Machine Learning for Data Science

Assignment 1

Group Member

Mamoona Akbar

Anosha Khan

Experiment Description:

This study aims to construct and evaluate a Naïve Bayes Classifier for predicting income levels (>50K or <=50K) using a subset of the census income dataset provided by the Course Instructor. This dataset comprises four attributes: Sector, Education, Relationship, and Sex, each associated with corresponding income labels.

The primary steps of the experiment are:

Data Preprocessing:

- Removal of samples with missing values.
- Transformation of the Education attribute to reduce the number of distinct values to five categories: Master, Bachelor, Doctorate, Primary, and High School.
- Eliminating samples with relationship values other than Wife, Husband, or Unmarried.

Development of Naïve Bayes Classifier:

- Utilization of the assumption of conditional independence between features given the class label to construct the classifier.

Training Model and Testing the Results:

- Application of the developed Naïve Bayes Classifier on the dataset.
- Evaluation of the model's accuracy by comparing the predicted and actual values for the last 100 test samples.

In the data preprocessing phase, we meticulously handled missing data and standardized the Education attribute to simplify the classification process. Subsequently, we crafted a Naïve Bayes Classifier, leveraging the feature independence assumption, to predict income levels based on the provided attributes. Finally, we evaluated the accuracy of our model by comparing its predictions against the ground truth for a subset of test samples.

Methodology of Naïve Bayes Classifier:

- ### Understanding Naïve Bayes:
    - Naïve Bayes is a probabilistic classification algorithm based on Bayes' Theorem, which predicts the probability of a class label given the values of input features.

    - It assumes that the features are conditionally independent given the class label, simplifying the computation of probabilities.

- Data Preprocessing:
    - Cleanse the dataset by handling missing values and encoding categorical features.
    - Split the dataset into training and testing sets for model evaluation.

- Model Development:
    - Initialize the Naïve Bayes Classifier.
    - Calculate the prior probabilities of each class label by counting the frequency of each class in the training set.
    - Estimate the likelihood probabilities of each feature value given each class label by counting the frequency of feature-value-class triples in the training set.
    - Apply Laplace smoothing to handle zero probabilities.
    - Store the calculated probabilities in the classifier for later use.

- Model Training:
    - Train the Naïve Bayes Classifier using the training dataset.
    - Use the prior and likelihood probabilities to make predictions on the testing dataset.

- Model Evaluation:
    - Evaluate the performance of the classifier using various metrics such as accuracy.
    - Generate a confusion matrix to visualize the classification results and identify any misclassifications.
    - Analyse the model's strengths and weaknesses based on the evaluation metrics and adjust the parameters or features if necessary.

- Optimization and Fine-Tuning:
    - Experiment with different preprocessing techniques, such as feature scaling or feature engineering, to improve model performance.
    - Fine-tune the model hyperparameters, such as the smoothing parameter for Laplace smoothing, using techniques like cross-validation.
    - Consider ensemble methods or other advanced techniques to enhance the classifier's accuracy further.

- Deployment and Monitoring:
    - Deploy the trained Naïve Bayes Classifier in production environments to make real-time predictions.
    - Monitor the model's performance over time and retrain it periodically with new data to maintain its accuracy and relevance.

By following this methodology, one can effectively develop, train, and evaluate a Naïve Bayes Classifier for various classification tasks.

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.

Loading Data Set

```
file=pd.read_csv('adult.csv')
print(file)
```

```
                ?   Some-college          Husband     Male    >50K
0               ?        7th-8th    Not-in-family     Male   <=50K
1               ?   Some-college        Own-child     Male   <=50K
2               ?           10th          Husband     Male   <=50K
3               ?           10th        Own-child   Female   <=50K
4               ?        HS-grad          Husband     Male   <=50K
...           ...            ...              ...      ...     ...
23455   State-gov        HS-grad          Husband     Male   <=50K
23456   State-gov   Some-college    Not-in-family   Female   <=50K
23457   State-gov        7th-8th             Wife   Female   <=50K
23458   State-gov        HS-grad        Own-child   Female   <=50K
23459   State-gov   Some-college   Other-relative   Female   <=50K

[23460 rows x 5 columns]
```

Add Header in Data Set

Add header in csv file

```
file.columns=['sector','education','relation','gender','salary']
file.to_csv('adult_2.csv')
```

Null values check

Null attribute check

```
print(file.isnull().mean())
```

```
sector       0.0
education    0.0
relation     0.0
gender       0.0
salary       0.0
dtype: float64
```

Finding Unique Values in an Attribute

Finding Unique values in a attribute

Sector attribute unique values check

```
file['sector'].unique()
```

```
array([' ?', ' Federal-gov', ' Local-gov', ' Never-worked', ' Private',
       ' Self-emp-inc', ' State-gov'], dtype=object)
```

Education attribute unique values check

```
file['education'].unique()
```

```
array([' 7th-8th', ' Some-college', ' 10th', ' HS-grad', ' 1st-4th',
       ' Bachelors', ' Masters', ' Assoc-voc', ' 11th', ' 12th',
       ' 5th-6th', ' 9th', ' Assoc-acdm', ' Doctorate', ' Preschool',
       ' Prof-school'], dtype=object)
```

Relation attribute unique values check

```
file['relation'].unique()
```

```
array([' Not-in-family', ' Own-child', ' Husband', ' Wife', ' Unmarried',
       ' Other-relative'], dtype=object)
```

Gender attribute unique values check

```
file['gender'].unique()
```

```
array([' Male', ' Female'], dtype=object)
```

Salary attribute unique values check

```
file['salary'].unique()
```

```
array([' <=50K', ' >50K'], dtype=object)
```

Replace Values

Replace values

Education field

([' 7th-8th', ' Some-college', ' 10th', ' HS-grad', ' 1st-4th', ' Bachelors', ' Masters', ' Assoc-voc', ' 11th', ' 12th', ' 5th-6th', ' 9th', ' Assoc-acdm', ' Doctorate', ' Preschool', ' Prof-school'])

replace [' 1st-4th',' 5th-6th'] to 'Primary' and [' 7th-8th', ' 10th',' 11th', ' 12th', ' 9th'] to 'HighSchool'

```
file3=file.replace([' 1st-4th',' 5th-6th'], 'Primary')
file3=file3.replace([' 7th-8th', ' 10th',' 11th', ' 12th', ' 9th'],'HighSchool')
```

```
file3['education'].unique()
```

```
array(['HighSchool', ' Some-college', ' HS-grad', 'Primary', ' Bachelors',
       ' Masters', ' Assoc-voc', ' Assoc-acdm', ' Doctorate',
       ' Preschool', ' Prof-school'], dtype=object)
```

Dropping Rows

Dropping rows

Education attribute

Delete rows having values [' Some-college' , ' HS-grad' , ' Assoc-voc' , ' Assoc-acdm' , ' Preschool' , ' Prof-school']

```
file3 = file3.drop(file3[file3['education'] == ' Some-college'].index)
file3 = file3.drop(file3[file3['education'] == ' HS-grad'].index)
file3 = file3.drop(file3[file3['education'] == ' Assoc-voc'].index)
file3 = file3.drop(file3[file3['education'] == ' Assoc-acdm'].index)
file3 = file3.drop(file3[file3['education'] == ' Preschool'].index)
file3 = file3.drop(file3[file3['education'] == ' Prof-school'].index)
```

unique value in education attribute is

```
file3['education'].unique()
```

```
array(['HighSchool', 'Primary', ' Bachelors', ' Masters', ' Doctorate'],
      dtype=object)
```

Relation field

[' Not-in-family', ' Own-child', ' Husband', ' Wife', ' Unmarried', ' Other-relative']

Delete rows having values [' Not-in-family', ' Own-child', ' Other-relative']

```
file3 = file3.drop(file3[file3['relation'] == ' Not-in-family'].index)
file3 = file3.drop(file3[file3['relation'] == ' Own-child'].index)
file3 = file3.drop(file3[file3['relation'] == ' Other-relative'].index)
```

Unique values in relation attribute is

```
file3['relation'].unique()
```

```
array([' Husband', ' Wife', ' Unmarried'], dtype=object)
```

Sector attribute

[' ?', ' Federal-gov', ' Local-gov', ' Never-worked', ' Private', ' Self-emp-inc', ' State-gov']

Delete rows having ' ?'

```
file3 = file3.drop(file3[file3['sector'] == ' ?'].index)
```

After Preprocessing Unique Values in an Attribute

Find each attributes number of unique value count

Education attribute

```
file3['education'].value_counts()
```

```
education
 Bachelors    2143
HighSchool    1101
 Masters       856
 Doctorate     218
Primary        174
Name: count, dtype: int64
```

Sector attribute

```
file3['sector'].value_counts()
```

```
sector
 Private         2992
 Local-gov        618
 Self-emp-inc     352
 State-gov        342
 Federal-gov      188
Name: count, dtype: int64
```

Relation attribute

```
file3['relation'].value_counts()
```

```
relation
 Husband      3396
 Unmarried     694
 Wife          402
Name: count, dtype: int64
```

Gender attribute

```
file3['gender'].value_counts()
```

```
gender
 Male      3581
 Female     911
Name: count, dtype: int64
```

Salary Attribute

```
file3['salary'].value_counts()
```

```
salary
 >50K     2273
 <=50K    2219
Name: count, dtype: int64
```

After Preprocessing Total Samples

Total rows in dataset

```
file3.shape
```

```
(4492, 5)
```

Naïve Model Training

Splitting Data set

Splitting training and test set

```
#file3 = file3.sample(frac = 1)      ##### shuffle sample

train =file3.iloc[:-100]
test = file3.iloc[4392:]
print("training dataset : " ,train.shape)
print("testing dataset : " ,test.shape)
```

```
training dataset :  (4392, 5)
testing dataset :  (100, 5)
```

```
x_train = train.iloc[:,:-1]
y_train=train.iloc[:,-1]
```

```
print(x_train.shape)
print(y_train.shape)
```

```
(4392, 4)
(4392,)
```

```
x_test = test.iloc[:,:-1]
y_test=test.iloc[:,-1]
print(x_test.shape)
print(y_test.shape)
```

```
(100, 4)
(100,)
```

Model Training and Testing

Calculate the prior probabilities of each class label by counting the frequency of each class in the training set.

Estimate the likelihood probabilities of each feature value given each class label by counting the frequency of feature-value-class triples in the training set.

Apply Laplace smoothing to handle zero probabilities.

Store the calculated probabilities in the classifier for later use

Complete Code

```
import matplotlib.pyplot as plt

import numpy

from sklearn import metrics


class naivebayes:

    def __init__(self,x_train,y_train,x_test,y_test):

        self.x_train=x_train

        self.y_train=y_train

        self.x_test=x_test

        self.y_test=y_test


        ##### Unique values in y_train or y_test

        self.y=y_train.unique()
```

```
        ##### prior_probality
        self.pri_probality=[]
        ##### likelihood probality
        self.probabilities={}
        ##### predicted probalities
        self.predicted_probalities=[]
        ##### predicted_value
        self.predicted_value=[]



    def prior_probality(self):
        length_y=len(self.y)
        length_x_train=len(self.x_train)
        for i in range(length_y):
            count=0
            for j  in range(length_x_train):
                if self.y[i]==self.y_train.iloc[j]:
                    count=count+1
            self.pri_probality.append([self.y[i],(count/length_x_train)])
#         print(self.pri_probality)

    def probality_store(self):
        # Initialize an empty dictionary to store the probabilities
       # self.probabilities = {}

        # Iterate over each column in x_train
        for column in x_train.columns:
            # Initialize an empty dictionary for the current column
            column_dict = {}

            # Get the unique values in the current column
```

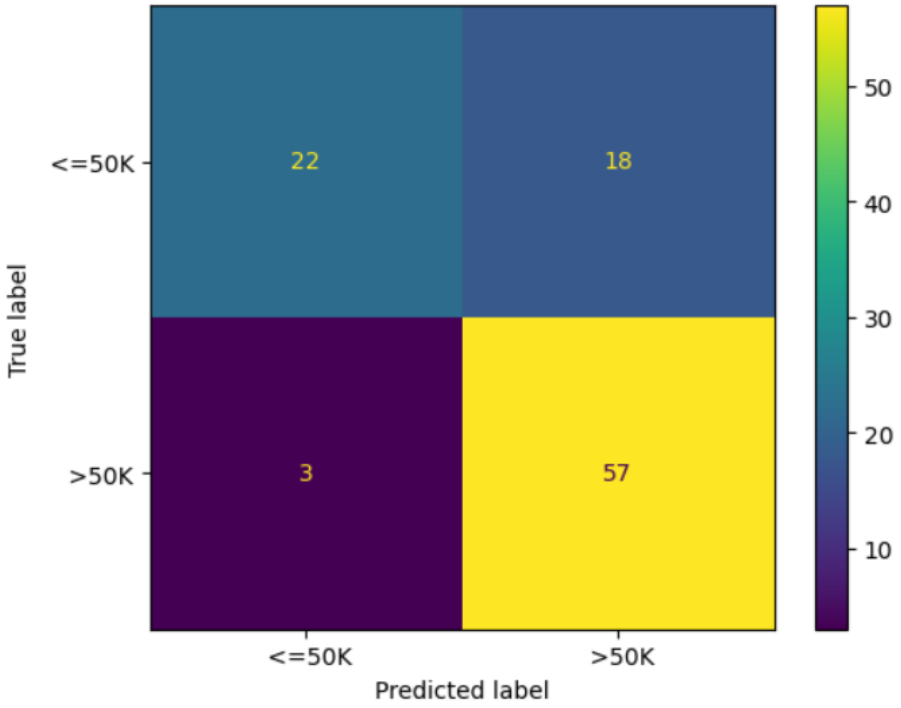
```
        unique_values = x_train[column].unique()

        # Iterate over each unique value in the current column
        for value in unique_values:
            # Initialize an empty dictionary for probabilities of each target value
            target_probabilities = {}

            # Iterate over each unique target value in y_train
            for target_value in y_train.unique():
                # Count occurrences of (value, target_value) pairs in the dataset
                count = ((x_train[column] == value) & (y_train == target_value)).sum()

                # Calculate the probability of (value, target_value) pair
                probability = count / len(y_train[y_train == target_value])

                # Store the probability in the dictionary
                target_probabilities[target_value] = probability

            # Store the dictionary of probabilities for the current value in the column
            column_dict[value] = target_probabilities

        # Store the dictionary of probabilities for the current column in the main dictionary
        self.probabilities[column] = column_dict
#       print(self.probabilities)
#       # Now you can access the probabilities for each category in each column
#       print(probabilities['sector'][' Federal-gov'])

    def predict_data(self):
        for i in range(len(self.x_test)):       ################  test data values
            pre_result=[]
            for j in range(len(self.y)):        ########  target specific value
```

```
            prob=float(self.pri_probality[j][1])
#               print("prob",prob)
            unique_values = x_train.columns


            for k in range(len(unique_values)):           ##### column
                if(float(self.probabilities[unique_values[k]][self.x_test.iloc[i,k]][self.y[j]]) == 0.0):  ## if any probality zero laplace smoothing apply
                    print("zero probality:",float(self.probabilities[unique_values[k]][self.x_test.iloc[i,k]][self.y[j]]))

prob=prob*((float(self.probabilities[unique_values[k]][self.x_test.iloc[i,k]][self.y[j]])+1)/1+len(self.probabilities[unique_values[k]]))
                else:
                    prob=prob*float(self.probabilities[unique_values[k]][self.x_test.iloc[i,k]][self.y[j]])
            pre_result.append(prob)
        self.predicted_probalities.append(pre_result)



    def accuracy(self):
        #### self.predicted_probalities   [-----,------] two probality
        #### self.y_test
        ##### self.y        list unique y_test label

        count=0
        for i in range(len(self.y_test)):
            maxi=0.0
            k=0
            for j in range(len(self.y)):      #### highest probality find
#               print("print",self.predicted_probalities[i][j])
                if( float(self.predicted_probalities[i][j])> float(maxi)):
                    maxi=float(self.predicted_probalities[i][j])
```

```
                k=j      #specific index highest value
            self.predicted_value.append(self.y[k])
            for j in range(len(self.y)):      #### highest probality find
                if ( (self.y[j]== self.y_test.iloc[i]) & (j==k)):     ### actual == predicted
                    count=count+1
        print("accuracy",count/len(self.y_test))



    def confusion_matrixx(self):
        actual = self.y_test
        predicted = self.predicted_value


        confusion_matrix= metrics.confusion_matrix(actual, predicted)


        cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = self.y)


        cm_display.plot()
        plt.show()



if __name__ == "__main__":
    abc=naivebayes(x_train,y_train,x_test,y_test)
    abc.prior_probality()
    abc.probality_store()
    abc.predict_data()
    abc.accuracy()
    abc.confusion_matrixx()
```

Accuracy

```
accuracy 0.79
```



Result and Discussion

In this study, we developed a Naive Bayes classifier to predict income class (>50K or <=50K) based on four attributes: Sector, education, relationship, and sex. Upon training the classifier on the provided dataset, we attained an accuracy of 79% on the test set. However, upon examining the confusion matrix of the results, it became evident that there were shortcomings in the predictions. Specifically, the classifier misclassified 3 samples belonging to the label >50K as <=50K, and 18 samples belonging to the label <=50K as >50K.

While achieving a 79% accuracy may seem satisfactory, it's clear that there is room for improvement. One potential limitation contributing to the achieved accuracy is the simplicity of both the dataset and the Naive Bayes classifier's assumption of feature independence. While Naive Bayes is well-suited for categorical features and is relatively simple to implement, it may struggle with datasets containing continuous or correlated features.

Furthermore, the large number of dropped samples during preprocessing may have also impacted the accuracy negatively. By including more samples, we could potentially enhance the classifier's performance.

To address these limitations and improve accuracy, future analyses could involve:

- Exploring additional features that may better capture the nuances of income prediction.
- Experimenting with different classification algorithms that may better handle the complexities of the dataset.

- Employing feature engineering techniques to extract more meaningful information from the existing attributes.

In conclusion, while Naive Bayes served as a suitable starting point for predicting salary class based on the selected attributes, further refinement, and exploration are necessary to achieve higher accuracy levels.

Conclusion

The Naive Bayes classifier demonstrated a commendable accuracy of 79%, suggesting its effectiveness in predicting salary classes based on the selected attributes. However, this study reveals avenues for further refinement and exploration to enhance predictive performance.

Addressing the limitations inherent in the Naive Bayes approach and considering alternative algorithms or feature engineering strategies could potentially improve the model's predictive capabilities. Exploring these avenues could lead to a more robust and accurate classifier, better suited to handle the complexities of real-world datasets.

In summary, while the Naive Bayes classifier provides a promising foundation for salary prediction, continued research and development efforts are warranted to unlock its full potential and achieve even higher levels of accuracy.