CMP5130 Homework #1 - Naive Bayes Classifier

Important Notes

--You should SUBMIT A SINGLE PDF DOCUMENT You should SUBMIT A SINGLE PDF DOCUMENT You should SUBMIT A SINGLE PDF DOCUMENT--

Submit your homework through itslearning. You should SUBMIT A SINGLE PDF DOCUMENT as detailed below.

This homework is due December 10, 2020 until 23:55.

Submissions after this date will not be evaluated. The submission link will be deactivated in this specific date.

You are asked to implement naive bayes classifier on abalone dataset. Note: Do not use a library or use a source code from internet, implement it yourself!

Detailed information about abalone dataset can be found at http://archive.ics.uci.edu/ml/datasets/Abalone

The aim of the dataset is to predict the age of abalone from physical measurements. Originally it is a regression problem in which the output is age in years. However, we will use it as a classification problem. The age value is already discretized as young, middle-aged, and old. The dataset (input features and class labels of the samples) is provided as a seperate text file (abalone_dataset.txt).

input: Sex,Length,Diameter,Height,Whole weight,Shucked weight,Viscera weight,Shell weight

output: class label (less than 8 in age belongs to class 1 (young), between 8 and 12 to class 2 (middle-aged), greater than 12 to class 3 (old))

Optimization on validation set is not required in Naive Bayes classification. So, the dataset will be divided into training and validation sets only (i.e. there will not be a test set).

You are asked to implement naive bayes classifier for the following four cases:

1) Assume gaussian distribution for continuous features. Report the accuracies for each of the following case:

1.1) 100 samples for training, and rest for validation set 1.2) 1000 samples for training, and rest for validation set

2) Use Histogram Estimator for each of the continuous feature. Determine the bin size for each feature with your own criterion. Report the accuracies for each of the following case:

2.1) 100 samples for training, and rest for validation set 2.2) 1000 samples for training, and rest for validation set

For each of the above cases,

- Report how many total misclassification errors are there on the training and validation sets, together with the confusion matrices. (Note: A confusion matrix is a 3x3 matrix (if # of classes is 3) where entry (i,j) contains the number of instances belonging to i but are assigned to j; ideally it should be a diagonal matrix.)
- Report the case in which highest accuracy is obtained. Write your comments about the results.

You should SUBMIT A SINGLE PDF DOCUMENT which contains the following: a) a report that gives your results and comments. b) all the the program code (source code) that does the calculation in an executable format.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
from sklearn.model_selection import StratifiedShuffleSplit
from scipy.stats import norm
from sklearn.metrics import confusion_matrix
```

In [2]:
```
df = pd.read_csv('abalone_dataset.txt', sep="\t", header=None, names=["Sex", "Length", "Diameter",
                                                              "Height", "Whole weight", "Shucked weight",
                                                              "Viscera weight", "Shell weight","Class"])
df.shape
```

Out[2]:
```
(4177, 9)
```

In [3]:
```
df.describe()
```

Out[3]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Class
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	0.238831	2.028968
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.139203	0.655710
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.001500	1.000000
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.130000	2.000000
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.234000	2.000000
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.329000	2.000000
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	3.000000

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Class           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
```

In [5]:
```
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Gaussian Probability Density Function

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

In [6]:
```
def gaussian_distribution(train,index,value,distinct_class):
    mean = np.mean(train[(train["Class"] == distinct_class)][index])
    std = np.std(train[(train["Class"] == distinct_class)][index])
    return norm.cdf(value, mean, std)
```

In [7]:
```
def calculate_class_probablity(train,distinct_class):
    prob = train[train["Class"]==distinct_class].shape[0]/train["Class"].shape[0]
    return prob
```

In [8]:
```
def calculate_category_probablity(train,index,value,distinct_class):
    sub_count = train[(train["Class"]==distinct_class) & (train[index]==value)].shape[0]
    count = train[train["Class"]==distinct_class]["Class"].shape[0]
    prob = sub_count/count
    return prob
```

In [9]:
```
def calculate_probablity(train,val_row):
    max_class = -1
    max_value = -1
    for distinct_class in np.sort(train.Class.unique()):
        vector = 1
        for index,val in val_row.iteritems():
            if train[index].dtypes == np.float64:
                vector = vector * gaussian_distribution(train,index,val,distinct_class)
            elif train[index].dtypes == np.int64:
                vector = vector * calculate_class_probablity(train,distinct_class)
            else:
                vector = vector * calculate_category_probablity(train,index,val,distinct_class)
        if vector > max_value:
            max_class = distinct_class
            max_value = vector
    return(max_class)
```

In [10]:
```
def calculate_likelihoods(train,validation):
    y_pred = []
    for i in range(len(validation.index)):
        y_pred.append(calculate_probablity(train,validation.iloc[i]))
    return pd.Series(y_pred)
```

In [11]:
```
def split_data(df,split_size):
    size = (str(round(1-split_size/df.shape[0], 5)))
    sss = StratifiedShuffleSplit(n_splits=5, test_size=float(size), random_state=42)
    sss.get_n_splits(df)
    for train_index, test_index in sss.split(df,df["Class"]):
        train = df.loc[train_index]
        validation = df.loc[test_index]
    return train,validation
```

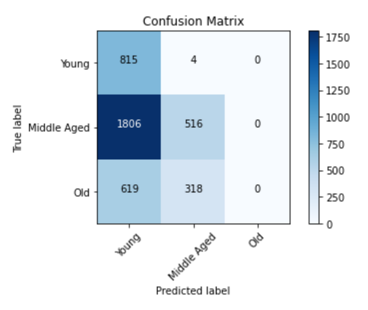
Bayes Classifier Formula

$$\hat{y} = argmax P(y) \prod_{i=1}^{n} P(xi \mid y)$$

In [12]:
```
def naive_bayes(df,size):
    train,validation = split_data(df,size)
    y_pred = calculate_likelihoods(train,validation)
    c_matrix = confusion_matrix(validation["Class"], y_pred)
    print("Confusion Matrix:")
    print(c_matrix)

    print("Total misclassification errors:")
    classification_errors = c_matrix[0][1]  + c_matrix[0][2] + c_matrix[1][0] + c_matrix[1][2] + c_matrix[2][0] + c_
matrix[2][1]
    print(classification_errors)

    print("Accuracy:")
    accuracy = ((c_matrix[0][0] + c_matrix[1][1] + c_matrix[2][2]) / (len(df))) * 100
    print("%s%%" % accuracy)

    plt.figure()
    plot_confusion_matrix(c_matrix, classes=["Young", "Middle Aged", "Old"],title='Confusion Matrix')
    plt.show()
```

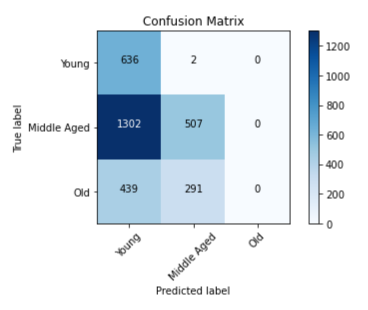
In [13]:
```
naive_bayes(df,100)
```

```
Confusion Matrix:
[[ 815    4    0]
 [1806  516    0]
 [ 619  318    0]]
Total misclassification errors:
2747
Accuracy:
31.864974862341395%
```



In [14]:
```
naive_bayes(df,1000)
```

```
Confusion Matrix:
[[ 636    2    0]
 [1302  507    0]
 [ 439  291    0]]
Total misclassification errors:
2034
Accuracy:
27.36413694038784%
```



In [15]:
```
def calculate_histogram_estimation(train,index,val,bin_width,distinct_class):
    min_data_point = train[index].min()
    max_data_point = train[index].max()
    interval_array = np.arange(min_data_point, max_data_point, bin_width).tolist()
    max_res = list(filter(lambda i: i > val, interval_array))
    min_res = list(filter(lambda i: i < val, interval_array))
    if len(max_res) >0:
        max_res = max_res[0]
    else:
        return 0
    if len(min_res) >0:
        min_res = min_res[-1]
    else:
        return 0
    sub_train = train[train["Class"] == distinct_class]
    observations = sub_train[(sub_train[index] > min_res) & (sub_train[index] < max_res)].shape[0]
    n = (bin_width*sub_train[index].shape[0])
    estimation = observations/n
    return estimation
```

In [16]:
```
def calculate_estimators(train,val_row):
    max_class = -1
    max_value = -1
    for distinct_class in np.sort(train.Class.unique()):
        vector = 1
        for index,val in val_row.iteritems():
            if train[index].dtypes == np.float64:
                bin_width = calculate_optimal_bin_width(train,index,distinct_class)
                vector = vector * calculate_histogram_estimation(train,index,val,bin_width,distinct_class)
            elif train[index].dtypes == np.int64:
                vector = vector * calculate_class_probablity(train,distinct_class)
            else:
                vector = vector * calculate_category_probablity(train,index,val,distinct_class)
        if vector > max_value:
            max_class = distinct_class
            max_value = vector
    return(max_class)
```

Freedman–Diaconis rule

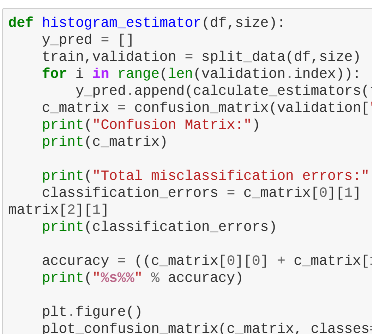
$$Binwidth = 2\frac{IQR(x)}{\sqrt[3]{n}}$$

In [17]:
```
def calculate_optimal_bin_width(train,index,distinct_class):
    q75, q25 = np.percentile(train[train["Class"] == distinct_class][index], [75 ,25])
    iqr = q75 - q25
    return(2*iqr*np.power(train[train["Class"] == distinct_class][index].shape[0],-1/3))
```

In [18]:
```
def histogram_estimator(df,size):
    y_pred = []
    train,validation = split_data(df,size)
    for i in range(len(validation.index)):
        y_pred.append(calculate_estimators(train,validation.iloc[i]))
    c_matrix = confusion_matrix(validation["Class"], y_pred)
    print("Confusion Matrix:")
    print(c_matrix)

    print("Total misclassification errors:")
    classification_errors = c_matrix[0][1]  + c_matrix[0][2] + c_matrix[1][0] + c_matrix[1][2] + c_matrix[2][0] + c_
matrix[2][1]
    print(classification_errors)

    accuracy = ((c_matrix[0][0] + c_matrix[1][1] + c_matrix[2][2]) / (len(df))) * 100
    print("%s%%" % accuracy)

    plt.figure()
    plot_confusion_matrix(c_matrix, classes=["Young", "Middle Aged", "Old"],title='Confusion Matrix')
    plt.show()
```

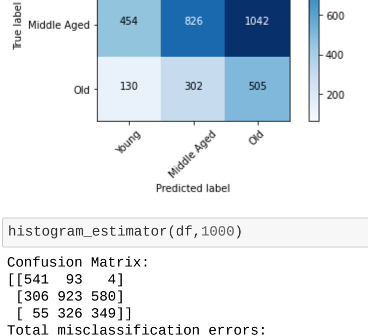
In [19]:
```
histogram_estimator(df,100)
```

```
Confusion Matrix:
[[ 688   68   63]
 [ 454  826 1042]
 [ 130  302  505]]
Total misclassification errors:
2059
48.336126406511845%
```



In [20]:
```
histogram_estimator(df,1000)
```

```
Confusion Matrix:
[[541  93   4]
 [306 923 580]
 [ 55 326 349]]
Total misclassification errors:
1364
43.404357194158486%
```



Conclusion

Naive-Bayes

Gaussian Distribution

For 100 samples: %31.86\ For 1000 samples: %27.36

Histogram Estimator

For 100 samples: %48.33\ For 1000 samples: %43.40

Naive Bayes Problem in some data points

This algorithm faces the ‘zero-frequency problem’ where it assigns zero probability to a categorical variable whose category in the test data set wasn’t available in the training dataset. That can cause mislabeling on classes. Smoothing techniques such as Laplace smoothing can solve this problem.

Why both algorithms are better in low training sample counts?

This situation can be explained by Hughes Phenomenon. Hughes phenomenon is a phenomenon that the classification precision increases gradually in the beginning as the number of spectral bands or dimensions increases, but when the band numbers reached at some point, the estimation accuracy begin to decrease dramatically.

Why Histogram Estimator is better than Gaussian Distribution in this case?

Gaussian distribution takes standart deviation and mean to compute probablities. If variables are too close in data set detecting proper class will be hard therefore accuracy will be low than expected. However, histogram estimator focuses on density calculation instead of standard deviation. In Histogram estimator, close data points will change just density whereas, in Gaussian distribution data points are changing both mean and standard deviation for all data points.