Data Science

Algorithms

Task II: Implement and evaluate a Naive Bayes classifier for a given dataset.

This question is divided into three sub-questions. Please follow the instructions and answer all sub-questions accordingly.

1. Implement the Naive Bayes classifier.

Given a dataset with categorical features and a categorical target variable, implement a Naive Bayes classifier from scratch. Your implementation should include the following functions:

1.1. Function to calculate the prior probabilities for each class in the target variable.

1.2. Function to calculate the likelihood for each feature, given a specific class.

1.3. Function to predict the class of a new instance based on the calculated prior probabilities and likelihoods.

You may use any programming language of your choice. Please provide clear and concise code, along with comments explaining your implementation.

Naive Bayes classification:

A group of extremely fast and simple classification techniques known as naive Bayes models is frequently appropriate for very high-dimensional datasets. As a quick-and-dirty baseline for a classification test, they prove to be highly valuable due to their rapidity and limited number of adjustable factors.

Bayesian classification techniques are the foundation of naive Bayes classifiers. These rely on the relationship between conditional probabilities of statistical quantities, as described by the equation known as Bayes's theorem.

Probability of a label given some observed features can be given as $P(L \mid \text{features})$

$$P(L \mid \text{features}) = \frac{P(\text{features} \mid L)P(L)}{P(\text{features})}$$

One method to choose between two labels, let's say L1 and L2, is to calculate the ratio of the posterior probability for each label:

$$\frac{P(L_1 \mid \text{features})}{P(L_2 \mid \text{features})} = \frac{P(\text{features} \mid L_1)}{P(\text{features} \mid L_2)} \frac{P(L_1)}{P(L_2)}$$

A generative model can be used to calculate P(features | Li) for each label since it describes the hypothetical random process that produces the data.

The key to training such a Bayesian classifier is to specify this generative model for every label.

It is possible to find a rough approximation of the generative model for each class and then proceed with the Bayesian classification, assuming relatively naive assumptions about the generative model for each label.

Bayes Theorem:

Equation for the Posterior Probability:

Posterior Probability = (Likelihood * Class prior probability) / Predictor prior probability

P(c|x) = (P(x|c) * p(c)) / P(x)

Function to calculate the prior probabilities for each class in the target variable.

```
In [12]: def _calc_class_prior(self):

             """ P(c) - Prior Class Probability """

             for outcome in np.unique(self.y_train):
                 outcome_count = sum(self.y_train == outcome)
                 self.class_priors[outcome] = outcome_count / self.train_size
```

Function to calculate the likelihood for each feature, given a specific class.

```
In [15]: def _calc_likelihoods(self):

             for feature in self.features:
                 for outcome in np.unique(self.y_train):
                     outcome_count = sum(self.y_train == outcome)
                     feat_likelihood =
                       self.X_train[feature][self.y_train[self.y_train == outcome].index.values.tolist()].value_counts().to_dict()

                     for feat_val, count in feat_likelihood.items():
                         self.likelihoods[feature][feat_val + '_' + outcome] = count/outcome_count


         def _calc_predictor_prior(self):
             for feature in self.features:
                 feat_vals = self.X_train[feature].value_counts().to_dict()

                 for feat_val, count in feat_vals.items():
                     self.pred_priors[feature][feat_val] = count/self.train_size
```

Function to predict the class of a new instance based on the calculated prior probabilities and likelihoods.

```python
In [16]: def predict(self, X):

    """ Calculates Posterior probability P(c|x) """

    results = []
    X = np.array(X)

    for query in X:
        probs_outcome = {}
        for outcome in np.unique(self.y_train):
            prior = self.class_priors[outcome]
            likelihood = 1
            evidence = 1

            for feat, feat_val in zip(self.features, query):
                likelihood *= self.likelihoods[feat][feat_val + '_' + outcome]
                evidence *= self.pred_priors[feat][feat_val]

            posterior = (likelihood * prior) / (evidence)

            probs_outcome[outcome] = posterior

        result = max(probs_outcome, key = lambda x: probs_outcome[x])
        results.append(result)

    return np.array(results)
```

2. Apply the Naive Bayes classifier to a dataset.

Choose a suitable dataset with categorical features and a categorical target variable. Apply your Naive Bayes classifier implementation from Sub-question 1 to this dataset.

2.1. Preprocess the dataset, including handling missing values, and split the data into training and testing sets.
Handling missing values = 1st or 2nd session.

2.2. Train your Naive Bayes classifier using the training set and predict the classes for the testing set.

2.3. Calculate the accuracy of your classifier on the testing set.

Please provide a brief explanation of the dataset you chose and any preprocessing steps you performed.

Below exercise's python file name: Algorithms_Assignment_Task2

Exercise 1:

Data source: mushrooms.txt from www.keggle.com

Loading data & viewing 1st five rows.

In [21]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math


df = pd.read_table("D:\mushrooms.txt")
df.head()
```

Out[21]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a

5 rows × 23 columns

In [22]:
```
df[df.columns[-1]]
```
Out[22]:
```
0       u
1       g
2       m
3       u
4       g
       ..
8119    l
8120    l
8121    l
8122    l
8123    l
Name: habitat, Length: 8124, dtype: object
```

Finding categorical variables.

In [28]:
```
# find categorical variables

categorical = [var for var in df.columns if df[var].dtype=='O']

print('There are {} categorical variables\n'.format(len(categorical)))

print('The categorical variables are :\n\n', categorical)
```
```
There are 23 categorical variables

The categorical variables are :

 ['class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor', 'gill-attachment', 'gill-spacing', 'gill-size', 'gi
ll-color', 'stalk-shape', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'stalk-color-above-ring',
'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number', 'ring-type', 'spore-print-color', 'population', 'habit
at']
```

Displaying categorical variable data.

In [29]:
```
# view the categorical variables

df[categorical].head()
```
Out[29]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a

5 rows × 23 columns

Checking any null values in the categorical variables’ column.

In [30]:
```
df[categorical].isnull().sum()
```

Out[30]:
```
class                     0
cap-shape                 0
cap-surface               0
cap-color                 0
bruises                   0
odor                      0
gill-attachment           0
gill-spacing              0
gill-size                 0
gill-color                0
stalk-shape               0
stalk-root                0
stalk-surface-above-ring  0
stalk-surface-below-ring  0
stalk-color-above-ring    0
stalk-color-below-ring    0
veil-type                 0
veil-color                0
ring-number               0
ring-type                 0
spore-print-color         0
population                0
habitat                   0
dtype: int64
```

Viewing the frequency distribution of categorical variables.

```
In [33]: import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization purposes
import seaborn as sns # for statistical data visualization
%matplotlib inline


# view frequency distribution of categorical variables

for var in categorical:

    print(df[var].value_counts()/float(len(df)))
```

```
e    0.011817
y    0.010586
o    0.007878
r    0.002954
Name: gill-color, dtype: float64
t    0.567208
e    0.432792
Name: stalk-shape, dtype: float64
b    0.464796
?    0.305268
e    0.137863
c    0.068439
r    0.023634
Name: stalk-root, dtype: float64
s    0.637125
k    0.291974
f    0.067947
y    0.002954
Name: stalk-surface-above-ring, dtype: float64
s    0.607582
```

```
In [43]: # view frequency distribution of categorical variables

         for var in categorical:

             print(df[var].value_counts()/float(len(df)))
```

```
e    0.011817
y    0.010586
o    0.007878
r    0.002954
Name: gill-color, dtype: float64
t    0.567208
e    0.432792
Name: stalk-shape, dtype: float64
b    0.464796
e    0.137863
c    0.068439
r    0.023634
Name: stalk-root, dtype: float64
s    0.637125
k    0.291974
f    0.067947
y    0.002954
Name: stalk-surface-above-ring, dtype: float64
s    0.607582
k    0.283604
```

Checking labels in the stalk_root variable.

Replacing unknown label with NaN.

```
In [50]: # check labels in stalk-root variable

         df['stalk-root'].unique()

Out[50]: array(['e', 'c', 'b', 'r', '?'], dtype=object)

In [51]: df['stalk-root'].replace('?', np.NaN, inplace=True)

In [52]: # check labels in stalk-root variable

         df['stalk-root'].unique()

Out[52]: array(['e', 'c', 'b', 'r', nan], dtype=object)
```

Checking labels in the cap_shape variable.

Replacing unknown label with NaN.

```
In [54]: # check labels in cap-shape variable

         df['cap-shape'].unique()

Out[54]: array(['x', 'b', 's', 'f', 'k', 'c'], dtype=object)

In [55]: # check labels in stalk-surface-above-ring variable

         df['stalk-surface-above-ring'].unique()

Out[55]: array(['s', 'f', 'k', 'y'], dtype=object)
```

Viewing the target variable column.

```
In [5]: # target column = class
        df[df.columns[0]]

Out[5]: 0       p
        1       e
        2       e
        3       p
        4       e
               ..
        8119    e
        8120    e
        8121    e
        8122    p
        8123    e
        Name: class, Length: 8124, dtype: object
```

Defining features and target column.

In [9]:
```python
# features and target column
X = df.drop([df.columns[0]], axis = 1)
y = df[df.columns[0]]

X,y
```

Out[9]:
```
(     cap-shape cap-surface cap-color bruises odor gill-attachment  \
 0            x           s         n       t    p               f
 1            x           s         y       t    a               f
 2            b           s         w       t    l               f
 3            x           y         w       t    p               f
 4            x           s         g       f    n               f
 ...        ...         ...       ...     ...  ...             ...
 8119         k           s         n       f    n               a
 8120         x           s         n       f    n               a
 8121         f           s         n       f    n               a
 8122         k           y         n       f    y               f
 8123         x           s         n       f    n               a

     gill-spacing gill-size gill-color stalk-shape  ...  \
 0               c         n          k           e  ...
 1               c         b          k           e  ...
 2               c         b          n           e  ...
 3               c         n          n           e  ...
 4               w         b          k           t  ...
 ...           ...       ...        ...         ...  ...
 8119            c         b          y           e  ...
 8120            c         b          y           e  ...
 8121            c         b          n           e  ...
 8122            c         n          b           t  ...
 8123            c         b          y           e  ...
```

```
      stalk-surface-below-ring stalk-color-above-ring stalk-color-below-ring  \
0                            s                      w                      w
1                            s                      w                      w
2                            s                      w                      w
3                            s                      w                      w
4                            s                      w                      w
...                        ...                    ...                    ...
8119                         s                      o                      o
8120                         s                      o                      o
8121                         s                      o                      o
8122                         k                      w                      w
8123                         s                      o                      o

      veil-type veil-color ring-number ring-type spore-print-color population  \
0             p          w           o         p                 k          s
1             p          w           o         p                 n          n
2             p          w           o         p                 n          n
3             p          w           o         p                 k          s
4             p          w           o         e                 n          a
...         ...        ...         ...       ...               ...        ...
8119          p          o           o         p                 b          c
8120          p          n           o         p                 b          v
8121          p          o           o         p                 b          c
8122          p          w           o         e                 w          v
8123          p          o           o         p                 o          c

      habitat
0           u
1           g
2           m
3           u
4           g
...       ...
8119        l
8120        l
8121        l
8122        l
8123        l
```

```
[8124 rows x 22 columns],
0       p
1       e
2       e
3       p
4       e
       ..
8119    e
8120    e
8121    e
8122    p
8123    e
Name: class, Length: 8124, dtype: object)
```

The complete model's python code.

```
In [16]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math


def accuracy_score(y_true, y_pred):

    """*score = (y_true - y_pred) / len(y_true) """

    return round(float(sum(y_pred == y_true))/float(len(y_true)) * 100 ,2)

def pre_processing(df):

    """ partioning data into features and target """

    X = df.drop([df.columns[0]], axis = 1)
    y = df[df.columns[0]]

    return X, y

def train_test_split(x, y, test_size = 0.25, random_state = None):

    """ partioning the data into train and test sets """

    x_test = x.sample(frac = test_size, random_state = random_state)
    y_test = y[x_test.index]

    x_train = x.drop(x_test.index)
    y_train = y.drop(y_test.index)

    return x_train, x_test, y_train, y_test
```

```
class  NaiveBayes:

    def __init__(self):

"""
    Attributes:
        likelihoods: Likelihood of each feature per class
        class_priors: Prior probabilities of classes
        pred_priors: Prior probabilities of features
        features: All features of dataset
"""
        self.features = list
        self.likelihoods = {}
        self.class_priors = {}
        self.pred_priors = {}

        self.X_train = np.array
        self.y_train = np.array
        self.train_size = int
        self.num_feats = int

    def fit(self, X, y):

        self.features = list(X.columns)
        self.X_train = X
        self.y_train = y
        self.train_size = X.shape[0]
        self.num_feats = X.shape[1]

        for feature in self.features:
            self.likelihoods[feature] = {}
            self.pred_priors[feature] = {}

            for feat_val in np.unique(self.X_train[feature]):
                self.pred_priors[feature].update({feat_val: 0})

                for outcome in np.unique(self.y_train):
                    self.likelihoods[feature].update({feat_val+'_'+outcome:0})
                    self.class_priors.update({outcome: 0})
```

```python
        self._calc_class_prior()
        self._calc_likelihoods()
        self._calc_predictor_prior()


    def _calc_class_prior(self):

    """ P(c) - Prior Class Probability """

        for outcome in np.unique(self.y_train):
            outcome_count = sum(self.y_train == outcome)
            self.class_priors[outcome] = outcome_count / self.train_size

    def _calc_likelihoods(self):

        """ P(x|c) - Likelihood """

        for feature in self.features:

            for outcome in np.unique(self.y_train):
                outcome_count = sum(self.y_train == outcome)
                feat_likelihood =
                self.X_train[feature][self.y_train[self.y_train == outcome].index.values.tolist()].value_counts().to_dict()

                for feat_val, count in feat_likelihood.items():
                    self.likelihoods[feature][feat_val + '_' + outcome] = count/outcome_count


    def _calc_predictor_prior(self):

        """ P(x) - Evidence """

        for feature in self.features:
            feat_vals = self.X_train[feature].value_counts().to_dict()

            for feat_val, count in feat_vals.items():
                self.pred_priors[feature][feat_val] = count/self.train_size
```

Function for the prediction of a new class.

```python
    def predict(self, X):

        """ Calculates Posterior probability P(c|x) """

        results = []
        X = np.array(X)

        for query in X:
            probs_outcome = {}
            for outcome in np.unique(self.y_train):
                prior = self.class_priors[outcome]
                likelihood = 1
                evidence = 1

                for feat, feat_val in zip(self.features, query):
                    likelihood *= self.likelihoods[feat][feat_val + '_' + outcome]
                    evidence *= self.pred_priors[feat][feat_val]

                posterior = (likelihood * prior) / (evidence)

                probs_outcome[outcome] = posterior

            result = max(probs_outcome, key = lambda x: probs_outcome[x])
            results.append(result)

        return np.array(results)
```

Printing the accuracy of the test set.

```
if __name__ == "__main__":


    df = pd.read_table("D:\mushrooms.txt")

    #Split fearures and target
    X,y  = pre_processing(df)

    #Split data into Training and Testing Sets : test size 20% train size 80%Predictor prior probability
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

    print(X_train, y_train)
    nb_clf = NaiveBayes()
    nb_clf.fit(X_train, y_train)
    print(X_train, y_train)

    #print("Train Accuracy: {}".format(accuracy_score(y_train, nb_clf.predict(X_train))))
    print("Test Accuracy: {}".format(accuracy_score(y_test, nb_clf.predict(X_test))))
```

```
     cap-shape cap-surface cap-color bruises odor gill-attachment  \
0            x           s         n       t    p               f
1            x           s         y       t    a               f
2            b           s         w       t    l               f
3            x           y         w       t    p               f
4            x           s         g       f    n               f
...        ...         ...       ...     ...  ...             ...
8117         k           s         e       f    y               f
8118         k           y         n       f    f               f
8119         k           s         n       f    n               a
8120         x           s         n       f    n               a
8121         f           s         n       f    n               a

     gill-spacing gill-size gill-color stalk-shape  ...  \
0               c         n          k           e  ...
1               c         b          k           e  ...
2               c         b          n           e  ...
3               c         n          n           e  ...
4               w         b          k           t  ...
...           ...       ...        ...         ...  ...
8117            c         n          b           t  ...
8118            c         n          b           t  ...
8119            c         b          y           e  ...
8120            c         b          y           e  ...
8121            c         b          n           e  ...

     stalk-surface-below-ring stalk-color-above-ring stalk-color-below-ring  \
0                           s                      w                      w
1                           s                      w                      w
2                           s                      w                      w
3                           s                      w                      w
4                           s                      w                      w
...                       ...                    ...                    ...
8117                        s                      p                      w
8118                        s                      p                      w
8119                        s                      o                      o
8120                        s                      o                      o
8121                        s                      o                      o
```

```
     veil-type veil-color ring-number ring-type spore-print-color population  \
0            p          w           o         p                 k          s
1            p          w           o         p                 n          n
2            p          w           o         p                 n          n
3            p          w           o         p                 k          s
4            p          w           o         e                 n          a
...        ...        ...         ...       ...               ...        ...
8117         p          w           o         e                 w          v
8118         p          w           o         e                 w          v
8119         p          o           o         p                 b          c
8120         p          n           o         p                 b          v
8121         p          o           o         p                 b          c

     habitat
0          u
1          g
2          m
3          u
4          g
...      ...
8117       d
8118       d
8119       l
8120       l
8121       l

[6499 rows x 22 columns] 0       p
1       e
2       e
3       p
4       e
       ..
8117    p
8118    p
8119    e
8120    e
8121    e
Name: class, Length: 6499, dtype: object
```

```
     cap-shape cap-surface cap-color bruises odor gill-attachment  \
0            x           s         n       t    p               f   
1            x           s         y       t    a               f   
2            b           s         w       t    l               f   
3            x           y         w       t    p               f   
4            x           s         g       f    n               f   
...        ...         ...       ...     ...  ...             ...   
8117         k           s         e       f    y               f   
8118         k           y         n       f    f               f   
8119         k           s         n       f    n               a   
8120         x           s         n       f    n               a   
8121         f           s         n       f    n               a   

     gill-spacing gill-size gill-color stalk-shape  ...  \
0               c         n          k           e  ...   
1               c         b          k           e  ...   
2               c         b          n           e  ...   
3               c         n          n           e  ...   
4               w         b          k           t  ...   
...           ...       ...        ...         ...  ...   
8117            c         n          b           t  ...   
8118            c         n          b           t  ...   
8119            c         b          y           e  ...   
8120            c         b          y           e  ...   
8121            c         b          n           e  ...   

     stalk-surface-below-ring stalk-color-above-ring stalk-color-below-ring  \
0                           s                      w                      w   
1                           s                      w                      w   
2                           s                      w                      w   
3                           s                      w                      w   
4                           s                      w                      w   
...                       ...                    ...                    ...   
8117                        s                      p                      w   
8118                        s                      p                      w   
8119                        s                      o                      o   
8120                        s                      o                      o   
8121                        s                      o                      o   
```

```
     veil-type veil-color ring-number ring-type spore-print-color population  \
0            p          w           o         p                 k          s
1            p          w           o         p                 n          n
2            p          w           o         p                 n          n
3            p          w           o         p                 k          s
4            p          w           o         e                 n          a
...        ...        ...         ...       ...               ...        ...
8117         p          w           o         e                 w          v
8118         p          w           o         e                 w          v
8119         p          o           o         p                 b          c
8120         p          n           o         p                 b          v
8121         p          o           o         p                 b          c

     habitat
0          u
1          g
2          m
3          u
4          g
...      ...
8117       d
8118       d
8119       l
8120       l
8121       l

[6499 rows x 22 columns] 0       p
1       e
2       e
3       p
4       e
       ..
8117    p
8118    p
8119    e
8120    e
8121    e
Name: class, Length: 6499, dtype: object
Test Accuracy: 99.94
```

Printing a prediction.

In [18]:
```
#Query
query = np.array([['b','s','y','t','a','f','c','b','g','e','c','s','s','w','w','p','w','o','p','k','s','m']])
print("Query 1:- {} ---> {}".format(query, nb_clf.predict(query)))
```

```
Query 1:- [['b' 's' 'y' 't' 'a' 'f' 'c' 'b' 'g' 'e' 'c' 's' 's' 'w' 'w' 'p' 'w' 'o'
  'p' 'k' 's' 'm']] ---> ['e']
```

The model has predicted the new class as ‘e’ which is 100% correct.

Below exercise's python file name: Algorithms_Assignment_Task2_2

Exercise 2:

The following Gaussian Naïve Bayes Classifier model predicts whether a person makes over 50K a year or less than that.

Loading data and viewing the first five rows.

In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math


df = pd.read_csv("D:\\adult.data")
df.head()
```

Out[1]:

	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K

Since there are no headers on the dataset, it is possible to hide the header by using header=None.

In [2]:
```python
df = pd.read_csv("D:\\adult.data", header=None)
df.head()
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

Checking the columns of categorical variables. There are 9 columns that contain categorical variables.

```
In [3]: # find categorical variables

        categorical = [var for var in df.columns if df[var].dtype=='O']

        print('There are {} categorical variables\n'.format(len(categorical)))

        print('The categorical variables are :\n\n', categorical)

        There are 9 categorical variables

        The categorical variables are :

         [1, 3, 5, 6, 7, 8, 9, 13, 14]
```

Checking null values in categorical variables.

```
In [4]: # check null values in categorical variables
        df[categorical].isnull().sum()

Out[4]: 1     0
        3     0
        5     0
        6     0
        7     0
        8     0
        9     0
        13    0
        14    0
        dtype: int64
```

Adding proper column names.

```
In [5]: col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status', 'occupation', 'relationship',
                     'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income']

        df.columns = col_names

        df.columns

Out[5]: Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
               'marital_status', 'occupation', 'relationship', 'race', 'sex',
               'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
               'income'],
              dtype='object')
```

View the first five rows of the dataset with the assigned header names.

```
In [6]: # preview the dataset

        df.head()

Out[6]:
```

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	nati
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	Ui
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	Ui
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	Ui
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	Ui
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	

Viewing the summary of the dataset.

```
In [7]: # summary

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education_num   32561 non-null  int64 
 5   marital_status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital_gain    32561 non-null  int64 
 11  capital_loss    32561 non-null  int64 
 12  hours_per_week  32561 non-null  int64 
 13  native_country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Viewing the categorical variables (header names)

```
In [8]: # categorical variables

categorical = [var for var in df.columns if df[var].dtype=='O']

print('There are {} categorical variables\n'.format(len(categorical)))

print('The categorical variables are :\n\n', categorical)
```

```
There are 9 categorical variables

The categorical variables are :

 ['workclass', 'education', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'native_country', 'income']
```

Viewing the first five rows of the categorical variables.

```
In [9]: # categorical variables

        df[categorical].head()
```

Out[9]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	income
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States	<=50K
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	<=50K
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	<=50K
3	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States	<=50K
4	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba	<=50K

Checking the missing values in the categorical variables (with header names)

```
In [10]: # missing values

         df[categorical].isnull().sum()

Out[10]: workclass         0
         education         0
         marital_status    0
         occupation        0
         relationship      0
         race              0
         sex               0
         native_country    0
         income            0
         dtype: int64
```

Viewing the frequency counts of values in the categorical variables. It is observable that there is an unknown label '?' with 1836 frequency under the 'workclass' categorical variable.

```
In [11]: # frequency counts of values in categorical variables

         for var in categorical:

             print(df[var].value_counts())
```

```
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: workclass, dtype: int64
 HS-grad         10501
 Some-college     7291
 Bachelors        5355
 Masters          1723
 Assoc-voc        1382
 11th             1175
 Assoc-acdm       1067
 10th              933
 7th-8th           646
```

Viewing frequency distribution of categorical variables.

In this way, it is easy to interpret the dataset's information.

```
In [12]: # frequency distribution of categorical variables

         for var in categorical:

             print(df[var].value_counts()/float(len(df)))
```

```
 Private             0.697030
 Self-emp-not-inc    0.078038
 Local-gov           0.064279
 ?                   0.056386
 State-gov           0.039864
 Self-emp-inc        0.034274
 Federal-gov         0.029483
 Without-pay         0.000430
 Never-worked        0.000215
Name: workclass, dtype: float64
 HS-grad         0.322502
 Some-college    0.223918
 Bachelors       0.164461
 Masters         0.052916
 Assoc-voc       0.042443
 11th            0.036086
 Assoc-acdm      0.032769
 10th            0.028654
 7th-8th         0.019840
```

Checking labels in the 'workclass' variable.

It is observed that the presence of an unknown label ' ?'.

```
In [13]: # labels in workclass variable

         df.workclass.unique()

Out[13]: array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',
                ' Local-gov', ' ?', ' Self-emp-inc', ' Without-pay',
                ' Never-worked'], dtype=object)
```

Viewing the frequency of this unknown label.

```
In [14]: # frequency distribution of values in workclass variable

         df.workclass.value_counts()

Out[14]:  Private             22696
          Self-emp-not-inc     2541
          Local-gov            2093
          ?                    1836
          State-gov            1298
          Self-emp-inc         1116
          Federal-gov           960
          Without-pay            14
          Never-worked            7
         Name: workclass, dtype: int64
```

Replacing the unknown label ‘ ?’ with NaN (Not a Number) and then re-viewing the ‘workclass’ labels’ frequency. Normally the missing values are coded as NaN.

```
In [22]: # replace '?' label in workclass variable with `NaN`

         df['workclass'].replace(' ?', np.NaN, inplace=True)
```

```
In [23]: # requency distribution of values in workclass variable

         df.workclass.value_counts()

Out[23]:  Private             22696
          Self-emp-not-inc     2541
          Local-gov            2093
          State-gov            1298
          Self-emp-inc         1116
          Federal-gov           960
          Without-pay            14
          Never-worked            7
         Name: workclass, dtype: int64
```

Similarly, missing values in the ‘occupation’ and ‘native_country’ variables have been replaced with NaN.

```
In [48]: # replace '?' values in occupation variable with `NaN`

         df['occupation'].replace(' ?', np.NaN, inplace=True)
```

```
In [49]: # labels in occupation variable

         df.occupation.unique()

Out[49]: array([' Adm-clerical', ' Exec-managerial', ' Handlers-cleaners',
                ' Prof-specialty', ' Other-service', ' Sales', ' Craft-repair',
                ' Transport-moving', ' Farming-fishing', ' Machine-op-inspct',
                ' Tech-support', nan, ' Protective-serv', ' Armed-Forces',
                ' Priv-house-serv'], dtype=object)
```

In [51]:
```python
# replace '?' values in native_country variable with `NaN`

df['native_country'].replace(' ?', np.NaN, inplace=True)
```

In [52]:
```python
# check labels in native_country variable

df.native_country.unique()
```

Out[52]:
```
array([' United-States', ' Cuba', ' Jamaica', ' India', nan, ' Mexico',
       ' South', ' Puerto-Rico', ' Honduras', ' England', ' Canada',
       ' Germany', ' Iran', ' Philippines', ' Italy', ' Poland',
       ' Columbia', ' Cambodia', ' Thailand', ' Ecuador', ' Laos',
       ' Taiwan', ' Haiti', ' Portugal', ' Dominican-Republic',
       ' El-Salvador', ' France', ' Guatemala', ' China', ' Japan',
       ' Yugoslavia', ' Peru', ' Outlying-US(Guam-USVI-etc)', ' Scotland',
       ' Trinadad&Tobago', ' Greece', ' Nicaragua', ' Vietnam', ' Hong',
       ' Ireland', ' Hungary', ' Holand-Netherlands'], dtype=object)
```

In [53]:
```python
# check frequency distribution of values in native_country variable

df.native_country.value_counts()
```

Out[53]:
```
 United-States                 29170
 Mexico                          643
 Philippines                     198
 Germany                         137
 Canada                          121
 Puerto-Rico                     114
 El-Salvador                     106
 India                           100
 Cuba                             95
 England                          90
 Jamaica                          81
 South                            80
 China                            75
 Italy                            73
 Dominican-Republic               70
 Vietnam                          67
 Guatemala                        64
 Japan                            62
```

Checking null values in the categorical variables after replacing missing values.

In [55]:
```
# check null values
df[categorical].isnull().sum()
```

Out[55]:
```
workclass         1836
education            0
marital_status       0
occupation        1843
relationship         0
race                 0
sex                  0
native_country     583
income               0
dtype: int64
```

Checking the number of labels each categorical variable contains. (Cardinality)

The variable ‘native_country’ has a high cardinality.

Usually a high cardinality may cause some serious issues on the ML model.

In [56]:
```
# check for cardinality in categorical variables

for var in categorical:

    print(var, ' contains ', len(df[var].unique()), ' labels')
```

```
workclass  contains  9  labels
education  contains  16  labels
marital_status  contains  7  labels
occupation  contains  15  labels
relationship  contains  6  labels
race  contains  5  labels
sex  contains  2  labels
native_country  contains  42  labels
income  contains  2  labels
```

Finding the numerical variables.

```
In [57]: # numerical variables

numerical = [var for var in df.columns if df[var].dtype!='O']

print('There are {} numerical variables\n'.format(len(numerical)))

print('The numerical variables are :', numerical)

There are 6 numerical variables

The numerical variables are : ['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss', 'hours_per_week']
```

Viewing the first five rows of the numerical variables.

```
In [58]: # numerical variables

df[numerical].head()

Out[58]:
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
0	39	77516	13	2174	0	40
1	50	83311	13	0	0	13
2	38	215646	9	0	0	40
3	53	234721	7	0	0	40
4	28	338409	13	0	0	40

Checking missing values in numerical variables.

```
In [59]: # missing values in numerical variables

df[numerical].isnull().sum()

Out[59]: age               0
         fnlwgt            0
         education_num     0
         capital_gain      0
         capital_loss      0
         hours_per_week    0
         dtype: int64
```

Declaring the target variable and features. Here the target variable is the income column.

```
In [60]: # feature vector and target variable

X = df.drop(['income'], axis=1)

y = df['income']
```

Split the dataset into training set and testing set based on the 75% 25% scenario.

```
In [61]: # split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Checking sizes of the training set and testing set and finding data types of the training set.

```
In [63]: # shape of X_train and X_test

         X_train.shape, X_test.shape

Out[63]: ((24420, 14), (8141, 14))
```

```
In [64]: # data types in X_train

         X_train.dtypes

Out[64]: age                int64
         workclass         object
         fnlwgt             int64
         education         object
         education_num      int64
         marital_status    object
         occupation        object
         relationship      object
         race              object
         sex               object
         capital_gain       int64
         capital_loss       int64
         hours_per_week     int64
         native_country    object
         dtype: object
```

Displaying categorical and numerical variables based on their data types.

```
In [65]: # categorical variables

categorical = [col for col in X_train.columns if X_train[col].dtypes == 'O']

categorical
```

```
Out[65]: ['workclass',
 'education',
 'marital_status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native_country']
```

```
In [66]: # numerical variables

numerical = [col for col in X_train.columns if X_train[col].dtypes != 'O']

numerical
```

```
Out[66]: ['age',
 'fnlwgt',
 'education_num',
 'capital_gain',
 'capital_loss',
 'hours_per_week']
```

Viewing percentages of missing values in the categorical variables in the training set.

```
In [67]: # percentage of missing values in the categorical variables in training set

X_train[categorical].isnull().mean()
```

```
Out[67]: workclass         0.055856
education         0.000000
marital_status    0.000000
occupation        0.056020
relationship      0.000000
race              0.000000
sex               0.000000
native_country    0.018550
dtype: float64
```

Printing only categorical variables with missing data.

```
In [68]: # categorical variables with missing data

         for col in categorical:
             if X_train[col].isnull().mean()>0:
                 print(col, (X_train[col].isnull().mean()))

         workclass 0.055855855855855854
         occupation 0.05601965601965602
         native_country 0.018550368550368855
```

Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located.

```
In [76]: # missing categorical variables with most frequent value

         for df2 in [X_train, X_test]:
             df2['workclass'].fillna(X_train['workclass'].mode()[0], inplace=True)
             df2['occupation'].fillna(X_train['occupation'].mode()[0], inplace=True)
             df2['native_country'].fillna(X_train['native_country'].mode()[0], inplace=True)
```

Finding missing values in categorical variables in the training set.

```
In [74]: # missing values in categorical variables in X_train

         X_train[categorical].isnull().sum()

Out[74]: workclass         0
         education         0
         marital_status    0
         occupation        0
         relationship      0
         race              0
         sex               0
         native_country    0
         dtype: int64
```

Finding missing values in categorical variables in the testing set.

In [77]:
```
# missing values in categorical variables in X_test

X_test[categorical].isnull().sum()
```

Out[77]:
```
workclass         0
education         0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
native_country    0
dtype: int64
```

Finding missing values in the train set.

In [78]:
```
# missing values in X_train

X_train.isnull().sum()
```

Out[78]:
```
age               0
workclass         0
fnlwgt            0
education         0
education_num     0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
dtype: int64
```

Finding missing values in the test set.

```
In [79]: # missing values in X_test

         X_test.isnull().sum()

Out[79]: age               0
         workclass         0
         fnlwgt            0
         education         0
         education_num     0
         marital_status    0
         occupation        0
         relationship      0
         race              0
         sex               0
         capital_gain      0
         capital_loss      0
         hours_per_week    0
         native_country    0
         dtype: int64
```

It is observed that there are no missing values in the test set and train set.

Viewing categorical variables.

```
In [80]: # Encoding

         # categorical variables

         categorical

Out[80]: ['workclass',
          'education',
          'marital_status',
          'occupation',
          'relationship',
          'race',
          'sex',
          'native_country']
```

Viewing the first five rows from the train set (categorical variables)

In [81]:
```
X_train[categorical].head()
```

Out[81]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_country
26464	Private	12th	Divorced	Transport-moving	Other-relative	Black	Male	United-States
16134	Private	Masters	Never-married	Other-service	Not-in-family	White	Female	United-States
4747	Private	Masters	Married-civ-spouse	Prof-specialty	Husband	White	Male	United-States
8369	Private	HS-grad	Married-civ-spouse	Sales	Husband	White	Male	United-States
5741	Private	1st-4th	Never-married	Machine-op-inspct	Own-child	White	Male	Puerto-Rico

Using one-hot encoding on the train and test sets.

In [83]:
```
# import category encoders

import category_encoders as ce
```

In [84]:
```
# encoding remaining variables with one-hot encoding

encoder = ce.OneHotEncoder(cols=['workclass', 'education', 'marital_status', 'occupation', 'relationship',
                                 'race', 'sex', 'native_country'])

X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
```

After one-hot encoding, viewing train set data.

In [86]:
```
X_train.head()
```

Out[86]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	workclass_8	fnlwgt	...	native_country_32	nati
26464	59	1	0	0	0	0	0	0	0	61885	...	0	
16134	71	1	0	0	0	0	0	0	0	180733	...	0	
4747	42	1	0	0	0	0	0	0	0	107762	...	0	
8369	26	1	0	0	0	0	0	0	0	35917	...	0	
5741	46	1	0	0	0	0	0	0	0	256522	...	0	

5 rows × 105 columns

In [87]:
```
X_train.shape
```

Out[87]: (24420, 105)

After one-hot encoding, viewing test set data.

In [88]:
```
X_test.head()
```

Out[88]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	workclass_8	fnlwgt	...	native_country_32	nati
22278	27	1	0	0	0	0	0	0	0	177119	...	0	
8950	27	1	0	0	0	0	0	0	0	216481	...	0	
7838	25	1	0	0	0	0	0	0	0	256263	...	0	
16505	46	1	0	0	0	0	0	0	0	147640	...	0	
19140	45	1	0	0	0	0	0	0	0	172822	...	0	

5 rows × 105 columns

In [89]:
```
X_test.shape
```

Out[89]: (8141, 105)

Feature scaling: Feature scaling is a data preprocessing technique used to transform the values of features or variables in a dataset to a similar scale. The purpose is to ensure that all features contribute equally to the model and to avoid the domination of features with larger values. By applying feature scaling, the dataset's features can be transformed to a more consistent scale, making it easier to build accurate and effective machine learning models. Scaling facilitates meaningful comparisons between features, improves model convergence, and prevents certain features from overshadowing others based solely on their magnitude.

In [91]:
```
cols = X_train.columns
```

In [92]:
```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [93]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [94]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

After applying feature scaling technique, RobustScaler.

In [95]:
```python
X_train.head()
```

Out[95]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	workclass_8	fnlwgt	...	native_country_32
0	1.157895	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.972949	...	0.0
1	1.789474	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.022866	...	0.0
2	0.263158	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.588550	...	0.0
3	-0.578947	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.190532	...	0.0
4	0.473684	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.657894	...	0.0

5 rows × 105 columns

In [96]:
```python
X_test.head()
```

Out[96]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	workclass_8	fnlwgt	...	native_country_32
0	-0.526316	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.007415	...	0.0
1	-0.526316	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.322395	...	0.0
2	-0.631579	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.655724	...	0.0
3	0.473684	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.254417	...	0.0
4	0.421053	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.043419	...	0.0

5 rows × 105 columns

Now the training and testing sets are ready to build the model.

Training the model.

In [97]:
```python
# the Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB


# instantiate the model
gnb = GaussianNB()


# fit the model
gnb.fit(X_train, y_train)
```

Out[97]:
```
▾ GaussianNB
GaussianNB()
```

Predicting the results.

```
In [98]: # predicting results

         y_pred = gnb.predict(X_test)

         y_pred

Out[98]: array([' <=50K', ' <=50K', ' >50K', ..., ' <=50K', ' >50K', ' <=50K'],
               dtype='<U6')
```

Checking the accuracy of the model.

```
In [102]: # calculating the model accuracy
          # y_test = true class labels
          # y_pred = predicted class labels

          from sklearn.metrics import accuracy_score

          print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))

          Model accuracy score: 0.8059
```

It is observed that the accuracy of this model is 80.59%.

Viewing the accuracy of the training set.

```
In [100]: y_pred_train = gnb.predict(X_train)

          y_pred_train

Out[100]: array([' <=50K', ' <=50K', ' >50K', ..., ' <=50K', ' >50K', ' <=50K'],
                dtype='<U6')

In [103]: print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))

          Training-set accuracy score: 0.8059
```

Compare accuracy scores of the training and test sets.

```
In [104]: # scores on training and test set

          print('Training set score: {:.4f}'.format(gnb.score(X_train, y_train)))

          print('Test set score: {:.4f}'.format(gnb.score(X_test, y_test)))

Training set score: 0.8059
Test set score: 0.8059
```

From the above, both scores are almost identical, meaning that there is no overfitting. Overfitting is an undesirable machine learning behavior that occurs when the machine learning model gives accurate predictions for training data but not for new data. When data scientists use machine learning models for making predictions, they first train the model on a known data set. Then, based on this information, the model tries to predict outcomes for new data sets. An overfit model can give inaccurate predictions and cannot perform well for all types of new data.

Even though the model accuracy (test set accuracy) is 0.8059, it is better to compare it with the null accuracy in order to certify the model's accuracy. The Null accuracy is the accuracy that could be achieved by always predicting the most frequent class.

Finding the class distribution in the test set.

```
In [105]: # class distribution in test set

          y_test.value_counts()

Out[105]:  <=50K    6159
           >50K     1982
          Name: income, dtype: int64
```

Calculating the null accuracy using the most frequent class, 6159.

In [109]:
```
# the null accuracy score

null_accuracy = (6159/(6159+1982))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.7565
```

Observation:

Model accuracy score = 0.8059

Null accuracy score = 0.7565

Review: This classification model should be good at predicting class labels.

Finding the effectiveness of the model:

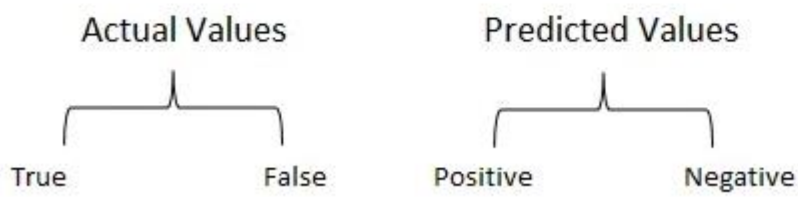
1. **Confusion Matrix:** Finding the type of errors that the model can make.

```
In [110]: # Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[4968 1191]
 [ 389 1593]]

True Positives(TP) =  4968

True Negatives(TN) =  1593

False Positives(FP) =  1191

False Negatives(FN) =  389
```

According to the confusion matrix, the total number of correct predictions is 6561 and the total number of incorrect predictions is 1580.

Note: predicted values describe as Positive and Negative whereas the actual values describe as True and Positive.



TP 4968: the model predicted 4968 times as positive, and it is true.

TN 1593: the model predicted 1593 times as negative, and it is true.

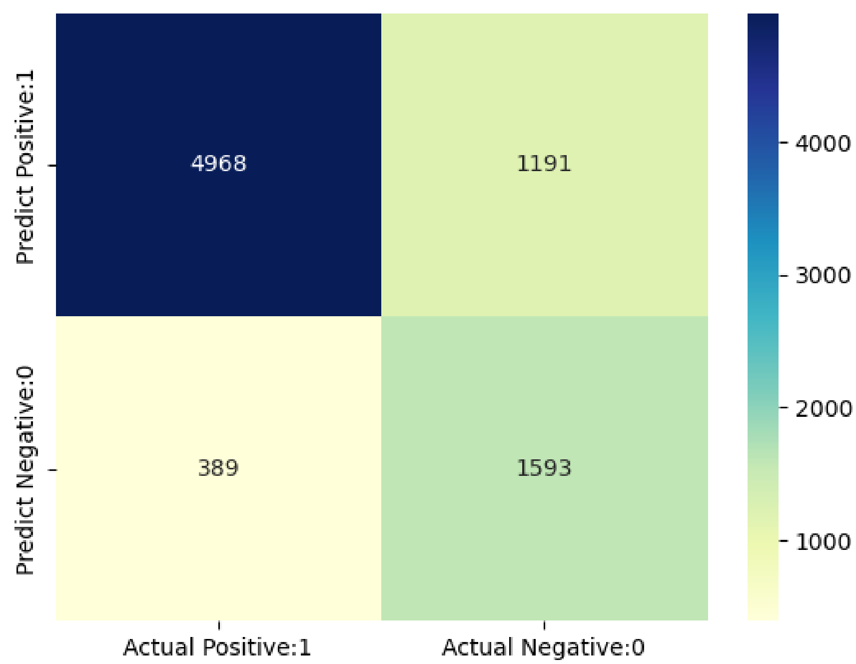
FP (type 1 error): the model predicted 1191 times as positive, but it is false.

FN (type 2 error): the model predicted 389 times as negative, but it is false.

Presenting the confusion matrix using seaborn heatmap.

In [112]:
```
# confusion matrix with seaborn heatmap
import seaborn as sns

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                  index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[112]: <Axes: >



2. **Classification report:** It is possible to use the classification report to evaluate the model's performance. In the classification report, it displays the scores of precision, recall, f1 and support.

```
In [113]: from sklearn.metrics import classification_report

          print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

        <=50K       0.93      0.81      0.86      6159
         >50K       0.57      0.80      0.67      1982

     accuracy                           0.81      8141
    macro avg       0.75      0.81      0.77      8141
 weighted avg       0.84      0.81      0.82      8141
```

Accuracy: This is how close the model can predict with a new value to be positioned at TP.

From all the classes (positive and negative), how many of them the model has predicted correctly. In this case, it is,

Accuracy = (6561/8141) x 100 = 80.59%.

Since the accuracy is comparatively high, the model's performance can be considered high.

F-Measure:

It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

$$F\text{ - measure} = \frac{2*\text{Recall}*\text{Precision}}{\text{Recall} + \text{Precision}}$$

Review:

When the income below or equal 50,000: recall is 0.86

When the income greater than 50,000: recall is 0.67

Classification accuracy:

The classification accuracy is the ratio of the number of correct predictions to the total number of input samples.

Finding the classification accuracy.

```
In [114]: TP = cm[0,0]
          TN = cm[1,1]
          FP = cm[0,1]
          FN = cm[1,0]

In [115]: # classification accuracy

          classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

          print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))

          Classification accuracy : 0.8059
```

Since the classification accuracy is comparatively high, the model's performance can be considered high.

Classification error:

Classification error is a type of measurement error by which the model does not provide a true response.

```
In [116]: # classification error

          classification_error = (FP + FN) / float(TP + TN + FP + FN)

          print('Classification error : {0:0.4f}'.format(classification_error))

          Classification error : 0.1941
```

Since the classification error is comparatively very low, the model's performance can be considered high.

Precision: Precision can be seen as a measure of quality and a higher precision means that an algorithm returns more relevant results than irrelevant ones.

The below equation can be explained by saying, from all the classes the model has predicted as positive, how many are actually positive.

$$\textit{Precision} = \frac{TP}{TP + FP}$$

```
In [117]: # precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))
```

```
Precision : 0.8066
```

Since the precision is comparatively high, the model's performance can be considered high.

Recall: This can be seen as measure of quantity and a high recall means that an algorithm returns most of the relevant results (whether or not irrelevant ones are also returned).

The below equation can be explained by saying, from all the positive classes, how many the model predicted correctly.

$$\textit{Recall} = \frac{TP}{TP + FN}$$

```
In [118]: recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.9274
```

True Positive Rate is synonymous with Recall**.**

```
In [119]: true_positive_rate = TP / float(TP + FN)


          print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))

          True Positive Rate : 0.9274
```

Since the recall value or the true positive value is comparatively high, the model's performance can be considered high.

Calculating the false positive rate.

```
In [120]: false_positive_rate = FP / float(FP + TN)


          print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))

          False Positive Rate : 0.4278
```

Specificity: When sensitivity is used to evaluate model performance, it is often compared to specificity. Specificity measures the proportion of true negatives that are correctly identified by the model. High specificity means that the model is correctly identifying most of the negative results, while a low specificity means that the model is mislabelling a lot of negative results as positive.

Calculating the specificity.

```
In [121]: specificity = TN / (TN + FP)

          print('Specificity : {0:0.4f}'.format(specificity))


          Specificity : 0.5722
```

Calculating class probabilities:

2 classes:

Class 0: <=50K that a person's income is less than or equal to 50,000

Class 1: >50K that a person's income is greater than 50,000

```
In [122]: # class probabilities

          # first 10 predicted probabilities of two classes- 0 and 1

          y_pred_prob = gnb.predict_proba(X_test)[0:10]

          y_pred_prob

Out[122]: array([[9.99999529e-01, 4.71254904e-07],
                 [9.99653330e-01, 3.46670117e-04],
                 [1.38701368e-01, 8.61298632e-01],
                 [2.05878343e-04, 9.99794122e-01],
                 [1.13213992e-08, 9.99999989e-01],
                 [8.90836096e-01, 1.09163904e-01],
                 [9.99999945e-01, 5.45662843e-08],
                 [9.99993961e-01, 6.03863116e-06],
                 [9.87132407e-01, 1.28675929e-02],
                 [9.99999997e-01, 2.73441203e-09]])
```

In [123]:
```python
# store probabilities in dataframe

y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Prob of - <=50K', 'Prob of - >50K'])

y_pred_prob_df
```

Out[123]:

	Prob of - <=50K	Prob of - >50K
0	9.999995e-01	4.712549e-07
1	9.996533e-01	3.466701e-04
2	1.387014e-01	8.612986e-01
3	2.058783e-04	9.997941e-01
4	1.132140e-08	1.000000e+00
5	8.908361e-01	1.091639e-01
6	9.999999e-01	5.456628e-08
7	9.999940e-01	6.038631e-06
8	9.871324e-01	1.286759e-02
9	1.000000e+00	2.734412e-09

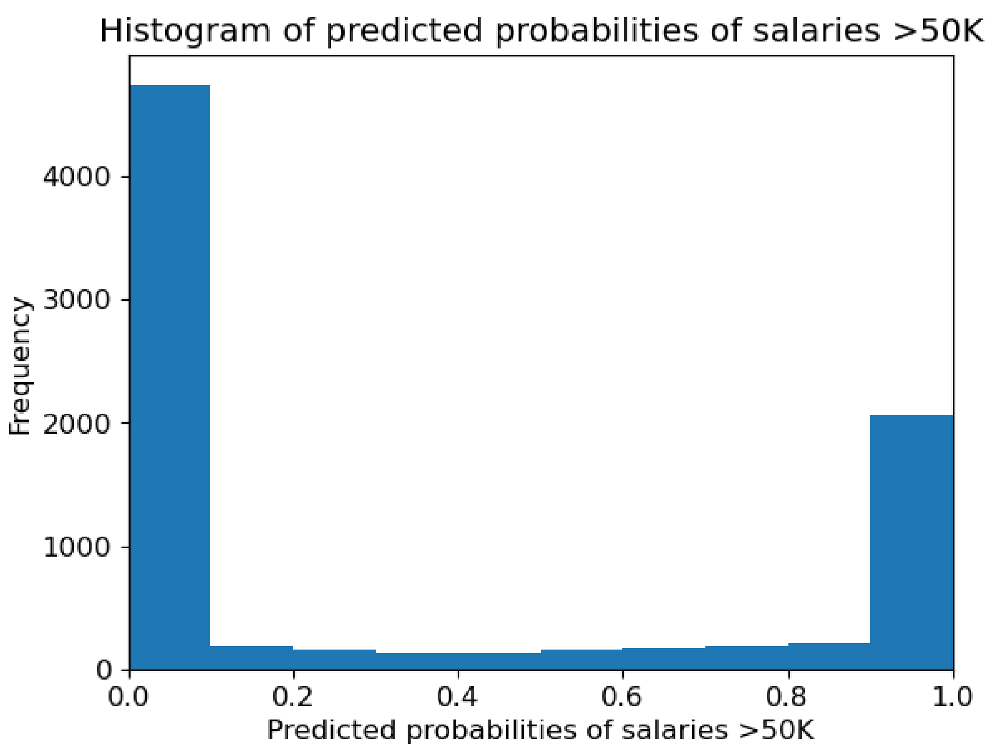
In [124]:
```python
# print the first 10 predicted probabilities for class 1 - Probability of >50K

gnb.predict_proba(X_test)[0:10, 1]
```

Out[124]:
```
array([4.71254904e-07, 3.46670117e-04, 8.61298632e-01, 9.99794122e-01,
       9.99999989e-01, 1.09163904e-01, 5.45662843e-08, 6.03863116e-06,
       1.28675929e-02, 2.73441203e-09])
```

In [125]:
```python
# store the predicted probabilities for class 1 - Probability of >50K

y_pred1 = gnb.predict_proba(X_test)[:, 1]
```

In [126]:
```
# plot histogram of predicted probabilities


# adjust the font size
plt.rcParams['font.size'] = 12


# plot histogram with 10 bins
plt.hist(y_pred1, bins = 10)


# set the title of predicted probabilities
plt.title('Histogram of predicted probabilities of salaries >50K')


# set the x-axis limit
plt.xlim(0,1)


# set the title
plt.xlabel('Predicted probabilities of salaries >50K')
plt.ylabel('Frequency')
```

Out[126]:
```
Text(0, 0.5, 'Frequency')
```



According to the diagram,

- The first column tell us that there are approximately 5700 observations with probability between 0.0 and 0.1 whose salary is <=50K.
- There are relatively small number of observations that predict that the salaries will be >50K.
- Majority of observations predict that the salaries will be <=50K.

ROC AUC: Receiver Operating Characteristic - Area Under Curve

This is another way of measuring the classification model's performance.

```
In [128]: # compute ROC AUC

          from sklearn.metrics import roc_auc_score

          ROC_AUC = roc_auc_score(y_test, y_pred1)

          print('ROC AUC : {:.4f}'.format(ROC_AUC))

          ROC AUC : 0.8942
```

ROC AUC is a single number summary of classifier performance. The higher the value, the better the classifier.

```
In [129]: # calculate cross-validated ROC AUC

          from sklearn.model_selection import cross_val_score

          Cross_validated_ROC_AUC = cross_val_score(gnb, X_train, y_train, cv=5, scoring='roc_auc').mean()

          print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))

          Cross validated ROC AUC : 0.8940
```

Since the ROC AUC value is comparatively high, it can be considered as the model is performing well.

k-Fold cross validation: This is another method of finding the performance of the classification model. The dataset is divided into k subsets or folds. The model is trained and evaluated k times, using a different fold as the validation set each time. Performance metrics from each fold are averaged to estimate the model's generalization performance.

```
In [130]: # Applying 10-Fold Cross Validation

          from sklearn.model_selection import cross_val_score

          scores = cross_val_score(gnb, X_train, y_train, cv = 10, scoring='accuracy')

          print('Cross-validation scores:{}'.format(scores))

Cross-validation scores:[0.80630631 0.82268632 0.7972973  0.8030303  0.80712531 0.7960688
 0.80712531 0.8001638  0.80917281 0.8046683 ]
```

```
In [131]: # compute Average cross-validation score

          print('Average cross-validation score: {:.4f}'.format(scores.mean()))

Average cross-validation score: 0.8054
```

So, it can be concluded that the model is accurate as 80.54% on average.

3. Analyze the results.

Based on the results of your Naive Bayes classifier, answer the following questions:

3.1. Discuss the performance of your classifier in terms of accuracy. Is it satisfactory, or do you think it could be improved?

1st exercise's classifier model accuracy: 99.94%

2nd exercise's classifier model accuracy: 80.59%

Both these models have high accuracy levels hence both of them are at a high performing level on their predictions.

However, in case, when need to improve these models' performance, it is possible to follow the below mentioned strategies.

1. **Feature Engineering:**
    - **Select appropriate features:** Selecting features that are most pertinent to the process of classification. By removing superfluous or irrelevant features, the performance of the model can be enhanced.
    - **Feature Scaling:** In order to guarantee that each feature contributes equally to the model, it is a best practise to normalize or standardize numerical features.

2. **Handling Missing Data:**
    - Making necessary adjustments for missing values. This is deciding whether to impute missing values or eliminate occurrences of missing values based on the type of data.

3. **Text Data Preprocessing:**
    - Preprocessing text data for text classification tasks can be achieved by converting text to lowercase, eliminating stop words, and stemming. When representing text features, methods like word embeddings or TF-IDF can be utilized.
4. **Handling Imbalanced Data:**
    - In order to balance the class distribution, when the dataset is unbalanced, oversampling the minority class or under-sampling the majority class is a better option, meaning that certain classes have a lot fewer instances than others.
5. **Model Selection:**

- Experimenting with other Naive Bayes variations based on the type of the data that the model has, like Multinomial Naive Bayes or Gaussian Naive Bayes, is another way of enhancing the performance of the model.
- Testing a few different algorithms can be done to evaluate performance and select the best categorization algorithm for the dataset.

6. **Hyperparameter Tuning:**
    - By adjusting hyperparameters, it is possible to determine which setup works best for the model. This may involve exploring the hyper parameter space using methods like grid search or random search.
7. **Cross-Validation:**
    - Using cross-validation, it is possible to evaluate the model's generalization performance. This provides a more accurate evaluation of the model's performance and helps in the detection of over fitting.
8. **Smoothing Techniques:**
    - Rare events may cause naive Bayes models to become sensitive. The issue of zero probabilities for unforeseen occurrences can be helped by smoothing techniques like Laplace smoothing (additive smoothing).
9. **Ensemble Methods:**
    - By combining several week classifiers using ensemble techniques like bagging or boosting, it is possible to turn them into a stronger one. This frequently enhances overall effectiveness.
10. **Model Evaluation:**
    - Depending on the type of classification problem, assessing the model using relevant metrics such as area under the ROC curve (AUC-ROC), precision, recall, F1-score, and so on can be mentioned.
11. **Update the Model:**
    - As new data becomes available, updating the model on a regular basis is particularly crucial in dynamic settings where the distribution of data could fluctuate over time.
12. **Domain Knowledge:**
    - Applying domain knowledge to the process of choosing features and creating models may result in more informed choices and enhanced model functionality.

3.2. Identify any potential limitations of using a Naive Bayes classifier for this dataset. Are there any assumptions made by the classifier that may not hold true in this context?

Limitations:

1. **Assumption of Independence:**

Given the class, Naive Bayes assumes that features are conditionally independent. This may not always be the case because numerous attributes are interdependent in real life. Dealing with associated features may result in less than ideal performance as a result.

2. **Sensitivity to Input Data Quality:**

The quality of input data affects the sensitivity of naive Bayes classifiers. The performance of the classifier may be impacted if the training data includes features that are noisy or irrelevant.

3. **Zero Probability Issue:**

Comparing Naive Bayes classifiers to more intricate models like decision trees or neural networks, the former exhibits greater expressiveness. They can find it difficult to identify complicated relationships in the data.

4. **Difficulty with Continuous and Numeric Features:**

Naive Bayes makes the assumption that features are categorical and that frequency counts can be used to estimate their probability.

5. **Imbalanced Datasets:**

In datasets that are unbalanced and have a substantial number of instances of one class compared to the others, Naive Bayes may not function well. It frequently favors the majority class, and the uncommon class could go unnoticed.

6. **Lack of Model Interpretability:**

Naive Bayes is easy to understand and straightforward, however it might not offer deep insights into the connections between features. Generally speaking, complex models are easier to interpret.

7. **Inability to Learn Feature Interactions:**

The independence requirement prevents Naive Bayes from modeling feature interactions. More complex models might be more suited in circumstances where feature interactions are important.

8. **Limited Performance on Text Classification Tasks:**

Although Naive Bayes is frequently used for text categorization, more sophisticated models, including deep learning techniques, may be able to better capture the semantics and contextual information included in the text.

3.2. Suggest alternative approaches or modifications to the Naive Bayes classifier that could potentially improve its performance on this dataset.

Suggestions:

1. **Relax the Independence Assumption:**

Consider utilizing more complex models, such as tree-based classifiers or ensemble techniques that can capture interactions between features, to relax the assumption that features are completely independent of one another.

2. **Kernelized Naive Bayes:**

   In order to enable the model to incorporate nonlinear relations among features, apply kernel approaches to modify the feature space. When handling non-linearly separable data, this can be quite helpful.

3. **Handling Continuous Features:**

   Gaussian Naive Bayes, which assumes a Gaussian distribution for continuous features, is an alternative to separating continuous features. This eliminates the requirement for separation and handles numerical data more effectively.

4. **Handling Imbalanced Data:**
   To solve imbalanced datasets and enhance the classifier's capacity to learn from the minority class, investigate strategies like oversampling the minority class, under sampling the majority class, or employing more sophisticated resampling algorithms.

5. **Feature Selection:**

   Reduce the influence of irrelevant or noisy features on the classification performance by using feature selection techniques to find and keep just the most useful features.

6. **Ensemble Methods:**

Using ensemble techniques like bagging or boosting, combine several Naive Bayes classifiers. By decreasing over fitting and boosting model robustness, this may help in enhancing overall performance.

7. **Advanced Text Processing Techniques:**

   Consider employing more sophisticated text processing methods for text classification tasks, including word embeddings (like Word2Vec, GloVe) or pre-trained language models (like BERT) to capture contextual information and semantic connections.

8. **Hybrid Models:**

   For building hybrid models, combine Naive Bayes with additional machine learning models. To benefit from the advantages of both, you can, for example, combine a simpler model with a Naive Bayes classifier.

9. **Cross-Validation and Parameter Tuning:**

   To evaluate the model's generalization performance, use cross-validation. Then, adjust the hyperparameters in order to see which configuration works best for your particular dataset.

10. **Probabilistic Classifiers:**

    More sophisticated probabilistic classifiers that are better able to represent complicated relationships between variables, like Bayesian networks or probabilistic graphical models, should be taken into consideration.

References

Jason Brownlee. (2023, 10 04). *A Gentle Introduction to k-fold Cross-Validation*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/k-fold-cross-validation/

Ajitesh Kumar . (2023, 11 18). *Machine Learning – Sensitivity vs Specificity Differences*. Retrieved from vitalflux.com: https://vitalflux.com/ml-metrics-sensitivity-vs-specificity-difference/

Aniruddha Bhandari . (2023, 10 27). *Feature Engineering: Scaling, Normalization, and Standardization*. Retrieved from www.analyticsvidhya.com: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

Hamming Distance. (n.d.). Retrieved from people.revoledu.com: https://people.revoledu.com/kardi/tutorial/Similarity/HammingDistance.html#google_vignette

Imputation of missing values. (n.d.). Retrieved from scikit-learn.org: https://scikit-learn.org/stable/modules/impute.html

Jason Brownlee. (2020, 08 19). *4 Distance Measures for Machine Learning*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/distance-measures-for-machine-learning/

Jason Brownlee. (2020, 02 24). *Develop k-Nearest Neighbors in Python From Scratch*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

MANAN KHURMA. (n.d.). *Euclidean Distance Formula*. Retrieved from www.cuemath.com: https://www.cuemath.com/euclidean-distance-formula/

Maximum Manhattan distance between a distinct pair from N coordinates. (2023, 01 04). Retrieved from www.geeksforgeeks.org: https://www.geeksforgeeks.org/maximum-manhattan-distance-between-a-distinct-pair-from-n-coordinates/

Rajeshsha. (2023, 11 30). *Detect and Remove the Outliers using Python*. Retrieved from www.geeksforgeeks.org: https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/

Rijk de Wet. (2023, 06 05). *Manhattan Distance Calculator*. Retrieved from www.omnicalculator.com: https://www.omnicalculator.com/math/manhattan-distance

Sarang Narkhede. (2018, 05 09). *Understanding Confusion Matrix*. Retrieved from towardsdatascience.com: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

What is Overfitting? (n.d.). Retrieved from aws.amazon.com: https://aws.amazon.com/what-is/overfitting/

www.shiksha.com. (2023, 03 06). *How to Compute Euclidean Distance in Python*. Retrieved from www.shiksha.com: https://www.shiksha.com/online-courses/articles/how-to-compute-euclidean-distance-in-python/

Zach. (2020, 12 17). *How to Calculate Hamming Distance in Python (With Examples)*. Retrieved from www.statology.org: https://www.statology.org/hamming-distance-python/