Li Chen & Li-Jung Chen

Data Mining Final Report-

Implement Naive Bayes Classifier from scratch to predict if a person makes over 50K a year.

Data Preparation:

```
mydata_data = pd.read_csv(r"/Users/lichen/Documents/adult.data.csv")
mydata_test = pd.read_csv(r"/Users/lichen/Documents/adult.test.csv")
#input the data which wants to be predicted
mydata_predict = pd.read_csv(r"/Users/lichen/Documents/adult.predict.csv")
#merge data
mydata = pd.read_csv(r"/Users/lichen/Documents/adult.data.merge.csv")

Y = mydata['label']
Y = Y.to_frame(name='label')
X = mydata
cols = list(mydata.columns.values)

X1_train, X1_test, Y1_train, Y1_test = train_test_split(X , Y , test_size=.4, random_state=42)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X , Y , test_size=.3, random_state=42)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(X , Y , test_size=.2, random_state=42)
```

Using hold out method and split the data into 60-40 of training-testing. (70-30/80-20)

Algorithm for calculating continuous attributes' probability

```
def gaussianProbability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1/(math.sqrt(2*math.pi)*stdev))*exponent
```

Data Training

```
def naive_bayse(X_train, Y_train,X_test,Y_test,pos_label):
    cols = list(X_train.columns.values)          #store attributes in cols
    cols.remove('label')
    labels = Y_train.label.unique()             #store label in labels
    df_global = pd.DataFrame(index=labels, columns=['Pr'])
    df_global = df_global.fillna(0)

    #Training ---------------------------------
    start_t = time.time()
    for label in labels:
        probability = len(X_train[X_train['label'] == label]) / len(X_train)
        df_global.loc[label, 'Pr'] = probability

    for col in cols:
        if((X_train[col].dtypes) == "object"):
            uniques = X_train[col].unique().tolist()        #store uniques class from the attributes
            exec("df_{0} = pd.DataFrame(index=uniques, columns=labels)".format(col.replace('-','")))
            exec("df_{0} = df_{1}.fillna(0)".format(col.replace('-',""), col.replace('-',"")))
            for unique in uniques:
                for label in labels:
                    count = len(X_train.loc[(X_train['label'] == label) & (X_train[col] == unique)])
                    total = len(X_train[X_train['label'] == label])
                    probability = float(count)/total
                    if(probability == 0):   #Modify the probability when the probability becomes to zero
                        for unique in uniques:
                            for label in labels:
                                count = len(X_train.loc[(X_train['label'] == label) & (X_train[col] == unique)])+1
                                total = len(X_train[X_train['label'] == label])+ (len(uniques))
                                probability = float(count)/total
                                exec("df_{0}.loc[unique, label] = probability".format(col.replace('-', "")))
                        break
                        break
                    elif(probability != 0):
                        exec("df_{0}.loc[unique, label] = probability".format(col.replace('-',"")))
            #exec("print(df_{0})".format(col.replace('-',"")))

        elif((X_train[col].dtypes) == "int64"):
            rows = ['mean','stdev']
            exec("df_{0} = pd.DataFrame(index=rows, columns=labels)".format(col.replace('-', "")))
            exec("df_{0} = df_{1}.fillna(0)".format(col.replace('-', ""), col.replace('-', "")))
            for label in labels:
                mean = statistics.mean((X_train.loc[(X_train['label'] == label)])[col])
                stdev = statistics.stdev((X_train.loc[(X_train['label'] == label)])[col])
                exec("df_{0}.loc['mean', label] = mean".format(col.replace('-', "")))
                exec("df_{0}.loc['stdev', label] = stdev".format(col.replace('-', "")))
            #exec("print(df_{0})".format(col.replace('-', "")))
    print("training time: ",time.time()-start_t)
```

Treat categorical data and continuous data differently for probability calculation.
Categorical data probability was obtained by counting number divided by the total class number.
And continuous data was obtained by implementing gaussian probability algorithm.
Saved the probability output result in the data frame. The outputs are like this

Continuous data

	>50K	<=50K
mean	44.041451	36.738198
stdev	10.283604	13.594688

Categorical data

	>50K	<=50K
White	0.910585	0.844685
Black	0.047224	0.107322
Amer-Indian-Eskimo	0.003553	0.010894
Asian-Pac-Islander	0.034937	0.027726
Other	0.003701	0.009373

Testing and Verifying

```
#Testing ---------------------------------
predicts = []
start_test = time.time()
for i in range(len(X_test)):
    results = []
    for label in labels:
        pro = 1.0
        for col in cols:
            if ((X_test[col].dtypes) == "object"):
                temp = X_test.iloc[i][col]
                exec("global table;table=df_{0}".format(col.replace('-', "")))
                if(temp in table.index):
                    exec("global value;value = df_{0}.loc[temp, label]".format(col.replace('-',"")))
                    #print(pro, value, col)
                    pro = pro * value

            elif ((X_test[col].dtypes) == "int64"):
                temp = X_test.iloc[i][col]
                exec("global mean_value;mean_value = df_{0}.loc['mean', label]".format(col.replace('-', "")))
                exec("global stdev_value;stdev_value = df_{0}.loc['stdev', label]".format(col.replace('-', "")))
                #print(pro ,gaussianProbability(temp,mean_value,stdev_value), col)
                pro = pro * gaussianProbability(temp,mean_value,stdev_value)
                #print(gaussianProbability(temp,mean,stdev),col)

        #print(pro, df_global.loc[label, 'Pr'], "global")
        pro = pro * df_global.loc[label, 'Pr']
        results.append(pro)
    predicts.append(labels[results.index(max(results))])
```

Input the testing data and call the probability data frame to calculate the probability result. The “pro” is multiplied by each probability result from every attribute input and the label probability.
Lastly compared the final probability result between two labels (<=50k and >50k), and the prediction will be the one with greater probability.

```
print("testing time: ", time.time() - start_test,'\n')

print("Accuracy score: {}".format(accuracy_score(Y_test['label'], predicts)))
print("Precision score: {}".format(precision_score(Y_test['label'], predicts,pos_label=pos_label)))
print("Recall score: {}".format(recall_score(Y_test['label'], predicts,pos_label=pos_label)))
acc = accuracy_score(Y_test['label'], predicts)
pre = precision_score(Y_test['label'], predicts,pos_label=pos_label)
rec = recall_score(Y_test['label'], predicts,pos_label=pos_label)

return acc,pre,rec
```

Finally, print out the time taken, accuracy, precision, and recall for the testing data.
The positive (1/true) is set to be >50k.

Predicting

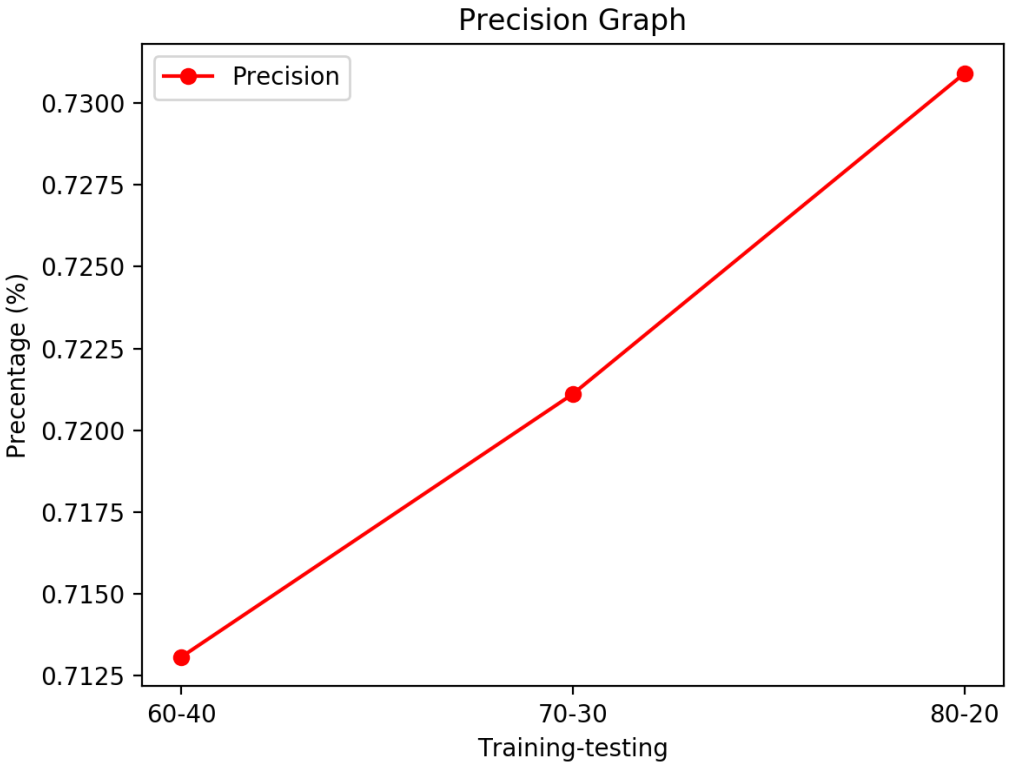
```
#input the data which wants to be predicted
mydata_predict = pd.read_csv(r"/Users/lichen/Documents/adult.predict.csv")
    #Predicting------------------------------
    result = []
    for label in labels:
        pro = 1.0
        for col in cols:
            if ((mydata_predict[col].dtypes) == "object"):
                temp = mydata_predict.iloc[0][col]
                exec("global table;table=df_{0}".format(col.replace('-', "")))
                if (temp in table.index):
                    exec("global value;value = df_{0}.loc[temp, label]".format(col.replace('-', "")))
                    # print(pro, value, col)
                    pro = pro * value

            elif ((mydata_predict[col].dtypes) == "int64"):
                temp = mydata_predict.iloc[0][col]
                exec("global mean_value;mean_value = df_{0}.loc['mean', label]".format(col.replace('-', "")))
                exec("global stdev_value;stdev_value = df_{0}.loc['stdev', label]".format(col.replace('-', "")))
                # print(pro ,gaussianProbability(temp,mean_value,stdev_value), col)
                pro = pro * gaussianProbability(temp, mean_value, stdev_value)
                # print(gaussianProbability(temp,mean,stdev),col)

        # print(pro, df_global.loc[label, 'Pr'], "global")
        pro = pro * df_global.loc[label, 'Pr']
        result.append(pro)
    predict = result.index(max(result))
    if (predict==0):
        print('Predict: ','<=50k')
    else:
        print('Predict: ','>50k')
```

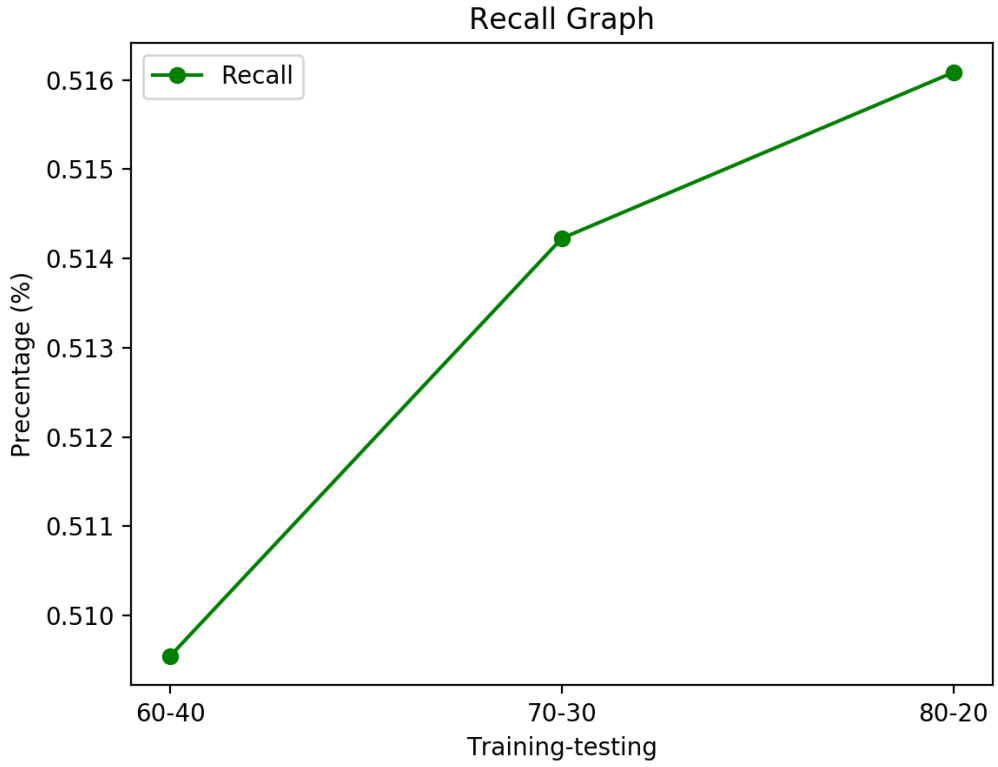
Input the desired data set in the csv file and the prediction will output the result as ‘<=50k’ or ‘>50K’.

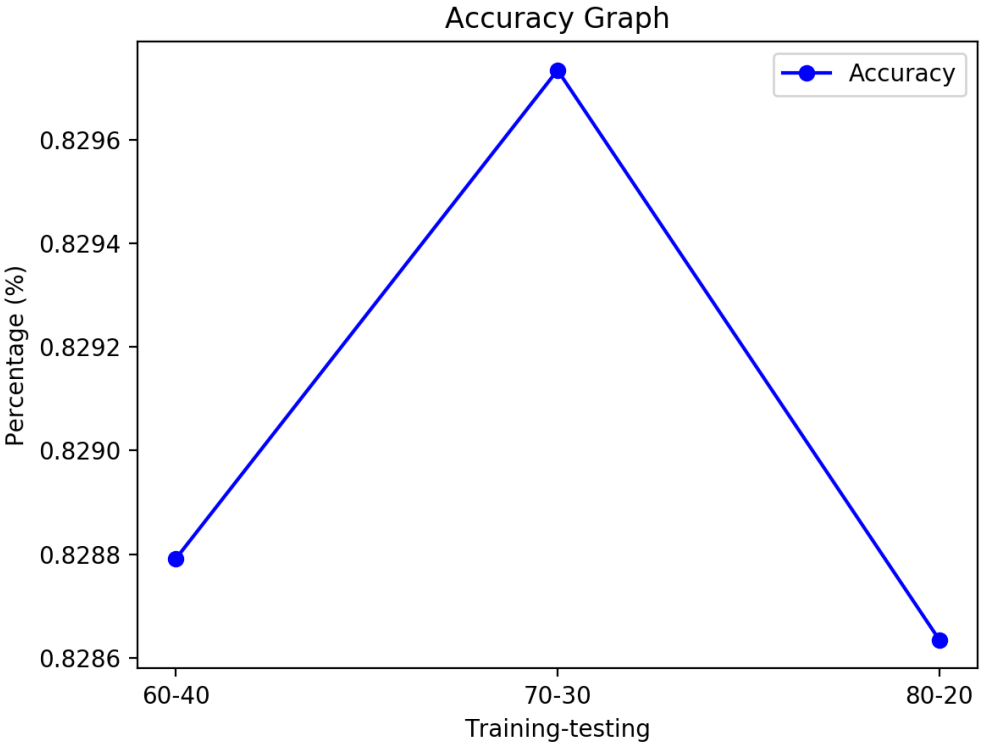
Final Result

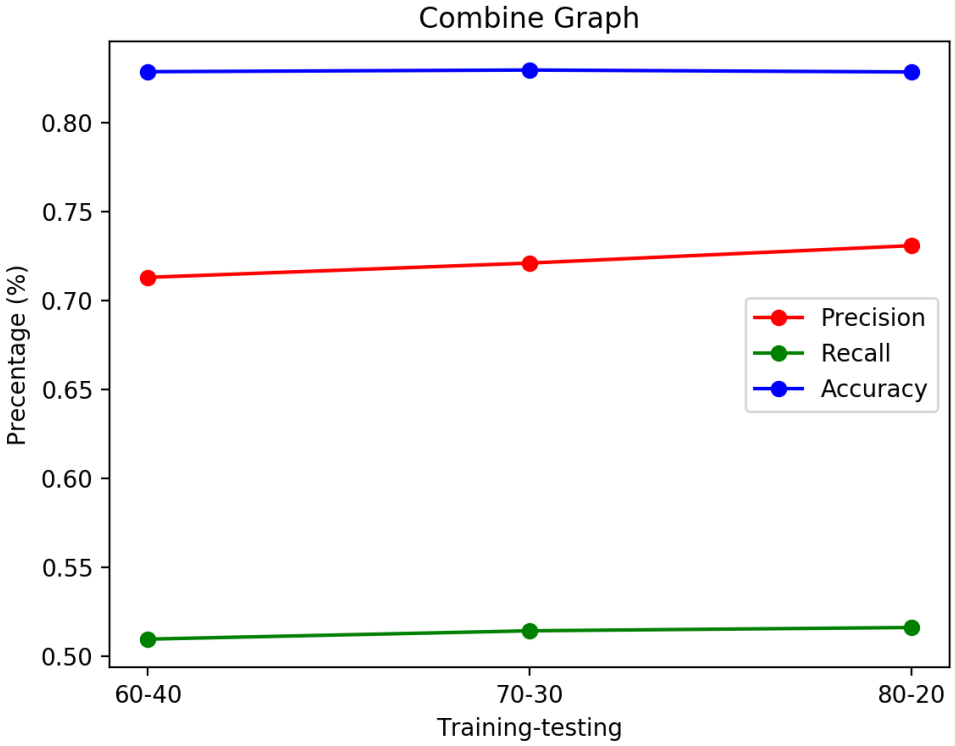
```
60-40 training-testing
training time:  2.707607030868530 3
testing time:  101.41898012161255

Accuracy score: 0.828790977942396
Precision score: 0.713073538654934 1
Recall score: 0.509544127554457 7


70-30 training-testing
training time:  3.335626840591430 7
testing time:  76.53147387504578

Accuracy score: 0.829733913171666 6
Precision score: 0.721113881961762 3
Recall score: 0.514226437462951 9


80-20 training-testing
training time:  3.063457012176513 7
testing time:  51.98260664939880 4

Accuracy score: 0.828634604754007 7
Precision score: 0.730911330049261 1
Recall score: 0.516086956521739 1
```

Result Graph






Accuracy Graph
Accuracy
0.8296
0.8294
0.8292
0.8290
0.8288
0.8286
Percentage (%)
60-40
70-30
80-20
Training-testing


Combine Graph
Precision
Recall
Accuracy
0.80
0.75
0.70
0.65
0.60
0.55
0.50
Precentage (%)
60-40
70-30
80-20
Training-testing