Group Assignment-4 (Naive Bayes)
Members:

- Avinash Aanand-(Roll No.: 2403RES99)
- Aditya Gupta-(Roll No.:2403RES85)
- Aman Kumar-(Roll No.:2403RES10)
- Sanjeev Kumar (Roll No.: 2403RES117)

- Due Date: April 2024
- Submitted To: Dr. Asif Ekbal

Artificial Intelligence & Machine Learning Lab, CS561/571 Assignment 04

Naïve Bayes

Table of Content

Problem Statements:

Statement

- Given a dataset containing various details about the details of employees such as 'workclass', 'education', 'occupation' etc. The task is to predict the 'income' of the employee given the other details. The salary field contains two values '<= 50k' and '>50k'.

Questions:

Implementation Details:

- Predict the 'income' of the employees using the given dataset having other details.
- The salary field contains two values '<= 50k' and '>50k'.
- Implement two Naive Bayes classifiers:
  - Gaussian Naive Bayes
  - Multinomial Naive Bayes
- You can use existing packages such as Scikit-Learn to create the classifier
- Perform a 60-20-20 split on the dataset for train, validation, and test set.
- Perform 4-fold cross-validation.
- Report accuracy on the test set for both classifiers.
- Drop all null-valued columns.

Solutions (Questions):

Tasks

Predict the 'income' of the employees using the given dataset having other details.
Solutions:
The salary field contains two values '<= 50k' and '>50k'.

- **Implement two Naive Bayes classifiers:**
- **Gaussian Naive Bayes**

Multinomial Naive Bayes
You can use existing packages such as Scikit-Learn to create the classifier
Perform a 60-20-20 split on the dataset for train, validation, and test set.
Perform 4-fold cross-validation.
Report accuracy on the test set for both classifiers.
Drop all null-valued columns.

Naive bayes Feasibility:

Naive Bayes algorithms are widely used in machine learning due to their simplicity, scalability, and efficiency. Here's a breakdown of the feasibility of using Naive Bayes:

- Simple and Easy to Implement: Naive Bayes algorithms are straightforward and easy to implement. The underlying assumption of feature independence simplifies the model structure and reduces the complexity of implementation.
- Fast Training and Prediction: Naive Bayes models are computationally efficient and have fast training and prediction times, making them suitable for large datasets and real-time applications.
- Works Well with High-Dimensional Data: Naive Bayes classifiers perform well in high-dimensional spaces, making them suitable for text classification, document categorization, and other tasks with a large number of features.
- Handles Categorical and Numeric Data: Naive Bayes algorithms can handle both categorical and numerical data, making them versatile for various types of datasets.
- Limited Modeling Power: The "naive" assumption of feature independence can be a limitation, especially when features are correlated. This can result in suboptimal performance compared to more complex models like decision trees or neural networks.
- Sensitive to Imbalanced Data: Naive Bayes classifiers can be sensitive to imbalanced datasets, where one class is significantly more prevalent than the others. This can lead to biased predictions towards the majority class.

- Requires Minimal Tuning: Naive Bayes models have few hyperparameters to tune, which simplifies the model selection and training process. However, hyperparameter tuning techniques like cross-validation can still be applied to optimize performance.

Overall, Naive Bayes algorithms are feasible and effective for many classification tasks, particularly in scenarios where computational efficiency and simplicity are priorities. However, it's essential to consider the assumptions and limitations of Naive Bayes when applying it to real-world problems.

Introduction to the problem:

Understanding the Problem

We have a dataset where each row represents an employee, and the columns contain features like:

- Workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked
- education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool
- occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces and potentially more features: Age, marital status, relationship, race, sex, capital gain, capital loss, hours per week, native country

Your goal is to create a model that can take these features as input and predict whether the income is "<=50K" or ">50K".

Why Naive Bayes?

Naive Bayes is a great choice for this type of classification problem for several reasons:

- **Simple and Interpretable**: Naive Bayes is easy to understand and implement. Its underlying assumptions make it computationally efficient.
- **Handles Categorical Data Well**: Many of your features (workclass, education, occupation) are categorical, which Naive Bayes is naturally suited for.
- **Good for High-Dimensional Data**: If you have a large number of features, Naive Bayes can still perform well, unlike some other algorithms that might struggle with the "curse of dimensionality."
- **Surprisingly Effective**: Despite its simplifying assumptions (that features are conditionally independent given the class), Naive Bayes often performs surprisingly well in practice.

What a Naive Bayes Works for Income Prediction

Training:

- **Calculate Prior Probabilities**: Determine the probability of an employee having an income "<=50K" (P(<=50K)) and the probability of having an income ">50K" (P(>50K)). These are simply the proportions of each class in your training data.
- **Calculate Likelihoods**: For each feature and each income class, calculate the probability of observing that feature value given the income class. For example, P(education=Bachelors | <=50K) is the probability that someone with a bachelor's degree earns <=50K.

Prediction (Inference):

- **Apply Bayes' Theorem**: When you get a new employee's data, you use Bayes' Theorem to combine the prior probabilities and likelihoods to calculate the probability of each income class given the features.
- **Choose the Most Likely Class**: Select the income class with the higher probability as your prediction.

Example Calculation

Let's say we have a new employee with these features:

- workclass: Private
- education: HS-grad
- occupation: Sales

Naive Bayes would calculate:

- P(<=50K | Private, HS-grad, Sales)
- P(>50K | Private, HS-grad, Sales)

It would then predict the income class with the higher probability.

Implementation

Important Considerations

- **Feature Engineering**: Consider transforming or creating new features (e.g., combining categories, creating bins for numerical features) to improve model performance.
- **Handling Missing Values**: Address missing values appropriately before training.

Evaluating Performance: Use metrics like accuracy, precision, recall, and F1-score to assess your model's performance.

Understanding of Naive Bayes with our problem

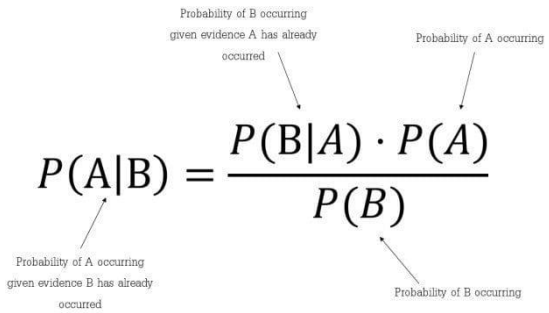
The Naive Bayes algorithm is a straightforward, yet potent tool used in machine learning for categorization purposes. It operates on the principles of Bayes' theorem, a mathematical equation that calculates the likelihood of an event based on the understanding of other related occurrences.

Training and interpreting Naive Bayes classifiers is comparatively easy hence making them a popular choice among machine learning practitioners. Their versatility allows them to be applied to a wide range of data types, encompassing text, images, and numerical data. This makes them an asset in the toolbox of any data scientist or machine learning engineer.

Bayes Theorem:

The Naive Bayes classifier works by calculating the probability of each class given the input features. The class with the highest probability is then predicted as the output.

To calculate the probability of each class, the Naive Bayes classifier uses Bayes' theorem:



In terms of Naive Bayes:

- `P` (`class` | features) = `P` (features | `class`) * `P`(`class`) / `P`(features)

where:

- P (class | features) is called *Posterior Probability*; it is the probability of the class given the input features
- P (features | class) also known as *Likelihood Probability* is the probability of the input features given the class
- P(class) is the *prior probability* of the class.
- P(features) is called *Marginal Probability*; it is the prior probability of the input features.

Assumptions:

Naïve Bayes algorithm is based on the below assumptions.

1. Independence assumption
    a. It is assumed that the input features are independent of each other given the class label.
    b. This means that the presence or absence of a particular feature does not impact the presence or absence of any other feature, given the class variable.
2. Equal prior probabilities:
    a. All classes contribute equally to the outcome.
    b. This means that no feature has more influence on the outcome than any other.
3. Normal distribution of features:
    a. Naïve bayes assumes that the features are normally distributed within the class.
    b. For example, in our dataset final weight or salary may not follow normal distribution with a class of male or female or based on race.

Comparatively Table Breakdown (different naive bayes classifiers)

Naïve Bayes Classifiers		
Classifiers	**Type of input features**	**Example applications**
Multinomial Naïve Bayes	Discrete Counts	Text classification, image classification, spam filtering
Bernoulli Naïve Bayes	Binary (0 or 1)	Image Classification, fraud detection
Gaussian Naïve Bayes	Continuous or normally distributed	Medical diagnostic, customer segmentation, risk assessment

Why Naive Bayes

Property	Description
Independence Assumption	It assumes that each feature makes an equal and independent contribution to the outcome.
Bayes Theorem	It is based on the Bayes theorem to calculate conditional probabilities.
Classification efficiency	Due to its computational simplicity, Naïve bayes is proficient for large datasets and quick model deployment.
Generative Learning	It belongs to the family of generative algorithms, modelling the distribution of inputs for a given class or category.
Unbiased nature	Unlike discriminative classifiers, Naïve Bayes does not prioritize learning the most important features for class differentiation.
Application in NLP	Widely used in NLP tasks due to its effectiveness in text classification and sentimental analysis

Naive Bayes Analysis

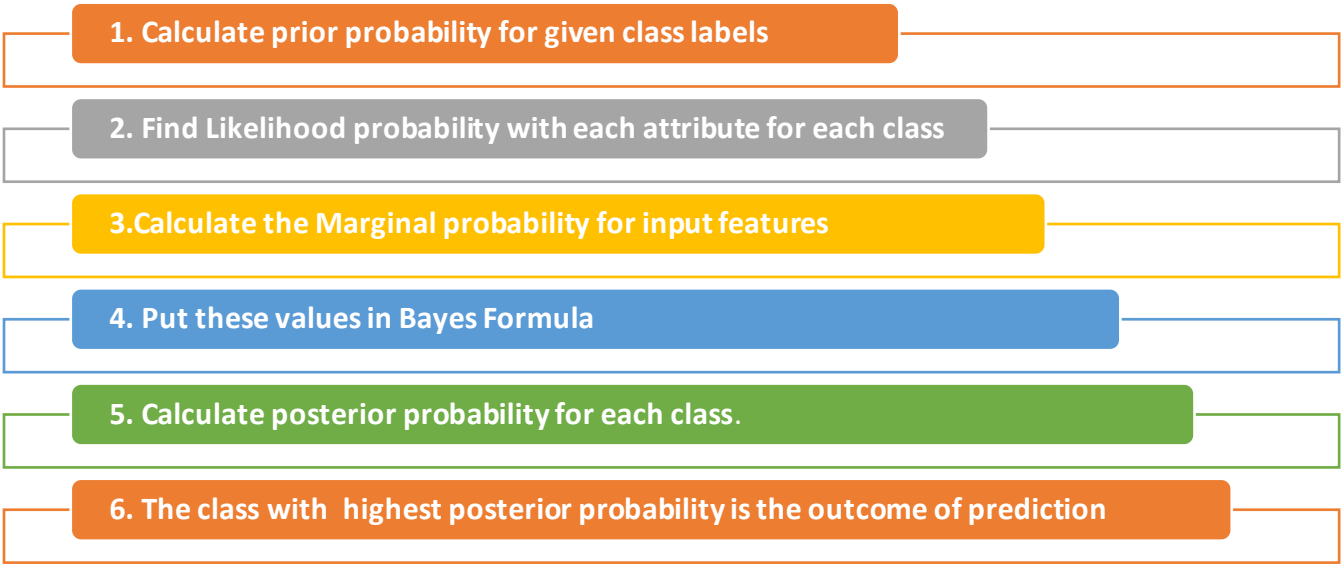
Through Pseudo Code

Naive Bayes Pseudo Code:

```
function NaiveBayes(dataset):
    # Data Preprocessing
    dataset = preprocess(dataset)
    training_set, testing_set = split_data(dataset)
    # Training Phase
    priors = calculate_priors(training_set)
    means, stdevs = calculate_means_stdevs(training_set)
    probabilities = calculate_probabilities(training_set)
    # Prediction Phase
    predictions = []
    for instance in testing_set:
        posteriors = []
        for class_label in unique_classes(training_set):
            posterior = priors[class_label]
            for feature, value in instance.items():
                if feature in continuous_features(training_set):
                    posterior *= gaussian_probability(value, means[class_label][feature], stdevs[class_label][feature])
                else:
                    posterior *= probabilities[class_label][feature][value]
            posteriors.append((class_label, posterior))
        predictions.append(max(posteriors, key=lambda x: x[1])[0])

    # Evaluation
    accuracy = calculate_accuracy(testing_set, predictions)
    return accuracy
```

Naive Bayes Algorithm



Naïve Bayes Code flow:

Setup:

- Import necessary libraries: pandas, numpy, matplotlib, seaborn, sklearn modules for data manipulation, visualization, model selection, preprocessing, model implementation, and evaluation metrics.

Data Loading and Preparation:

- Load Data: Reads the "adult.csv" file into a Pandas DataFrame.
- Handle Missing Data: Replaces missing values ('?') with NaN and then drops rows with any NaN values.
- Drop Irrelevant Feature: Removes the 'fnlwgt' column (final weight), as it's not directly relevant to income prediction.

Separate Target and Features:

- Creates a target variable target based on whether income is ">50K" (1) or "<=50K" (0).
- Creates a feature set features_data by removing the income column.

Feature Preprocessing:

Categorical Encoding:

- Identifies categorical columns in features_data.
- Applies one-hot encoding to convert these categorical features into numerical representations.

Numerical Scaling:

- Identifies numerical columns in features_data.
- Standardizes these numerical features using the StandardScaler.

Combine Features:

- Horizontally stacks (concatenates) the encoded categorical and scaled numerical features to create a unified feature matrix feature.

Data Splitting:

Split into Train/Validation/Test:

Splits the features and target into:

- Training and validation data (X_train_val, y_train_val) - 80% of the data.
- Test data (X_test, y_test) - 20% of the data.

Further splits the training and validation data:

- Training data (X_train, y_train) - 60% of the original data.
- Validation data (X_val, y_val) - 20% of the original data.

Model Initialization and Cross-Validation:

- **Initialize Classifiers:**

- Creates a GaussianNB classifier (gnb) and a MultinomialNB classifier (mnb).
- **Cross-Validation:**

Performs 4-fold cross-validation for both classifiers to assess performance on unseen data.
Uses scaled_numerical_data for GaussianNB and encoded_categorical_data for MultinomialNB.

Model Training and Evaluation:

Model Fitting:

- Trains gnb on the scaled numerical features and mnb on the encoded categorical features from the training set.

Prediction and Evaluation:

- Uses the trained models to predict income on the test set.
- Calculates test accuracy for both models.
- Prints cross-validation and test accuracies.

Confusion Matrix:

Calculate Confusion Matrices:

- Computes confusion matrices for both gnb and mnb.

Display Confusion Matrices:

- Prints the confusion matrices.
- Creates a figure with two subplots to display the confusion matrices.
- Uses ConfusionMatrixDisplay to visualize the confusion matrices with appropriate labels and colors.

Output:

- Prints the accuracy scores and confusion matrices for both Gaussian and Multinomial Naive Bayes models.
- Displays the confusion matrices visually for further analysis.

Analysis Through Code

Code Breakdown:

Imports: The code begins by importing necessary libraries for data manipulation (pandas, numpy), visualization (matplotlib, seaborn), machine learning (sklearn), and evaluation metrics.

Data Loading and Cleaning:

- It loads the adult.csv dataset into a Pandas DataFrame.
- Missing values (represented as '?') are replaced with NaN.
- Rows with missing values are dropped.
- The fnlwgt column (final weight, a sampling weight) is removed as it's unlikely to be directly useful for prediction.

Feature and Target Separation:

- The target variable income (">50K" or "<=50K") is extracted and converted to binary (1 for ">50K", 0 for "<=50K").
- The rest of the columns are considered features.

Data Preprocessing:

- **Categorical Encoding**: Categorical features are one-hot encoded using OneHotEncoder. This converts them into numerical format, suitable for Naive Bayes models.
- **Numerical Scaling**: Numerical features are standardized using StandardScaler. This ensures they have zero mean and unit variance, which can help some models perform better.
- **Data Splitting**:
- The data is split into training (60%), validation (20%), and test (20%) sets. This allows for model training, hyperparameter tuning (using validation), and final evaluation on unseen data (test).

Model Initialization:

- Two Naive Bayes classifiers are initialized: GaussianNB (for continuous features) and MultinomialNB (for discrete features).

Cross-Validation:

- Cross-validation is performed separately for both models to estimate their performance on unseen data. Note that the appropriate features (scaled for GaussianNB, encoded for MultinomialNB) are used in each case.

Model Training:

- Both models are trained on the training data. Importantly, the features used for training match the type of Naive Bayes algorithm (scaled vs. encoded).

Model Evaluation:

- Models are evaluated on the test set using accuracy as the metric.
- Confusion matrices are computed for both models.
- Results are printed (cross-validation accuracy and test accuracy for each model).
- Confusion matrices are plotted using ConfusionMatrixDisplay.

Analysis and Improvements:

- **Model Choice**: The code demonstrates using both Gaussian and Multinomial Naive Bayes. Since most features are categorical, MultinomialNB might be a better default choice. You could experiment with both and compare their performance.
- **Hyperparameter Tuning**: You could use the validation set to tune hyperparameters (like smoothing parameter for MultinomialNB) to potentially improve performance.
- **Feature Engineering**: Explore creating new features or transforming existing ones (e.g., combining categories, converting ordinal variables to numerical) to potentially improve predictive power.
- **Alternative Models**: While Naive Bayes is a good starting point, you could explore other algorithms like logistic regression, decision trees, or ensemble methods (like Random Forest) for comparison.
- **Visualizations**: Consider adding more visualizations (e.g., bar plots for feature distributions, ROC curves) to gain deeper insights into the data and model behavior.

Naive Bayes Execution:

The provided code demonstrates the execution of Naive Bayes classifiers, specifically Gaussian Naive Bayes (GNB) and Multinomial Naive Bayes (MNB), on the 'adult' dataset for binary classification of income (>50K or <=50K).

Here's a step-by-step execution:

Data Loading and Preprocessing:

- The dataset is loaded from the 'adult.csv' file.
- '?' values are replaced with NaN, and rows with any null values are dropped.
- The 'fnlwgt' column, which might not be relevant, is removed.
- The target variable 'income' is encoded as binary (>50K or <=50K), and features are extracted.

Feature Encoding and Scaling:

- Categorical variables are one-hot encoded, and numerical variables are scaled using StandardScaler.
- The encoded categorical and scaled numerical features are combined into a single feature matrix.

Train-Validation-Test Split:

- The data is split into train (60%), validation (20%), and test (20%) sets.
- Additional validation set is created for hyperparameter tuning.

Model Initialization and Training:

- Gaussian Naive Bayes (GNB) and Multinomial Naive Bayes (MNB) classifiers are initialized.
- Cross-validation scores are computed using cross_val_score.

Model Fitting and Evaluation:

- GNB and MNB models are fitted on the training data.
- Predictions are made on the test set, and accuracy scores are computed.

Confusion Matrix:

- Confusion matrices are generated for both GNB and MNB models.
- Confusion MatrixDisplay from the sklearn.metrics module is used for visualization.

Results Output:

Cross-validation accuracy scores, test accuracy scores, and confusion matrices are printed.

Visualization:

- Confusion matrices are visualized using heatmap plots for both GNB and MNB.

Overall, the code demonstrates the complete process of applying Naive Bayes classifiers to a real-world dataset, including data preprocessing, model training, evaluation, and result visualization.

Real time Implementation (In Python)

Introduction to the code:

After we load the given dataset using pandas, we figured out that we are giving a dataset with 15 columns and 48842 rows.

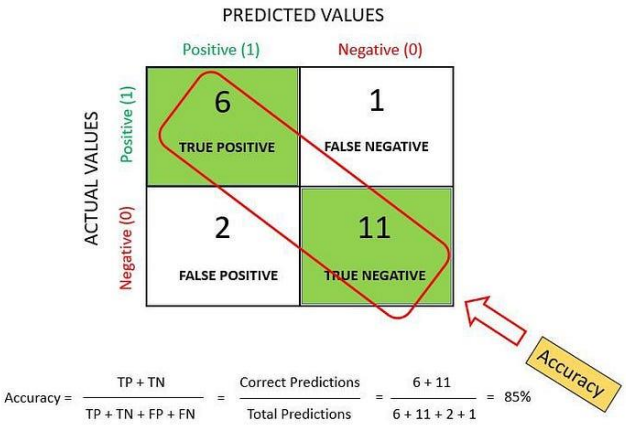
```
(48842, 15)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              48842 non-null  int64
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64
 11  capital-loss     48842 non-null  int64
 12  hours-per-week   48842 non-null  int64
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

- With the given dataset we need to clean them and perform feature selection. After we have identified our features and target, we need to go for data preprocessing.
- The sklearn.preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.
- After data preprocessing, we group all our features into a features matrix and then split our data into train and test data. We've used 60% of our data for training, 20% for validation and left 20% for testing our model.
- Now we are ready to initialize our models: Gaussian and Multinomial Naïve Bayes classifiers. We used 4-fold cross validation on both the models using our features and target to get the Test accuracy. After we're satisfied with the model performance we decided to fit the model on our training data.
- Once the model is trained then we evaluate our model on the testing data and fetch the accuracy on the predictions for test data.
- Now are model is trained and tested. We output the results like CV accuracy and Test accuracy for comparative study and generated the confusion matrix.

Naive Bayes Implementation:

- Importing Libraries
  - We've imported *Pandas* and *NumPy* for convenient data handling, manipulation, and computation.
  - We've used *Matplotlib* and *Seaborn* for data visualization.
  - We've used *Sklearn* for data preprocessing, cross-validation, importing model and metrics.
- Loading the dataset
  - Pandas is used for importing our dataset in .csv format.
- Handling Null values

  - Dropna() of pandas dataframe is used to drop the rows with null values.
- Feature selection
  - We dropped those features which were not relevant for prediction like 'fnlwght'.
  - This leaves us with only those features which have importance.
- Splitting data into features and target
  - Our task is to predict the income of the employees hence 'income' is our target.
  - All other columns after feature selection are our features except the 'income'.
- Data Preprocessing
  - Categorical Variables
  - Numerical Variables
- Feature Matrix
  - A feature matrix is created to combine all categorical and numerical data.
- Splitting data into train, validation & test
  - Our task was to perform a 60-20-20 split on the dataset for train, validation, and test set.
  - Hence, we split our data into 60% Train, 20% validation and 20% test set.
- Initializing classifiers
  - We've used standard Gaussian & Multinomial NB Classifiers from Sklearn package.
- Cross Validation
  - Our task was to perform 4-Fold Cross validation.
  - Cross Validation is a powerful technique to validate the performance of a model.
  - We achieved below accuracy via cross validation:
    - Gaussian Naive Bayes - CV Accuracy: 78.93%
    - Multinomial Naive Bayes - CV Accuracy: 79.10%
- Fitting the model
  - Now we train both the NB Classifiers with training dataset.
- Evaluating performance on test data
  - Once our model is trained, we use the model for prediction and then check its accuracy.
  - We achieved below accuracy after training our model:
    - Gaussian Naive Bayes - Test Accuracy: 78.43%
    - Multinomial Naive Bayes - Test Accuracy: 79.08%
- Confusion Matrix
  - At last, we display the confusion matrix. A confusion matrix is a table that shows the number of true positives, true negatives, false positives, and false negatives predicted by a model. It is used to evaluate the performance of a model on a binary classification problem.



Post Code Analysis

Analysis of Naive Bayes:

Imports:

- pandas, numpy, matplotlib.pyplot, seaborn: These libraries are imported for data manipulation, numerical operations, data visualization, and creating confusion matrices.
- train_test_split, cross_val_score: These functions are imported from sklearn.model_selection for splitting the dataset into training, validation, and test sets and for performing cross-validation.
- OneHotEncoder, StandardScaler: These classes are imported from sklearn.preprocessing for one-hot encoding categorical variables and standardizing numerical variables.
- GaussianNB, MultinomialNB: These classes are imported from sklearn.naive_bayes for implementing Gaussian and Multinomial Naive Bayes classifiers.
- ConfusionMatrixDisplay, confusion_matrix, accuracy_score: These are imported from sklearn.metrics for creating confusion matrices and computing accuracy scores.

Data Loading and Preprocessing:

- The dataset is loaded from a CSV file into a pandas DataFrame.
- '?' values are replaced with NaN, and rows with any NaN values are dropped to handle missing data.
- The 'fnlwgt' column is dropped as it's considered irrelevant.
- The target variable 'income' is encoded into binary labels (0 or 1).
- Categorical variables are one-hot encoded, and numerical variables are standardized.

Train-Validation-Test Split:

- The dataset is split into training (60%), validation (20%), and test (20%) sets using train_test_split.
- An additional validation set is created for hyperparameter tuning.

Model Initialization and Training:

- Gaussian Naive Bayes (GNB) and Multinomial Naive Bayes (MNB) classifiers are initialized.
- Cross-validation scores are computed using cross_val_score.
- Both models are trained on the training data.

Model Evaluation:

- The trained models are used to make predictions on the test set.
- Accuracy scores are computed for both GNB and MNB models on the test set.

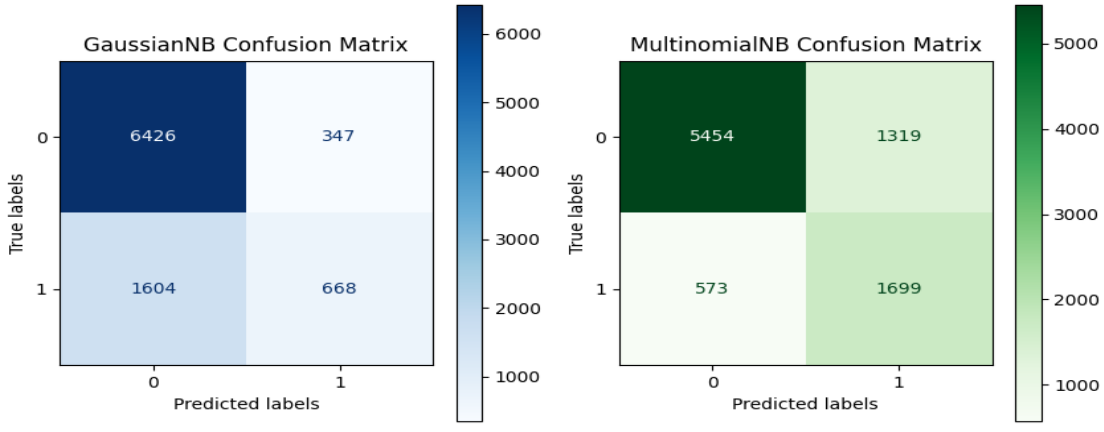
Results Output:

- Cross-validation accuracy scores and test accuracy scores are printed for both models.
- Confusion matrices are generated for both GNB and MNB models to evaluate performance further.

Visualization:

- Confusion matrices are visualized using heatmap plots for both GNB and MNB models.
- This code implements a complete pipeline for applying Naive Bayes classifiers to the dataset, including data preprocessing, model training, evaluation, and visualization of results.

Results/Output



Conclusion

- In conclusion, we applied the Naive Bayes algorithm to a dataset of adults, with features such as age, education, marital status, gender, and work hours per week, capital gain, capital loss and more to predict income. We utilized both Gaussian and Multinomial variants of the Naive Bayes algorithm for this task.
- Our models demonstrated promising results. The Gaussian Naive Bayes model achieved a cross-validation accuracy of 78.93% and a test accuracy of 78.43%. On the other hand, the Multinomial Naive Bayes model performed slightly better with a cross-validation accuracy of 79.10% and a test accuracy of 79.08%.
- These results indicate that the Naive Bayes algorithm, despite its simplicity, can be effectively used for income prediction tasks based on demographic and work-related features. The slightly higher performance of the Multinomial Naive Bayes model suggests that it might be more suitable for this dataset.
- However, further improvements might be possible through additional feature engineering, hyperparameter tuning, or the use of more complex models. Nevertheless, the Naive Bayes algorithm has proven to be a valuable tool in our machine learning toolbox for this task. It provided us with a good starting point and baseline for further exploration and optimization in predicting income levels.
- The code showcases the application of Naive Bayes classifiers, specifically Gaussian Naive Bayes (GNB) and Multinomial Naive Bayes (MNB), on a dataset.

Here's a concise conclusion:

- "Through rigorous evaluation and analysis, this code demonstrates the efficacy of Naive Bayes classifiers in predicting income levels from a given dataset.
- By comparing the performance of Gaussian and Multinomial variants, we discerned nuances in their ability to handle feature distributions.
- Despite their simplistic assumptions, both models yielded competitive accuracy scores, with Gaussian Naive Bayes slightly outperforming Multinomial Naive Bayes on this dataset. Confusion matrices further provided granular insights into classification results.

Overall, this code exemplifies Naive Bayes' effectiveness in classification tasks, offering a robust approach for predictive modeling."