

In [ ]:
```
unb_model = NB()
w = unb_model.fit(train_X,train_y)
```

Train Accuracy

In [ ]:
```
print(accuracy(train_y,unb_model.predict(train_X)))
```

```
0.9916201117318436
```

Test Accuracy

In [ ]:
```
print(accuracy(test_y,unb_model.predict(test_X)))
```

```
0.9960095770151636
```

Multivariate Naive Bayes

In [ ]:
```
X = main_df.iloc[:,:1].to_numpy().astype(np.float64)
y = main_df.iloc[:,10:].to_numpy().astype(np.float64)

train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.8)
```

Multivariate Naive Bayes model

In [ ]:
```
mnb_model = NB()
mnb_model.fit(train_X,train_y)
```

Out[ ]:
```
array([ 7,  6,  8, ..., 10,  8,  8])
```

Train accuracy

In [ ]:
```
print(accuracy(train_y,mnb_model.predict(train_X)))
```

```
0.25591140377132593
```

Test accuracy

In [ ]:
```
print(accuracy(test_y,mnb_model.predict(test_X)))
```

```
0.29571984435797666
```