Aim

To preprocess and analyze the U.S. Census dataset using RAPIDS, leveraging GPU acceleration for faster computation and evaluating various machine learning models for predictive accuracy.

Objective

1. Preprocess the U.S. Census dataset using RAPIDS libraries.
2. Encode categorical features and standardize the dataset.
3. Implement and evaluate machine learning models including Logistic Regression, K-Nearest Neighbors, Random Forest, and Support Vector Machine.
4. Compare the performance of different models based on accuracy and confusion matrix metrics.

Summary

This project demonstrates the application of RAPIDS libraries to preprocess and analyze the U.S. Census dataset efficiently using GPU acceleration. By encoding categorical features and standardizing the dataset, we prepare it for machine learning models. Four models—Logistic Regression, K-Nearest Neighbors, Random Forest, and Support Vector Machine—are implemented and evaluated for their predictive performance. The results are compared using accuracy scores and confusion matrices to determine the best-performing model.

Tools and Libraries Used

- RAPIDS (cudf, cuml)
- CuPy
- NumPy
- Matplotlib
- Pickle
- Google Colab

Procedure

1. **Setup Environment**:
    - Clone RAPIDS utilities and install dependencies.

CODE:

```
!git clone https://github.com/rapidsai/rapidsai-csp-utils.git

!python rapidsai-csp-utils/colab/pip-install.py
```

2. **Import Libraries**:
    - Import necessary libraries including RAPIDS (cudf, cuml), CuPy, NumPy, and Matplotlib.

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
import cudf
import cupy as cp
```

3. **Load Dataset**:
    - Load the U.S. Census dataset using cudf.

CODE:

```
census = cudf.read_csv('/content/drive/MyDrive/adult/adult.csv')
```

4. **Data Exploration**:
    - Explore the dataset, check for missing values, and visualize the income distribution.

CODE:

```
census.describe()
census.isnull().sum()
plt.hist(x = census["income"]);
```

5. **Feature Selection**:
    - Separate features and target variable.

CODE:

```
X_census = census.iloc[:, 0:14]
y_census = census.iloc[:, 14]
```

6. **Encode Categorical Features**:
    - Use LabelEncoder to encode categorical features.

CODE:

```
from cuml.preprocessing import LabelEncoder
X_census['workclass'] = LabelEncoder().fit_transform(X_census['workclass'])
# Repeat for other categorical columns
```

7. **One-Hot Encoding**:
    - Apply OneHotEncoder to specific categorical features.

CODE:

```
from cuml.preprocessing import OneHotEncoder

from cuml.compose import ColumnTransformer

onehotencoder_census = ColumnTransformer(transformers=[('OneHot', OneHotEncoder(sparse=False), [1, 3, 5, 6, 7, 8, 9, 13])], remainder='passthrough')

X_census = onehotencoder_census.fit_transform(X_census)
```

8. **Standardization**:
    - Standardize the dataset using StandardScaler.

CODE:

```
from cuml.preprocessing import StandardScaler

scaler_census = StandardScaler()

X_census = scaler_census.fit_transform(X_census)
```

9. **Train-Test Split**:
    - Split the dataset into training and testing sets.

CODE:

```
from cuml.model_selection import train_test_split

X_census_train, X_census_test, y_census_train, y_census_test = train_test_split(X_census, y_census, test_size=0.15, random_state=42)
```

10. **Save Preprocessed Data**:
    - Save the preprocessed data using Pickle.

CODE:

```
import pickle

with open('census.pkl', mode="wb") as f:

  pickle.dump([X_census_train, y_census_train, X_census_test, y_census_test], f)
```

11. **Load Preprocessed Data**:
    - Load the saved preprocessed data.

CODE:

```
with open('census.pkl', mode="rb") as f:

  X_census_train, y_census_train, X_census_test, y_census_test = pickle.load(f)
```

12. **Model Training and Evaluation**:

- Train and evaluate Logistic Regression, K-Nearest Neighbors, Random Forest, and Support Vector Machine models.

CODE:

```
from cuml.linear_model import LogisticRegression

from cuml.neighbors import KNeighborsClassifier

from cuml.ensemble import RandomForestClassifier

from cuml.svm import SVC

from cuml.metrics import accuracy_score, confusion_matrix


# Logistic Regression

logistic_census = LogisticRegression()

logistic_census.fit(X_census_train, y_census_train)

predictions = logistic_census.predict(X_census_test)

print("Logistic Regression Accuracy:", accuracy_score(y_census_test, predictions))


# K-Nearest Neighbors

knn_census = KNeighborsClassifier(n_neighbors=10)

knn_census.fit(X_census_train, y_census_train)

predictions = knn_census.predict(X_census_test)

print("KNN Accuracy:", accuracy_score(y_census_test, predictions))


# Random Forest

random_forest_census = RandomForestClassifier(n_estimators=10, split_criterion="entropy",
random_state=42)

random_forest_census.fit(X_census_train, y_census_train)

predictions = random_forest_census.predict(X_census_test)

print("Random Forest Accuracy:", accuracy_score(y_census_test, predictions))


# Support Vector Machine
```

```
svm_census = SVC(kernel="linear", random_state=1)

svm_census.fit(X_census_train, y_census_train)

predictions = svm_census.predict(X_census_test)

print("SVM Accuracy:", accuracy_score(y_census_test, predictions))
```

Highlights

1. **GPU Acceleration**: Leveraged RAPIDS libraries for efficient GPU-accelerated data processing.
2. **Comprehensive Preprocessing**: Included encoding and standardization of features to ensure model readiness.
3. **Model Comparison**: Implemented and compared multiple machine learning models to determine the best performer.
4. **Data Persistence**: Used Pickle for saving and loading preprocessed data efficiently.

Conclusion

This project effectively demonstrates the use of RAPIDS for preprocessing and analyzing large datasets with GPU acceleration, significantly improving computational efficiency. By comparing different machine learning models, we identified the best-performing model for the U.S. Census dataset based on accuracy and confusion matrix metrics, highlighting the potential of RAPIDS in data science workflows.