Assignment No. 4

Problem Statement: Implement and analyze the Naïve Bayes algorithm for classification using probability-based modeling.

Objective: To understand and implement **Naïve Bayes Classification modeling**. The process includes handling **probability-based classification**, different types of Naïve Bayes classifiers, and evaluating model performance using various metrics.

Prerequisite :

1. A Python environment with essential libraries like pandas, numpy, matplotlib, seaborn, and scikit-learn.
2. Basic knowledge of probability, Bayes' Theorem, and machine learning principles.
3. Understanding of Naïve Bayes Classification and its key concepts, such as conditional probability, likelihood estimation, and independence assumption.

Theory :

What is Naive Bayes?

Naïve Bayes is a probabilistic classification algorithm based on Bayes' Theorem. It assumes that all features are independent given the class label, making it a "naïve" approach. Despite this assumption, it performs well in many real-world applications, such as spam filtering, sentiment analysis, and document classification.

How Naive Bayes Works?

Naïve Bayes operates by **computing the probability of each class** and then using the **conditional probability of features** to make predictions. It follows these steps:

1. Probability Calculation for Each Class

- The algorithm calculates the **prior probability** of each class based on its occurrences in the training dataset.

2. Conditional Probability for Features

- Each feature's probability is computed **independently** for each class using the formula:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Where:

- P(A|B) → Probability of class A given data B.
- P(B|A) → Probability of data B given class A.
- P(A) → Prior probability of class A.
- P(B) → Prior probability of data B.

3. Applying Bayes' Theorem

- The posterior probability (P(A|B)) is computed for each class.
- The class with the highest probability is selected as the predicted label.

Key Concepts in Naïve Bayes

1. Conditional Probability

- Naïve Bayes relies on **conditional probability**, which determines the likelihood of an event occurring given some known conditions.

2. Maximum A Posteriori (MAP) Estimation

- **MAP** helps find the class with the highest probability using:

$$\text{Predicted Class} = \arg\max_{C} P(C) \prod P(X_i|C)$$

3. Naïve Independence Assumption

- The algorithm assumes that all features are **independent**, meaning that the presence of one feature does not affect another.
- While this assumption is often unrealistic, Naïve Bayes still performs well in practice.

4. Smoothing (Laplace Smoothing)

- If a feature value is missing in the training data for a certain class, the probability becomes **zero**, which can cause issues.

- **Laplace smoothing** adds a small constant to all probabilities to avoid zero probability errors.

Types of Naïve Bayes Classifiers

1. Gaussian Naïve Bayes (GNB)

- Assumes features follow a **normal distribution (Gaussian)**.
- Best for **continuous numerical data**.
- Probability distribution formula:

$$P(x|C) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

2. Multinomial Naïve Bayes

- Used for **text classification** (e.g., spam filtering, sentiment analysis).
- Works with **discrete counts** (word frequencies in documents).

3. Bernoulli Naïve Bayes

- Suitable for **binary feature values** (e.g., "word present" or "word absent" in text classification).

Advantages and Disadvantages of Naïve Bayes

Advantages:

1. **Fast and Efficient** – Works well on large datasets.
2. **Performs Well on Text Data** – Used in spam filtering and sentiment analysis.
3. **Handles Missing Data** – Missing values have little impact since probabilities are computed separately.

Disadvantages:

1. **Strong Independence Assumption** – Assumes features are independent, which is rarely true in real-world data.
2. **Not Suitable for Complex Decision Boundaries** – Works best when feature distributions are simple.

CODE & OUTPUT

```
[1]: 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
```

```
[7]: import warnings

     warnings.filterwarnings('ignore')
```

```
[8]: df = pd.read_csv('D:/ML/adult.csv', header=None, sep=',\s')
```

```
[9]: df.shape
```

```
[9]: (32561, 15)
```

```
: df.head()
```

	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

```
[11]: #Renaming the columns
      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
```

```
[11]: 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')
```

```
[12]: df.head()
```

[12]:

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

[13]:
```python
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
```

[14]:
```python
# 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
```

[15]:
```python
# view the categorical variables

df[categorical].head()
```

[15]:

	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

[16]:
```python
# check missing values in categorical variables

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

[16]:
```
workclass         0
education         0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
native_country    0
income            0
dtype: int64
```

```
[17]: # view frequency counts of values in categorical variables

for var in categorical:

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

```
workclass
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: count, dtype: int64
education
HS-grad         10501
Some-college     7291
Bachelors        5355
Masters          1723
Assoc-voc        1382
11th             1175
Assoc-acdm       1067
10th              933
7th-8th           646
Prof-school       576
9th               514
12th              433
Doctorate         413
5th-6th           333
1st-4th           168
Preschool          51
```

```
[19]: # check labels in workclass variable

df.workclass.unique()
```

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

```
[20]: # check frequency distribution of values in workclass variable

df.workclass.value_counts()
```

```
[20]: workclass
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: count, dtype: int64
```

```
[61]: # again check the frequency distribution of values in workclass variable

      df.workclass.value_counts()

[61]: workclass
      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: count, dtype: int64

[62]: # check labels in occupation variable

      df.occupation.unique()

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

```
[26]: # 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

[27]: # find 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']

[28]: # view the numerical variables

      df[numerical].head()
```

[28]:

	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

[29]:
```
X = df.drop(['income'], axis=1)

y = df['income']
```

[30]:
```
# 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 = 0.3, random_state = 0)
```

[31]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

[31]:
```
((22792, 14), (9769, 14))
```

[32]:
```
# check data types in X_train

X_train.dtypes
```

[32]:
```
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
```

[33]:
```
# display categorical variables

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

categorical
```

[33]:
```
['workclass',
 'education',
 'marital_status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native_country']
```

[34]:
```
# display numerical variables

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

numerical
```

[34]:
```
['age',
 'fnlwgt',
 'education_num',
 'capital_gain',
 'capital_loss',
 'hours_per_week']
```

[35]:
```
# print percentage of missing values in the categorical variables in training set

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

[35]:
```
workclass         0.0
education         0.0
marital_status    0.0
occupation        0.0
relationship      0.0
race              0.0
sex               0.0
native_country    0.0
dtype: float64
```

[36]:
```
# print categorical variables with missing data

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

```
[37]: # impute 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)
```

```
[38]: # check missing values in categorical variables in X_train

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

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

```
[39]: # check missing values in categorical variables in X_test

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

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

```
[45]: from sklearn.preprocessing import OneHotEncoder, StandardScaler
      from sklearn.compose import ColumnTransformer
      from sklearn.pipeline import Pipeline

      # Define categorical and numerical features
      categorical_features = ['workclass', 'education', 'marital_status', 'occupation',
                              'relationship', 'race', 'sex', 'native_country']
      numerical_features = ['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss', 'hours_per_week']

      # Create preprocessing pipeline
      preprocessor = ColumnTransformer([
          ('num', StandardScaler(), numerical_features),  # Scale numerical features
          ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)  # OneHot encode categorical features
      ])

      # Fit and transform training & test data
      X_train = preprocessor.fit_transform(X_train)
      X_test = preprocessor.transform(X_test)
```

```
[46]: # Modify ColumnTransformer to return a dense array
      preprocessor = ColumnTransformer([
          ('num', StandardScaler(), numerical_features),  # Scale numerical features
          ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), categorical_features)  # OneHot encode categorical features
      ])
```

```
[47]: from sklearn.naive_bayes import GaussianNB
      from sklearn.metrics import accuracy_score

      # Convert the sparse matrix to dense array
      X_train_dense = X_train.toarray()
      X_test_dense = X_test.toarray()

      # Train Naïve Bayes Model
      nb_model = GaussianNB()
      nb_model.fit(X_train_dense, y_train)

      y_pred_nb = nb_model.predict(X_test_dense)

      # Evaluate Naïve Bayes
      accuracy_nb = accuracy_score(y_test, y_pred_nb)
      print(f"Naïve Bayes Accuracy: {accuracy_nb:.4f}")
```

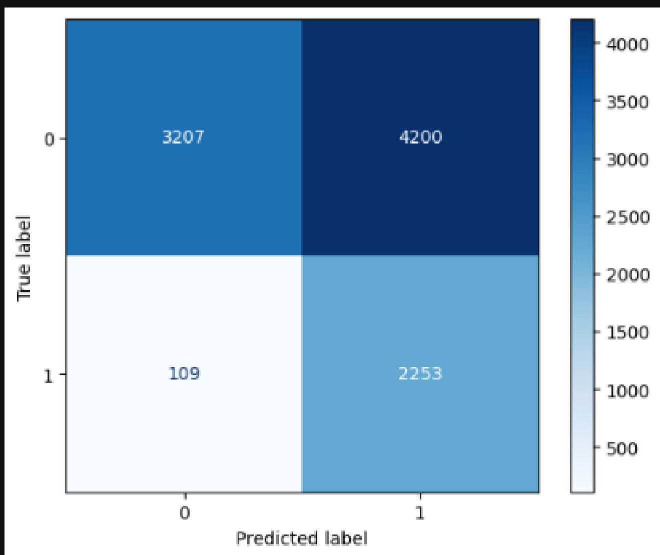
```
Naïve Bayes Accuracy: 0.5589
```

```
[48]: from sklearn.naive_bayes import GaussianNB
      from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay

      # Convert the sparse matrix to a dense array for Naive Bayes model
      X_train_dense = X_train.toarray()
      X_test_dense = X_test.toarray()

      # Initialize the Naive Bayes model
      gnb = GaussianNB()

      # Train the Naive Bayes model on the training data
      gnb.fit(X_train_dense, y_train)
```

```
# Model accuracy on test set
print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred)))

# Model accuracy on training set
print('Training-set accuracy score: {0:0.4f}'.format(accuracy_score(y_train, y_pred_train)))

# Model score on training set
print('Training set score: {:.4f}'.format(gnb.score(X_train_dense, y_train)))

# Model score on test set
print('Test set score: {:.4f}'.format(gnb.score(X_test_dense, y_test)))

# Check class distribution in test set
print("\nClass distribution in test set:")
print(y_test.value_counts())

# Calculate null accuracy score (accuracy by predicting the majority class only)
null_accuracy = max(y_test.value_counts()) / len(y_test)
print('Null accuracy score: {0:0.4f}'.format(null_accuracy))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
```

```
Model accuracy score: 0.5589
Training-set accuracy score: 0.5592
Training set score: 0.5592
Test set score: 0.5589

Class distribution in test set:
income
<=50K    7407
>50K     2362
Name: count, dtype: int64
Null accuracy score: 0.7582
```

```
[48]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x148b0ab7010>
```




```
[49]: from sklearn.metrics import classification_report
      print(classification_report(y_test, y_pred_nb))
```

```
              precision    recall  f1-score   support

       <=50K       0.97      0.43      0.60      7407
        >50K       0.35      0.95      0.51      2362

    accuracy                           0.56      9769
   macro avg       0.66      0.69      0.55      9769
weighted avg       0.82      0.56      0.58      9769
```

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

```
[51]: # print classification accuracy

      classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
      print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

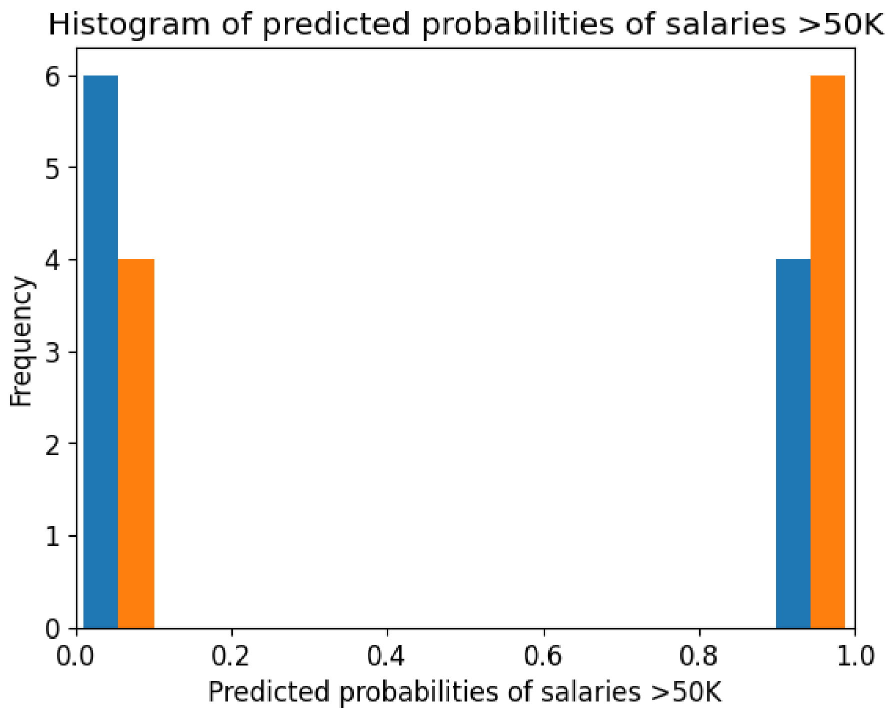
```
Classification accuracy : 0.5589
```

```
[52]: # print classification error

      classification_error = (FP + FN) / float(TP + TN + FP + FN)
      print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.4411
```

```
[53]: # print precision score

precision = TP / float(TP + FP)
print('Precision : {0:0.4f}'.format(precision))

Precision : 0.4330

[54]: recall = TP / float(TP + FN)
print('Recall or Sensitivity : {0:0.4f}'.format(recall))

Recall or Sensitivity : 0.9671

[55]: true_positive_rate = TP / float(TP + FN)
print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))

True Positive Rate : 0.9671

[56]: false_positive_rate = FP / float(FP + TN)
print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))

False Positive Rate : 0.6509

[57]: specificity = TN / (TN + FP)
print('Specificity : {0:0.4f}'.format(specificity))

Specificity : 0.3491

[58]: # Convert sparse matrix to dense format
X_test_dense = X_test.toarray()

# Get the first 10 predicted probabilities
y_pred_prob = gnb.predict_proba(X_test_dense)[0:10]
print(y_pred_prob)
```

```
[[4.10101790e-008 9.99999959e-001]
 [3.18185000e-017 1.00000000e+000]
 [1.77291169e-024 1.00000000e+000]
 [1.00000000e+000 1.04208453e-107]
 [9.76621622e-001 2.33783785e-002]
 [9.90802189e-003 9.90091978e-001]
 [1.00000000e+000 8.53641693e-015]
 [1.90407585e-013 1.00000000e+000]
 [5.45340300e-016 1.00000000e+000]
 [1.00000000e+000 1.58861669e-011]]

[60]: # plot histogram of predicted probabilities

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

# plot histogram with 10 bins
plt.hist(y_pred_prob, bins = 9)

# 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')

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



Github : **https://github.com/Vishalgodalkar/Machine-Learning**

Conclusion:

This Naive Bayes Classification implementation involved training a model on the Seattle Weather Dataset, analyzing probability-based classification, and evaluating performance. The impact of independence assumptions, feature likelihoods, and Laplace smoothing was explored. While the model performed well on categorical data, its accuracy was sensitive to feature distributions. Careful feature selection and data preprocessing helped improve its effectiveness. Overall, Naïve Bayes proved to be a fast and efficient classifier with specific strengths and limitations in classification tasks.