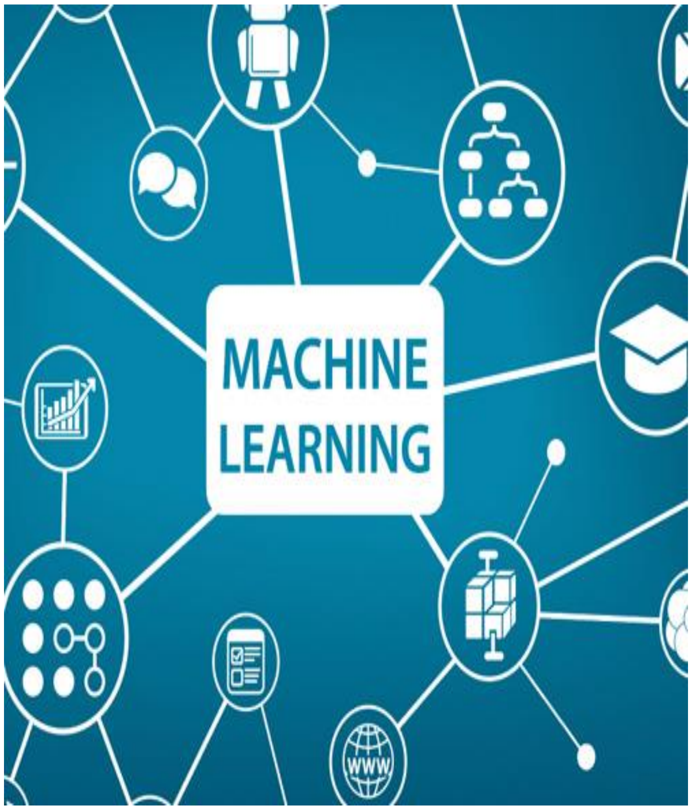


NAIVE BAYES CLASSIFIER

Assignment 1

MACHINE LEARNING FOR DATA SCIENCE

Submitted by:

Anosha Khan

Mamoona Akbar

Assignment 1

Implementation of Naive Bayes Classifier

Experiment Description:

The objective of this work is to develop and evaluate a Naïve Bayes Classifier to forecast income (>50K or <=50K) using a reduced version of the census income dataset provided by the Course Instructor.

The dataset has four attributes i.e. Work class, Education, Relationship and Sex along with their corresponding income labels (<=50K or >50K)

The main steps of the experiment are:

1. Data preprocessing
2. Development of Naïve Bayes Classifier
3. Training Model and testing the results

In data preprocessing, we removed all the samples with missing values, education attribute is processed to reduce the number of distinct values to five categories: Masters, Bachelors, Doctorate, Primary, and HighSchool. Samples with relationship values other than Wife, Husband, or Unmarried are removed.

In second phase, we have developed a Naïve Bayes Classifier by utilizing the assumption of conditional independence between features given the class label.

In third phase, we have given last 100 test samples to our Naïve Bayes Classifier and determine the difference between the actual values and predicted values to check the accuracy.

Methodology:

Tools:

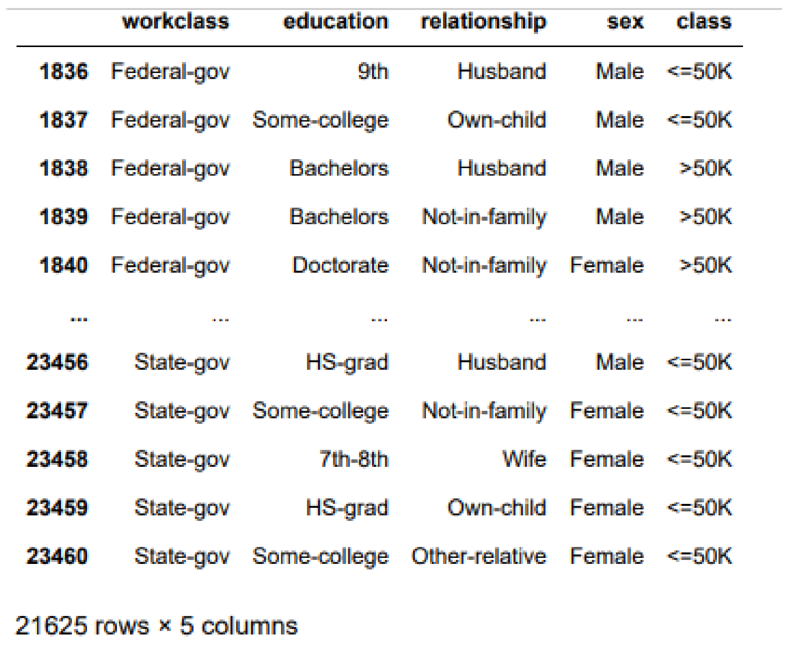
1. Jupyter Notebook
2. Python programming language
3. Pandas library for data manipulation and analysis

Techniques:

1. **Data Preprocessing:**

   Initially total samples were 23461 in the dataset. We have performed the following task in data preprocessing.

   a) **Removing Missing values**
   The provided dataset has 1836 missing values (?) in **work class** attribute so we have discarded all those samples with a “?” from the dataset. We got the resultant dataset of 21625 samples.



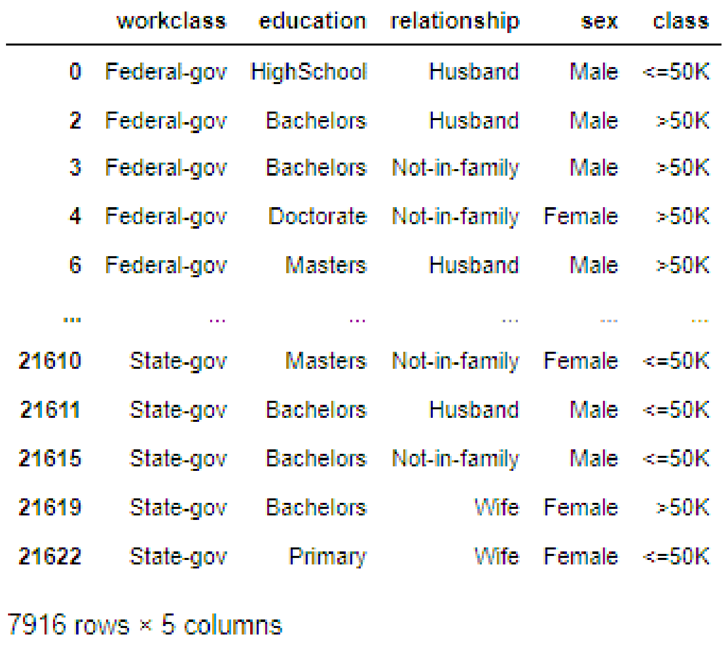
	workclass	education	relationship	sex	class
1836	Federal-gov	9th	Husband	Male	<=50K
1837	Federal-gov	Some-college	Own-child	Male	<=50K
1838	Federal-gov	Bachelors	Husband	Male	>50K
1839	Federal-gov	Bachelors	Not-in-family	Male	>50K
1840	Federal-gov	Doctorate	Not-in-family	Female	>50K
...	...	...	...	...	...
23456	State-gov	HS-grad	Husband	Male	<=50K
23457	State-gov	Some-college	Not-in-family	Female	<=50K
23458	State-gov	7th-8th	Wife	Female	<=50K
23459	State-gov	HS-grad	Own-child	Female	<=50K
23460	State-gov	Some-college	Other-relative	Female	<=50K

21625 rows × 5 columns

Figure 1: Dataset after Removal of Missing Values

b) Replacement of values in education attribute
We replaced 1st-4th, 5th-6th, and 7th-8th by **Primary** and 9th, 10th, 11th, and 12th by **HighSchool.**

c) Reduction of samples on the basis of Education attribute
We removed all the samples having any other values except Primary, HighSchool, Masters, Bachelors and Doctorate in **education** attribute. 13709 samples are dropped and we got 7916 samples in the resultant dataset.



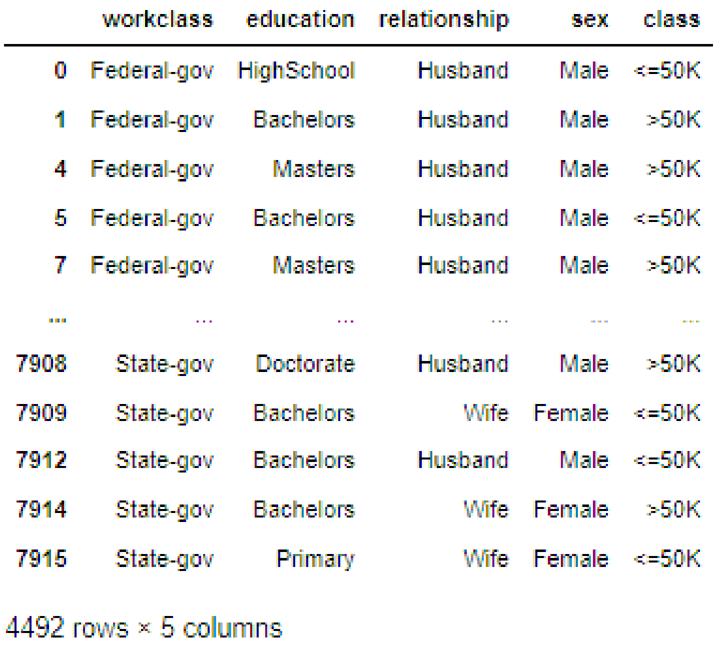
	workclass	education	relationship	sex	class
0	Federal-gov	HighSchool	Husband	Male	<=50K
2	Federal-gov	Bachelors	Husband	Male	>50K
3	Federal-gov	Bachelors	Not-in-family	Male	>50K
4	Federal-gov	Doctorate	Not-in-family	Female	>50K
6	Federal-gov	Masters	Husband	Male	>50K
...	...	...	...	...	...
21610	State-gov	Masters	Not-in-family	Female	<=50K
21611	State-gov	Bachelors	Husband	Male	<=50K
21615	State-gov	Bachelors	Not-in-family	Male	<=50K
21619	State-gov	Bachelors	Wife	Female	>50K
21622	State-gov	Primary	Wife	Female	<=50K

7916 rows × 5 columns

Figure 2: Dataset after reduction on the basis of education attribute

d) Reduction of samples on the basis of Relationship attribute

We removed all the samples having any other values except Husband, Unmarried, and Wife in **Relationship** attribute. 3424 samples are dropped and we got **4492** samples in the resultant dataset.



	workclass	education	relationship	sex	class
0	Federal-gov	HighSchool	Husband	Male	<=50K
1	Federal-gov	Bachelors	Husband	Male	>50K
4	Federal-gov	Masters	Husband	Male	>50K
5	Federal-gov	Bachelors	Husband	Male	<=50K
7	Federal-gov	Masters	Husband	Male	>50K
...	...	...	...	...	...
7908	State-gov	Doctorate	Husband	Male	>50K
7909	State-gov	Bachelors	Wife	Female	<=50K
7912	State-gov	Bachelors	Husband	Male	<=50K
7914	State-gov	Bachelors	Wife	Female	>50K
7915	State-gov	Primary	Wife	Female	<=50K

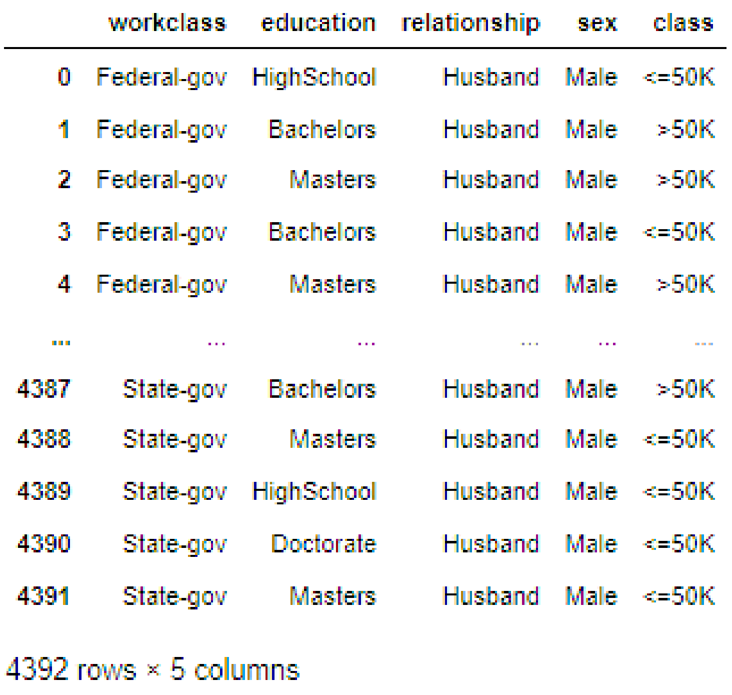
4492 rows × 5 columns

Figure 3: Dataset after reduction on the basis of relationship attribute

2. Naïve Bayes Classifier Training/Testing

a) Train/Test Split

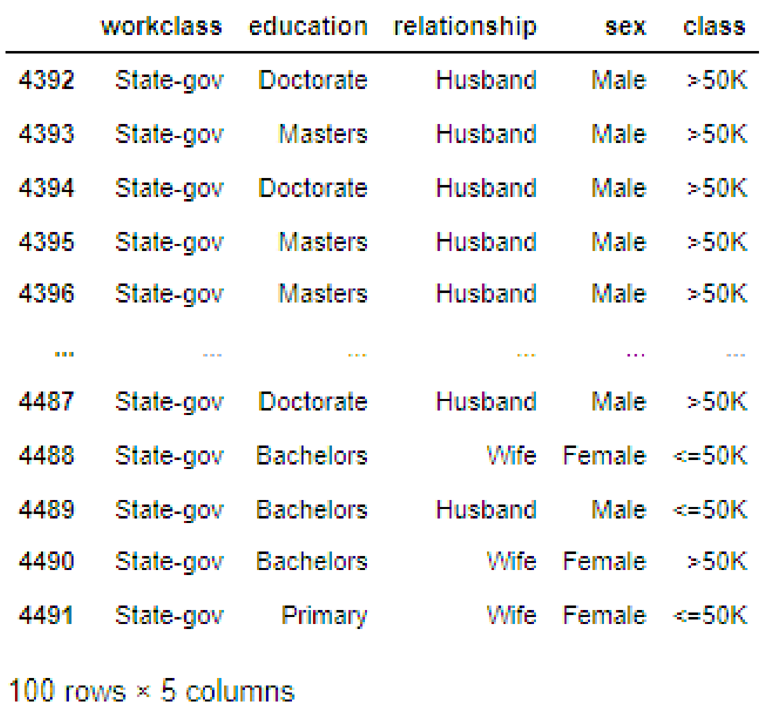
As per the instructions, we have divided out dataset into two parts **4392 training samples** and **100 testing sample**.



	workclass	education	relationship	sex	class
0	Federal-gov	HighSchool	Husband	Male	<=50K
1	Federal-gov	Bachelors	Husband	Male	>50K
2	Federal-gov	Masters	Husband	Male	>50K
3	Federal-gov	Bachelors	Husband	Male	<=50K
4	Federal-gov	Masters	Husband	Male	>50K
...	...	...	...	...	...
4387	State-gov	Bachelors	Husband	Male	>50K
4388	State-gov	Masters	Husband	Male	<=50K
4389	State-gov	HighSchool	Husband	Male	<=50K
4390	State-gov	Doctorate	Husband	Male	<=50K
4391	State-gov	Masters	Husband	Male	<=50K

4392 rows × 5 columns

Figure 4: Training Data



	workclass	education	relationship	sex	class
4392	State-gov	Doctorate	Husband	Male	>50K
4393	State-gov	Masters	Husband	Male	>50K
4394	State-gov	Doctorate	Husband	Male	>50K
4395	State-gov	Masters	Husband	Male	>50K
4396	State-gov	Masters	Husband	Male	>50K
...	...	...	...	...	...
4487	State-gov	Doctorate	Husband	Male	>50K
4488	State-gov	Bachelors	Wife	Female	<=50K
4489	State-gov	Bachelors	Husband	Male	<=50K
4490	State-gov	Bachelors	Wife	Female	>50K
4491	State-gov	Primary	Wife	Female	<=50K

100 rows × 5 columns

Figure 5: Testing Data

b) Development and Evaluation Details of Naïve Bayes Classifier

We developed a Class of Naïve based classifier with a number of functions as follows:

- **count_label** function for counting class labels from training data i.e. count of samples having >50k and having <=50k as labels in training data.

- **count_event_given_label** function which returns the count of value of one attribute given the class label

- **train_and_predict** function that picks samples from test data one by one and calculates the probabilities of features given one class label, second class label and so on separately. After calculating probabilities of a sample for all the class labels it assigns the label with high probability to that sample. This functions returns a list of all the predicted labels.

- **accuracy** function is developed which compares the predicted and actual labels and gives the accuracy of our model. The formula of accuracy is as follows:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\textit{Total Number of Predictions}} \text{ x } 100\%$$

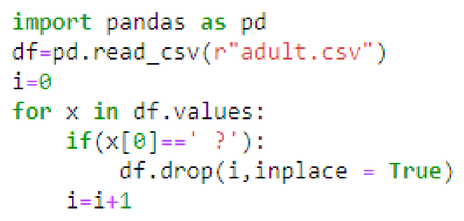
- **applySmoothing** function is also developed for avoiding zero probabilities. We developed a separate function applySmoothing that is based on the following formula:
  If P(Event)=0 then,

$$\text{P(Event)}= \frac{1}{count\ of\ that\ label+1+\ \alpha}$$ where α is number of distinct values in that attribute

Experimental setup

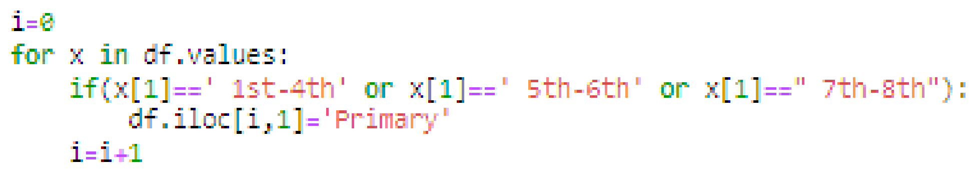
We used python programming language to ease our process of data preprocessing, Model Development, Training and Testing. We have performed the following task in data preprocessing.

1. **Removing Missing values**
   The provided dataset has 1836 missing values (?) in **work class** attribute so we have discarded all those samples with a "?" from the dataset by the following lines of code.

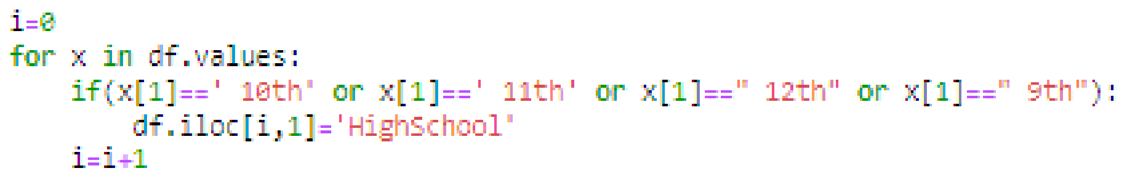


```
import pandas as pd
df=pd.read_csv(r"adult.csv")
i=0
for x in df.values:
    if(x[0]==' ?'):
        df.drop(i,inplace = True)
    i=i+1
```

2. **Replacement of values in education attribute**
   We replaced 1st-4th, 5th-6th, and 7th-8th by **Primary** and 9th, 10th, 11th, and 12th by **HighSchool** using the following piece of code.

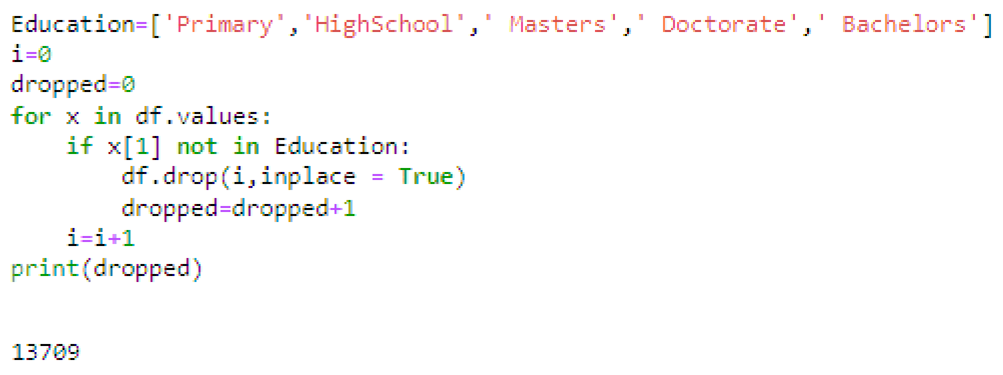


```
i=0
for x in df.values:
    if(x[1]==' 1st-4th' or x[1]==' 5th-6th' or x[1]==" 7th-8th"):
        df.iloc[i,1]='Primary'
    i=i+1
```



```
i=0
for x in df.values:
    if(x[1]==' 10th' or x[1]==' 11th' or x[1]==" 12th" or x[1]==" 9th"):
        df.iloc[i,1]='HighSchool'
    i=i+1
```

3. **Reduction of samples on the basis of Education attribute**
   We removed all the samples having any other values except Primary, HighSchool, Masters, Bachelors and Doctorate in **education** attribute by using the following piece of code.



```
Education=['Primary','HighSchool',' Masters',' Doctorate',' Bachelors']
i=0
dropped=0
for x in df.values:
    if x[1] not in Education:
        df.drop(i,inplace = True)
        dropped=dropped+1
    i=i+1
print(dropped)
```

```
13709
```

   13709 samples are dropped and we got 7916 samples in the resultant dataset.

4. **Reduction of samples on the basis of Relationship attribute**
   We removed all the samples having any other values except Husband, Unmarried, and Wife in **Relationship** attribute by using the following piece of code.

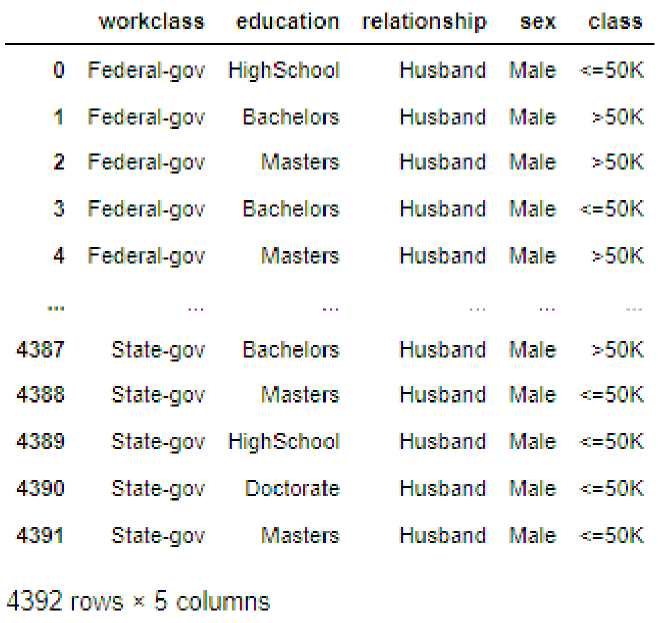
```
relationship=[' Husband',' Unmarried',' Wife']
```

```
i=0
drop=0
for x in df.values:
    if x[2] not in relationship:
        df.drop(i,inplace = True)
        drop=drop+1
    i=i+1
print(drop)

3424
```

   3424 samples are dropped and we got 4492 samples in the resultant dataset.

5. **Train/Test split**
   As per the instructions, we have divided out dataset into two parts 4392 training samples and 100 testing sample by the following lines of code:

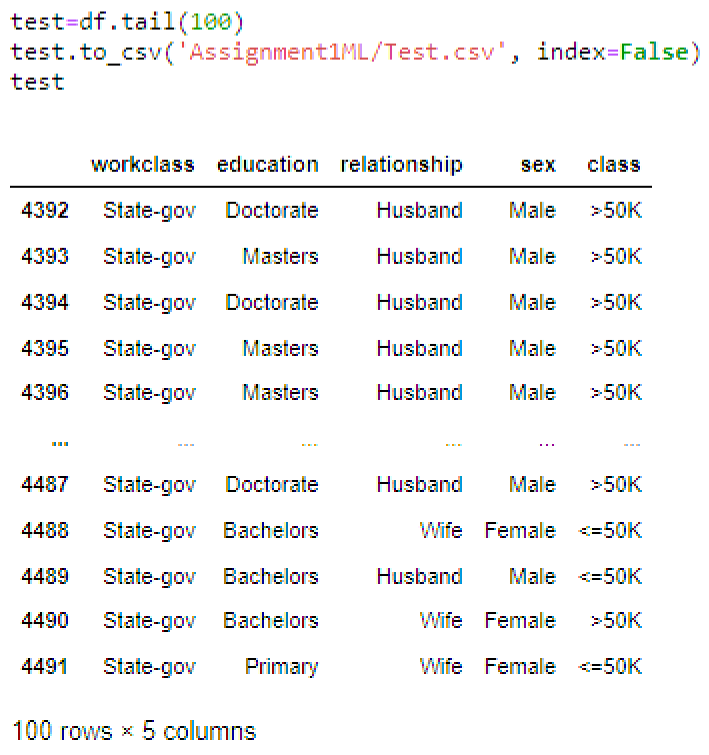
```
train=df.head(4392)
train.to_csv('Assignment1ML/Train.csv', index=False)
train
```



	workclass	education	relationship	sex	class
0	Federal-gov	HighSchool	Husband	Male	<=50K
1	Federal-gov	Bachelors	Husband	Male	>50K
2	Federal-gov	Masters	Husband	Male	>50K
3	Federal-gov	Bachelors	Husband	Male	<=50K
4	Federal-gov	Masters	Husband	Male	>50K
...	...	...	...	...	...
4387	State-gov	Bachelors	Husband	Male	>50K
4388	State-gov	Masters	Husband	Male	<=50K
4389	State-gov	HighSchool	Husband	Male	<=50K
4390	State-gov	Doctorate	Husband	Male	<=50K
4391	State-gov	Masters	Husband	Male	<=50K

4392 rows × 5 columns

Figure 6: Training data



```
test=df.tail(100)
test.to_csv('Assignment1ML/Test.csv', index=False)
test
```

	workclass	education	relationship	sex	class
4392	State-gov	Doctorate	Husband	Male	>50K
4393	State-gov	Masters	Husband	Male	>50K
4394	State-gov	Doctorate	Husband	Male	>50K
4395	State-gov	Masters	Husband	Male	>50K
4396	State-gov	Masters	Husband	Male	>50K
...	...	...	...	...	...
4487	State-gov	Doctorate	Husband	Male	>50K
4488	State-gov	Bachelors	Wife	Female	<=50K
4489	State-gov	Bachelors	Husband	Male	<=50K
4490	State-gov	Bachelors	Wife	Female	>50K
4491	State-gov	Primary	Wife	Female	<=50K

100 rows × 5 columns

Figure 7: Testing data

6. Development and Evaluation of Naïve Bayes Classifier

We developed a function for counting class labels from training data i.e. count of samples having >50k and having <=50k in training data.

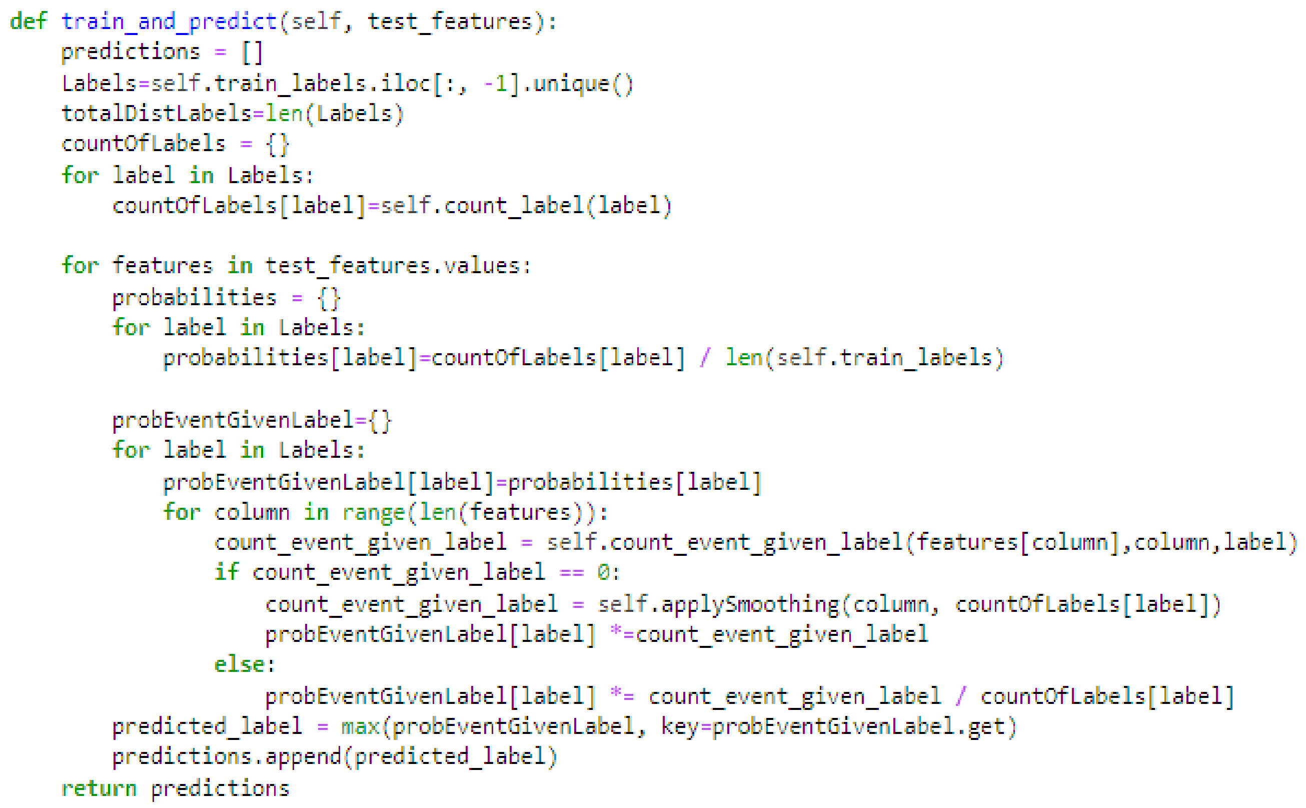
```
def count_label(self, label):
    totallabel=0;
    for row in self.train_labels.values:
        if row == label:
            totallabel=totallabel+1
    return totallabel
```

Then we have developed a function which returns the count of value of one attribute given the class label as follows:

```
def count_event_given_label(self, event, column, label):
    totalevent=0;
    for i in range(len(self.train_features)):
        if self.train_features.values[i][column]==event and self.train_labels.values[i] == label:
            totalevent=totalevent+1
    return totalevent
```

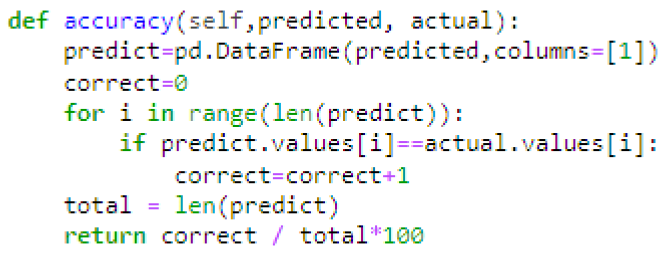
Then we have developed a function that picks samples from test data one by one and calculates the probabilities of features given one class label and then second class label separately. After calculating probabilities of a sample for both class labels it assigns the label with high probability to that sample.

The code is as follows:



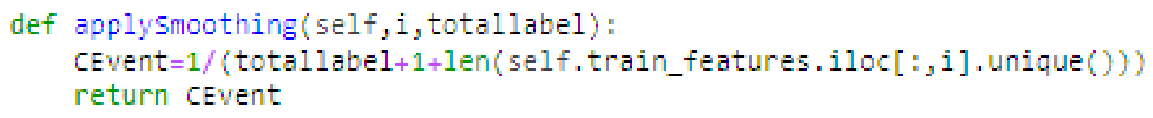
```
def train_and_predict(self, test_features):
    predictions = []
    Labels=self.train_labels.iloc[:, -1].unique()
    totalDistLabels=len(Labels)
    countOfLabels = {}
    for label in Labels:
        countOfLabels[label]=self.count_label(label)

    for features in test_features.values:
        probabilities = {}
        for label in Labels:
            probabilities[label]=countOfLabels[label] / len(self.train_labels)

        probEventGivenLabel={}
        for label in Labels:
            probEventGivenLabel[label]=probabilities[label]
            for column in range(len(features)):
                count_event_given_label = self.count_event_given_label(features[column],column,label)
                if count_event_given_label == 0:
                    count_event_given_label = self.applySmoothing(column, countOfLabels[label])
                    probEventGivenLabel[label] *=count_event_given_label
                else:
                    probEventGivenLabel[label] *= count_event_given_label / countOfLabels[label]
        predicted_label = max(probEventGivenLabel, key=probEventGivenLabel.get)
        predictions.append(predicted_label)
    return predictions
```

Then we have developed the accuracy function which compares the predicted and actual labels and gives the accuracy of our model. The following is the code for that:



```
def accuracy(self,predicted, actual):
    predict=pd.DataFrame(predicted,columns=[1])
    correct=0
    for i in range(len(predict)):
        if predict.values[i]==actual.values[i]:
            correct=correct+1
    total = len(predict)
    return correct / total*100
```

Then we have developed the smoothing function for avoiding zero probabilities. The following is the code for that:



```
def applySmoothing(self,i,totallabel):
    CEvent=1/(totallabel+1+len(self.train_features.iloc[:,i].unique()))
    return CEvent
```

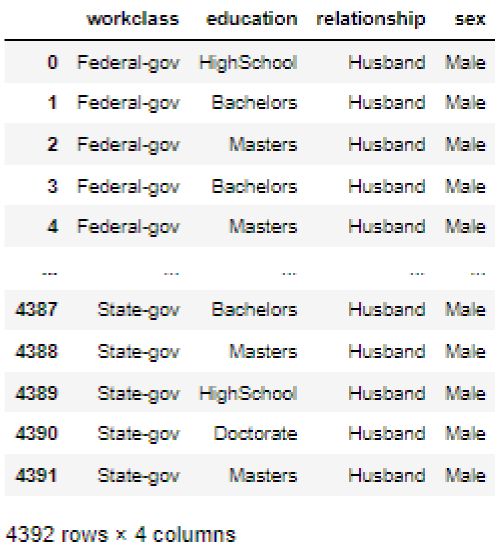
Complete Code of Naïve Bayes Classifier Class is given below:

```
class NaiveBayesClassifier:
  def __init__(self, train_features, train_labels):
    self.train_features = train_features
    self.train_labels = train_labels
  def count_label(self, label):
    totallabel=0;
    for row in self.train_labels.values:
      if row == label:
        totallabel=totallabel+1
    return totallabel
  def count_event_given_label(self, event, column, label):
    totalevent=0;
    for i in range(len(self.train_features)):
      if self.train_features.values[i][column]==event and self.train_labels.values[i] == label:
        totalevent=totalevent+1
    return totalevent
  def applySmoothing(self,i,totallabel):
    CEvent=1/(totallabel+1+len(self.train_features.iloc[:,i].unique()))
    return CEvent
  def train_and_predict(self, test_features):
    predictions = []
    Labels=self.train_labels.iloc[:, -1].unique()
    totalDistLabels=len(Labels)
    countOfLabels = {}
    for label in Labels:
      countOfLabels[label]=self.count_label(label)
    for features in test_features.values:
      probabilities = {}
      for label in Labels:
        probabilities[label]=countOfLabels[label] / len(self.train_labels)
       probEventGivenLabel={}
      for label in Labels:
        probEventGivenLabel[label]=probabilities[label]
        for column in range(len(features)):
              count_event_given_label = self.count_event_given_label(features[column],column,label)
          if count_event_given_label == 0:
            count_event_given_label = self.applySmoothing(column, countOfLabels[label])
            probEventGivenLabel[label] *=count_event_given_label
          else:
            probEventGivenLabel[label] *= count_event_given_label / countOfLabels[label]
      predicted_label = max(probEventGivenLabel, key=probEventGivenLabel.get)
      predictions.append(predicted_label)
    return predictions
  def accuracy(self,predicted, actual):
    predict=pd.DataFrame(predicted,columns=[1])
    correct=0
    for i in range(len(predict)):
      if predict.values[i]==actual.values[i]:
        correct=correct+1
    total = len(predict)
    return correct / total*100
```

We have divided our dataset into two parts **4392 training samples** and **100 testing sample**. We have extracted Training features, Training labels, Test features and Test Labels using the following code:

```
trainfeatures=train.drop(train.columns[-1], axis=1)
trainlabels=train.iloc[:, 4:]
testfeatures=test.drop(train.columns[-1], axis=1)
testlabels=test.iloc[:, 4:]
```

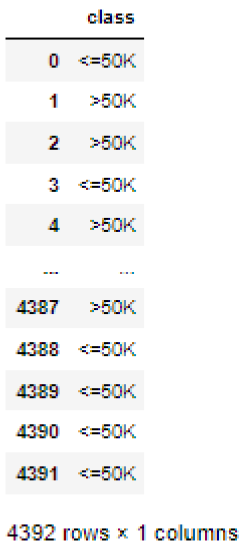
We got the data frames as follows:



	workclass	education	relationship	sex
0	Federal-gov	HighSchool	Husband	Male
1	Federal-gov	Bachelors	Husband	Male
2	Federal-gov	Masters	Husband	Male
3	Federal-gov	Bachelors	Husband	Male
4	Federal-gov	Masters	Husband	Male
...	...	...	...	...
4387	State-gov	Bachelors	Husband	Male
4388	State-gov	Masters	Husband	Male
4389	State-gov	HighSchool	Husband	Male
4390	State-gov	Doctorate	Husband	Male
4391	State-gov	Masters	Husband	Male

4392 rows × 4 columns

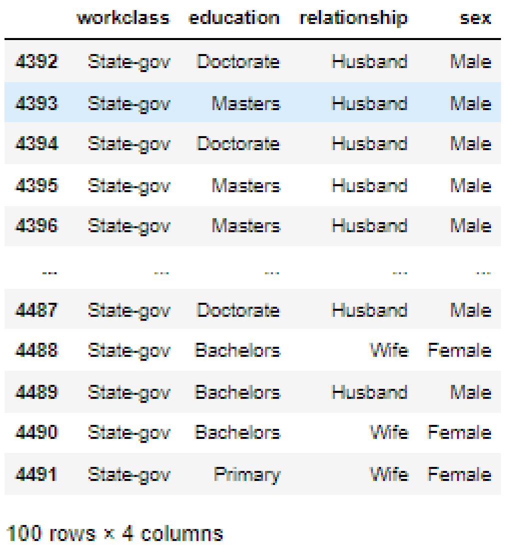
Figure 8: Training Features



	class
0	<=50K
1	>50K
2	>50K
3	<=50K
4	>50K
...	...
4387	>50K
4388	<=50K
4389	<=50K
4390	<=50K
4391	<=50K

4392 rows × 1 columns

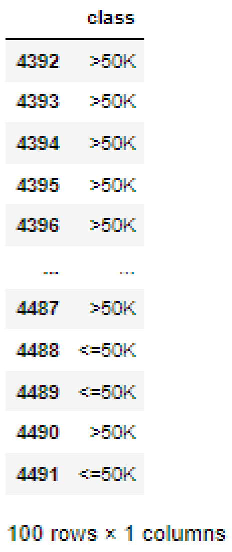
Figure 9: Training Labels



	workclass	education	relationship	sex
4392	State-gov	Doctorate	Husband	Male
4393	State-gov	Masters	Husband	Male
4394	State-gov	Doctorate	Husband	Male
4395	State-gov	Masters	Husband	Male
4396	State-gov	Masters	Husband	Male
...	...	...	...	...
4487	State-gov	Doctorate	Husband	Male
4488	State-gov	Bachelors	Wife	Female
4489	State-gov	Bachelors	Husband	Male
4490	State-gov	Bachelors	Wife	Female
4491	State-gov	Primary	Wife	Female

100 rows × 4 columns

Figure 11: Testing Features



	class
4392	>50K
4393	>50K
4394	>50K
4395	>50K
4396	>50K
...	...
4487	>50K
4488	<=50K
4489	<=50K
4490	>50K
4491	<=50K

100 rows × 1 columns

Figure 10: Testing Labels

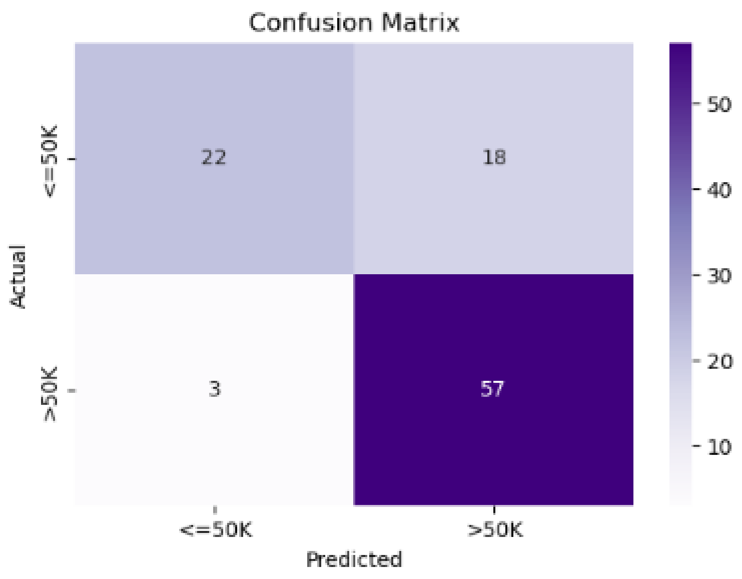
Model is Train and evaluated using the following code:

```
classifier= NaiveBayesClassifier(trainfeatures,trainlabels)
predictions=classifier.train_and_predict(testfeatures)
accuracy=classifier.accuracy(predictions, testlabels)
print('Accuracy '+str(accuracy)+'%')
```

Evaluation method is calculating the accuracy by finding the differences between predicted labels and actual labels.

Results and Discussion

In this study, we developed a Naive Bayes classifier to predict income class (>50K or <=50K) based on four attributes: classwork, education, relationship, and sex. After training the classifier on the provided dataset, we achieved an accuracy of 79% on the test set. The following is the confusion matrix of the results.



As it clear from the above confusion matrix, there are 3 samples belongs to the label **>50K** but Naïve Bayes Classifier predicted their labels as **<=50K** and there are 18 samples belongs to the label **<=50K** but our Classifier predicted their labels as **>50K**.

The achieved accuracy of 79% indicates that the Naive Bayes classifier performed not very well in predicting income class based on the selected attributes. While this accuracy is satisfactory, there may still be room for improvement.

One possible reason for the achieved accuracy could be the simplicity of the dataset and the Naive Bayes classifier's assumption of feature independence. Naive Bayes works well with categorical features and is relatively simple to implement, making it suitable for this task.

However, there are limitations to the Naive Bayes approach. It assumes that features are conditionally independent given the class label, which may not always hold true in real-world datasets. Additionally, the classifier may struggle with datasets containing continuous or correlated features. Also we are dropping so much samples that could be a main reason for poor accuracy by adding more samples can give better results.

Further analysis could involve exploring additional features or experimenting with different classification algorithms to improve accuracy. Feature engineering techniques could also be employed to extract more meaningful information from the existing attributes.

Conclusion

In conclusion, while the Naive Bayes classifier achieved a respectable accuracy of 79%, there is potential for further refinement and exploration to improve predictive performance. This could involve addressing the limitations of the Naive Bayes approach and considering alternative algorithms or feature engineering strategies to enhance the model's predictive capabilities.