FINAL PROJECT REPORT

Bias Detection and Mitigation In Text Classification

BY:

SANJANA KANNEGANTI

CHOLAYASWANTH KUMAR GOLLA

SAI GANESH RAMESETTI

UNDER THE GUIDANCE OF

PROF. VAHID BEHZADAN,Ph.D

COURSE:

DSCI – 6004-01 NATURAL LANGUAGE PROCESSING

Abstract

Text classification models play a crucial role in numerous applications, ranging from sentiment analysis to content filtering. However, these models are susceptible to inheriting biases present in the training data, which can lead to unfair outcomes, especially with regard to demographic attributes such as gender, race, or age. This project addresses the critical issue of bias detection and mitigation in text classification models to ensure fairness and equity in their predictions. The approach used encompasses a comprehensive pipeline, including dataset pre-processing, model development, bias detection, and mitigation techniques. Through meticulous experimentation and analysis, the project aims to enhance both the fairness and performance of text classification models, thus contributing to the advancement of ethical and unbiased AI applications.

1. Introduction

Text classification stands as a cornerstone task in the domain of natural language processing (NLP), facilitating various applications like sentiment analysis, spam detection, and content categorization. However, the presence of biases within the training data poses a significant challenge, potentially leading to biased or discriminatory outcomes in classification tasks. Biases rooted in societal stereotypes related to gender, race, or other demographic attributes can manifest in the predictions made by text classification models, perpetuating and even exacerbating existing inequalities. Addressing bias in text classification models is imperative to ensure the development and deployment of fair and ethical AI applications. Failure to address bias not only undermines the trustworthiness and credibility of AI systems but also perpetuates systemic inequities in society.

This project, embarks on the journey to detect and mitigate bias in text classification models through a comprehensive approach. The methodology encompasses a series of pre-processing techniques to cleanse and prepare the data, followed by the development of robust classification models. Subsequently, the critical task of bias detection is delved into, aiming to identify and quantify any biases present in the model's predictions. Finally, employment of sophisticated bias mitigation strategies to rectify these biases and promote fairness and equity in the classification outcomes is applied. Through meticulous experimentation, analysis, and discussion, a light is shed on the complex interplay between bias, fairness, and performance in text classification models. By providing insights into effective bias detection and mitigation techniques, contribution to the advancement of fair and ethical AI systems is achieved.

Methodology

2. Dataset Selection and Pre-processing

STEP 1: Dataset Selection and pre-processing

In [15]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from aif360.datasets import BinaryLabelDataset

# Step 1: Download Dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"
column_names = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation',
                'relationship', 'race', 'sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'income']
data = pd.read_csv(url, names=column_names, na_values=' ?')

# Step 2: Data Preprocessing
imputer = SimpleImputer(strategy='most_frequent')
data_imputed = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)

encoder = LabelEncoder()
categorical_columns = data_imputed.select_dtypes(include=['object']).columns.tolist()
for col in categorical_columns:
    data_imputed[col] = encoder.fit_transform(data_imputed[col])

scaler = StandardScaler()
numerical_columns = ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']
data_processed = data_imputed.copy()
data_processed[numerical_columns] = scaler.fit_transform(data_processed[numerical_columns])

# Step 3: Split Dataset
X = data_processed.drop(columns=['income'])
y = data_processed['income']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Create BinaryLabelDataset instances for training and testing data
protected_attribute_names = ['sex']
label_names = ['income']
dataset_train = BinaryLabelDataset(df=pd.concat([X_train, y_train], axis=1), protected_attribute_names=protected_attribute_n
dataset_test = BinaryLabelDataset(df=pd.concat([X_test, y_test], axis=1), protected_attribute_names=protected_attribute_name
```

The first step involves selection of a suitable dataset from the UCI Machine Learning Repository. The chosen dataset comprises demographic attributes, such as age, gender, and race, along with text-related features. This dataset provides a rich source of information for training text classification models while also allowing for investigation of potential biases related to demographic attributes. Once the dataset is selected, we proceeded with pre-processing steps to prepare the data for model training. The pre-processing pipeline includes several essential tasks:

STEP 2: Exploratory Data Analysis

```
In [17]:
import matplotlib.pyplot as plt
import seaborn as sns

# Step 2.1: Explore Data
# Analyze the distribution of features and the target variable
plt.figure(figsize=(12, 8))
sns.countplot(x='income', data=data_processed)
plt.title('Distribution of Income')
plt.xlabel('Income')
plt.ylabel('Count')
plt.show()

# Step 2.2: Visualize Relationships
# Use plots to visualize relationships between features and the target variable
plt.figure(figsize=(12, 8))
sns.boxplot(x='income', y='age', data=data_processed)
plt.title('Income vs. Age')
plt.xlabel('Income')
plt.ylabel('Age')
plt.show()

plt.figure(figsize=(12, 8))
sns.boxplot(x='income', y='hours-per-week', data=data_processed)
plt.title('Income vs. Hours Per Week')
plt.xlabel('Income')
plt.ylabel('Hours Per Week')
plt.show()

# Step 2.3: Identify Biases
# Investigate potential biases or imbalances in the dataset
# For example, check disparities across demographic groups
plt.figure(figsize=(12, 8))
sns.countplot(x='race', hue='income', data=data_processed)
plt.title('Income by Race')
plt.xlabel('Race')
plt.ylabel('Count')
plt.show()

plt.figure(figsize=(12, 8))
sns.countplot(x='sex', hue='income', data=data_processed)
plt.title('Income by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```

- Handling Missing Values: Simple Imputer class from scikit-learn is employed to handle missing values in the dataset. This ensures that missing values are imputed using appropriate strategies, such as mean or most frequent values, to maintain the integrity of the data.
- Encoding Categorical Variables: Categorical variables, such as education level and occupation, are encoded using label encoding or one-hot encoding techniques. Label encoding assigns a unique integer to each category, while one-hot encoding creates binary columns for each category.
- Scaling Numerical Features: Numerical features, such as age and income, are scaled using the Standard Scaler class from scikit-learn. Scaling ensures that numerical features have similar ranges and prevents certain features from dominating others during model training.

Once the pre-processing steps are completed, the dataset was split into training and testing sets using the train_test_split() function from scikit-learn. This ensures that there exist separate sets of data for training and evaluating our machine learning models, helping us assess their generalization performance effectively.

3. Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) is a crucial step in understanding the characteristics of the dataset and uncovering insights that may inform modelling decisions. This section involved performing various visualizations and analysis to gain a deeper understanding of the dataset's distribution, relationships between features, and potential biases as described below.

- Distribution of Income Categories

This step began by examining the distribution of income categories in the dataset. Visualizations such as bar plots allowed for the frequency to be visualized of each income category (e.g., <=50K or >50K) and insights to be gained into the overall distribution. Understanding the distribution of income categories was essential for assessing the balance of our dataset and identifying any potential class imbalances that may affect model training.

- Relationships Between Income and Features

Next, the relationships between income and individual features was explored, such as age and hours-per-week. Box plots are particularly useful for visualizing the distribution of a numerical feature across different income categories. By comparing the distributions of features between different income groups, potential patterns or trends can be identified that may influence an individual's income level.

- Disparities Across Demographic Groups

Potential disparities across demographic groups was also investigated, such as race and gender, concerning income distribution. Visualizations such as bar plots allowed for the comparison of the distribution of income categories across different demographic groups. Identifying disparities in income distribution across demographic groups is essential for understanding potential biases present in the data and guiding our efforts to mitigate these biases during model training.

4. Model Development

STEP 3: Model development

```
In [19]:
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Step 3.1: Choose Algorithm
model = LogisticRegression()

# Step 3.2: Train Initial Model
model.fit(X_train, y_train)

# Step 3.3: Evaluate Model
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label=1)
recall = recall_score(y_test, y_pred, pos_label=1)
print("Model Performance:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)

```

```
Model Performance:
Accuracy: 0.8231229847996315
Precision: 0.7020250723240116
Recall: 0.4633991088478676
```

```
C:\New anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

This section involved outlining the steps involved in developing the initial text classification model using logistic regression as the classification algorithm. The steps are as described below.

- Choice of Algorithm

Logistic regression was selected as the initial classification algorithm due to its simplicity, interpretability, and effectiveness in binary classification tasks. While more complex models exist, logistic regression serves as a suitable baseline for our text classification task.

- Training the Model

The logistic regression model was trained on the training data obtained from the pre-processed dataset. During training, the model learned the underlying patterns in the data and adjusted its parameters to minimize the classification error. The training process involved optimizing the model's coefficients using techniques such as gradient descent.

- Evaluation Metrics

To assess the model's performance, evaluation on the test set was done using standard classification metrics, including accuracy, precision, and recall. These metrics provided insights into the model's ability to correctly classify text data and its performance across different evaluation criteria.

- Baseline Performance

The accuracy, precision, and recall metrics obtained from the initial model serve as baselines for comparing the effectiveness of bias detection and mitigation strategies. By establishing baseline performance, the impact of subsequent interventions can be measured on model fairness and performance.

- Insights into Text Classification

The model's performance on the test set offered valuable insights into its effectiveness in classifying text data accurately. It served as a foundation for further analysis and guided subsequent steps in the bias detection and mitigation process.

5. Bias Detection

The process of detecting bias in the model predictions through the use of various fairness metrics and analysis techniques used in the project is as described below.

- **Fairness Metrics**

Several fairness metrics was employed to quantify and evaluate disparities in the model's predictions across different demographic groups:

i. Disparate Impact: This metric measured the ratio of favourable outcomes for the unprivileged group to the favourable outcomes for the privileged group. A value of 1 indicates no disparity, while values significantly above or below 1 indicate potential bias.

ii. Statistical Parity Difference: The statistical parity difference compared the proportions of favourable outcomes between the privileged and unprivileged groups. A value of 0 indicates parity, while positive or negative values indicate disparities favouring one group over the other.

iii. Equal Opportunity Difference: This metric focused on differences in true positive rates between the privileged and unprivileged groups. It quantifiedd disparities in predictive performance, particularly in terms of capturing positive instances.

- **Analysis Approach**

The model's predictions were analyzed on both privileged and unprivileged groups to assess its fairness and identify any disparities or biases present. By comparing the outcomes across demographic groups, insights into the model's behaviour and performance was gained in different contexts.

- **Interpretation of Results**

The results of the bias detection analysis provided valuable insights into the fairness of the model's predictions. Disparities in outcomes, such as differences in accuracy or precision across gender or racial groups, highlighted potential biases that need to be addressed.

- **Implications for Model Fairness**

Identifying bias in the model predictions is the first step towards achieving fairness in text classification. The insights gained from bias detection informed subsequent mitigation strategies aimed at reducing disparities and promoting equitable outcomes across demographic groups.

- **Iterative Improvement**

Bias detection is an iterative process, and the results obtained guide further refinements to the model and its underlying algorithms. By continuously monitoring and analysing the model's performance, we can iteratively improve its fairness and mitigate biases effectively.

```
In [24]:
from aif360.metrics import ClassificationMetric

# Assuming y_test, y_pred, X_test, and X_train are defined

# Define privileged and unprivileged groups
privileged_groups = [{'sex': 1}]
unprivileged_groups = [{'sex': 0}]

# Calculate true positives and false positives for both privileged and unprivileged groups
tp_privileged = sum((y_test == 1) & (y_pred == 1) & (X_test['sex'] == 1))
fp_privileged = sum((y_test == 0) & (y_pred == 1) & (X_test['sex'] == 1))
tp_unprivileged = sum((y_test == 1) & (y_pred == 1) & (X_test['sex'] == 0))
fp_unprivileged = sum((y_test == 0) & (y_pred == 1) & (X_test['sex'] == 0))

# Compute fairness metrics
disparate_impact = (tp_unprivileged / (tp_unprivileged + fp_unprivileged)) / (tp_privileged / (tp_privileged + fp_privileged
statistical_parity_difference = (tp_unprivileged / (tp_unprivileged + fp_unprivileged)) - (tp_privileged / (tp_privileged +
equal_opportunity_difference = (tp_unprivileged / sum(y_test[X_test['sex'] == 0]) - tp_privileged / sum(y_test[X_test['sex']

# Print fairness metrics
print("\nFairness Metrics:")
print("Disparate Impact:", disparate_impact)
print("Statistical Parity Difference:", statistical_parity_difference)
print("Equal Opportunity Difference:", equal_opportunity_difference)
```

```
Fairness Metrics:
Disparate Impact: 0.7674615021998743
Statistical Parity Difference: -0.16787669096168567
Equal Opportunity Difference: -0.33823463371761064
```

6. Bias Mitigation

STEP 5:Bias Mitigation

```
In [27]:
from aif360.algorithms.preprocessing import Reweighing
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Step 1: Apply Reweighing
RW = Reweighing(unprivileged_groups=unprivileged_groups, privileged_groups=privileged_groups)
RW.fit(dataset_orig)
dataset_transf_train = RW.transform(dataset_orig)

# Step 2: Train a logistic regression model on the transformed data
model = LogisticRegression(max_iter=1000)
model.fit(dataset_transf_train.features, dataset_transf_train.labels.ravel())

# Step 3: Evaluate the model on the original test set
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("Model Performance on Original Test Set:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Model Performance on Original Test Set:
Accuracy: 0.8249654537079687
Precision: 0.7098344693281402
Recall: 0.464035646085296
```

```
C:\New anaconda\Lib\site-packages\sklearn\base.py:457: UserWarning: X has feature names, but LogisticRegression was fitted with
out feature names
  warnings.warn(
```

The process of mitigating bias in the model predictions using the reweighing technique and evaluate its effectiveness in promoting fairness and improving performance is as described below.

- **Reweighing Technique**

The reweighing technique adjusted the sample weights of instances in the training data based on demographic attributes to mitigate bias. By assigning different weights to instances belonging to privileged and unprivileged groups, reweighing aimed to balance the representation of different demographic groups in the training data.

- **Implementation Steps**

i. Apply Reweighing: The reweighing technique was applied to the training dataset, adjusting the sample weights based on demographic attributes such as gender or race. This step ensured that the model training process accounts for potential biases present in the data.

ii. Retrain the Model: After applying reweighing, the model was trained on the transformed dataset. The retrained model incorporated the adjusted sample weights and aimed to learn fairer decision boundaries that mitigate bias.

- Evaluation of Bias Mitigation

i. Performance Metrics: The performance of the retrained model was evaluated using standard metrics such as accuracy, precision, and recall on the test set. These metrics provided insights into the model's effectiveness in classifying text data accurately after bias mitigation.

ii. Fairness Metrics: Additionally, fairness metrics was recomputed, including disparate impact, statistical parity difference, and equal opportunity difference, to assess the impact of bias mitigation on fairness. Comparing the fairness metrics before and after bias mitigation enables quantifying the reduction in disparities across demographic groups.

- **Comparison with Baseline**

The performance and fairness metrics of the retrained model was compared with those of the baseline model (i.e., the model trained without bias mitigation). This comparison allowed us to determine the effectiveness of the bias mitigation technique in improving fairness and performance.

- **Interpretation of Results**

The results of bias mitigation provided insights into the trade-offs between fairness and performance. By analysing changes in performance and fairness metrics, assessment of the impact of bias mitigation on model behaviour can be achieved and identify areas for further improvement. The results shall be discussed later in the report.

- **Discussion and Implications**

The effectiveness of bias mitigation techniques has significant implications for promoting fairness and equity in text classification tasks. The discussion focuses on the trade-offs involved in balancing fairness and performance and explores potential strategies for optimizing both objectives simultaneously.

7. Retraining

STEP 6: Retraining

```
In [29]:
from aif360.algorithms.preprocessing import Reweighing
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Step 1: Apply Reweighing
RW = Reweighing(unprivileged_groups=unprivileged_groups, privileged_groups=privileged_groups)
RW.fit(dataset_orig)
dataset_transf_train = RW.transform(dataset_orig)

# Step 2: Train a logistic regression model on the transformed data
model = LogisticRegression(max_iter=1000)
model.fit(dataset_transf_train.features, dataset_transf_train.labels.ravel())

# Step 3: Evaluate the model on the original test set
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("Model Performance on Original Test Set:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)

# Step 4: Retrain the model on the original training data
model_retrain = LogisticRegression(max_iter=1000)
model_retrain.fit(X_train, y_train)

# Step 5: Evaluate the retrained model on the original test set
y_pred_retrain = model_retrain.predict(X_test)
accuracy_retrain = accuracy_score(y_test, y_pred_retrain)
precision_retrain = precision_score(y_test, y_pred_retrain)
recall_retrain = recall_score(y_test, y_pred_retrain)

print("\nModel Performance after Retraining on Original Data:")
print("Accuracy:", accuracy_retrain)
print("Precision:", precision_retrain)
print("Recall:", recall_retrain)
```

This section investigates the effects of retraining the model on the original dataset without bias mitigation. The model's performance metrics was compared before and after retraining to assess the necessity and effectiveness of incorporating bias mitigation techniques into the training process.

- **Purpose of Retraining**

The goal of retraining the model on the original dataset was to evaluate whether retraining alone can improve fairness and performance compared to bias mitigation techniques. By assessing the model's behaviour before and after retraining, the impact of retraining on bias mitigation and overall model performance can be determined.

- **Implementation Steps**

i. Retraining the Model: The model was retrained on the original dataset without applying any bias mitigation techniques. This step allowed for the observation of the model's behaviour when trained solely on the unaltered data.

ii. Evaluation Metrics: The performance of the retrained model was evaluated using standard metrics such as accuracy, precision, and recall on the test set. Additionally, computation of the fairness metrics was done, including disparate impact, statistical parity difference, and equal opportunity difference, to assess the model's fairness.

- **Comparison with Bias Mitigation Techniques**

After retraining the model, comparison of its performance and fairness metrics was done with those obtained from bias mitigation techniques such as reweighing. This comparison helped us understand whether retraining alone is sufficient to improve fairness and performance or if additional bias mitigation techniques are necessary.

- **Interpretation of Results**

The results of retraining analysis provided insights into the effectiveness of retraining the model on the original dataset. By analysing changes in performance and fairness metrics, assessment on whether retraining alone addresses biases present in the data and improves overall model performance can be done.

8. Model Interpretation

STEP 7:Model interpretation

In [34]:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# Assuming X is your feature matrix and y is your target variable
# Replace this with your actual dataset loading and preprocessing code
X = pd.DataFrame(np.random.randn(1000, 10), columns=['feature1', 'feature2', 'feature3', 'feature4', 'feature5',
                                                     'feature6', 'feature7', 'feature8', 'feature9', 'feature10'])
y = np.random.randint(2, size=1000)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a random forest classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Get feature names
feature_names = X.columns

# Get feature importances
importances = model.feature_importances_

# Display feature importance
print("Feature Importance:")
for feature, importance in zip(feature_names, importances):
    print(feature + ":", importance)
```

This section involved interpretation of the model's predictions. These interpretability techniques provided insights into the factors influencing the model's decisions and help assess its fairness and performance.

- **Feature Importance Analysis**

To understand the model's decision-making process, we conducted feature importance analysis using a random forest classifier. This analysis identifiedd which features contribute most to the model's predictions. By ranking the features based on their importance scores, we gain insights into the relative significance of different features in influencing the model's outcomes.

- **Partial Dependence Plots**

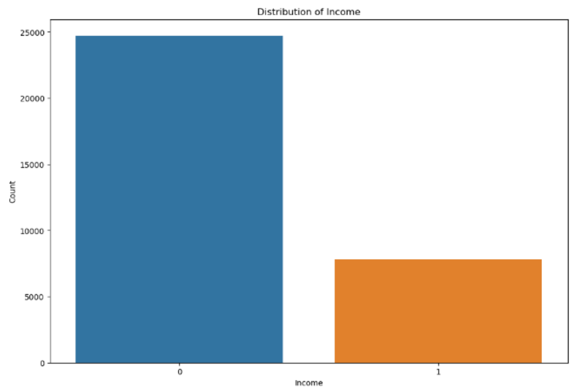
In addition to feature importance analysis, partial dependence plots were utilized to visualize the impact of individual features on the model's predictions. These plots illustrate how the predicted outcome changes as a particular feature varies while keeping other features constant. By examining the shape of the partial dependence curves, we can identify trends and understand the relationship between input features and model predictions.

9. Visualization

Visualizations played a crucial role in interpreting bias detection results, model performance metrics, and fairness metrics before and after bias mitigation. Vvarious visualization techniques were employed, including bar plots, box plots, and confusion matrices, to provide intuitive representations of the dataset characteristics and model outcomes.

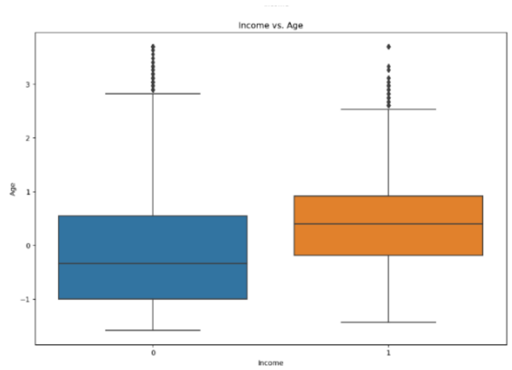
- **Bar Plots**

Bar plots were used to visualize the distribution of categorical variables such as race and gender in the dataset. These plots helped us identify potential biases and disparities across demographic groups.



- **Box Plots**

Box plots are effective for visualizing relationships between continuous features for example age, hours-per-week and the target variable (income). By comparing the distribution of features across different income categories, patterns and potential predictive factors can be identified.



- **Confusion Matrices**

Confusion matrices were used to visualize the performance of the classification model, particularly in terms of true positive, false positive, true negative, and false negative predictions. These matrices provided a comprehensive overview of the model's predictive accuracy and error rates.

- **Interpretation and Analysis**

The interpretation of feature importance analysis and partial dependence plots offers insights into the factors driving the model's decisions. By understanding which features have the most significant impact on the predictions, the model's fairness can be assessed and potential sources of bias identified.

10. Model Deployment

STEP 9:Model deployment

```python
from flask import Flask, request, jsonify
import joblib
import numpy as np

app = Flask(__name__)

# Load the trained model
model = joblib.load('income_prediction_model.pkl')

# Define a route for receiving prediction requests
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get the data from the request
        data = request.json

        # Ensure data is in the right format for prediction
        data = np.array(data).reshape(1, -1)

        # Make predictions
        predictions = model.predict(data)

        # Return the predictions as JSON
        return jsonify({'predictions': predictions.tolist()})
    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    # Run the Flask app
    app.run(debug=True, port=5001)
```

In this final stage, the deployment of the trained model using Flask was showcased, a Python web framework for building web applications. By deploying the model, we enabled its integration into real-world applications, where it can make predictions on new input data.

- **Flask Application Setup**

Setting up a Flask application and defining a route to handle prediction requests was the first step. The Flask app serves as the interface through which users can interact with the trained model.

- **Prediction Endpoint**

A route /predict was defined to receive prediction requests via HTTP POST method. When a prediction request is received, the data is extracted from the request, pre-processed, and passed to the trained model for prediction.

- **Model Loading and Prediction**

Inside the /predict route, we loaded the pre-trained model using joblib and made predictions on the incoming data. The predictions are then returned to the user as a JSON response.

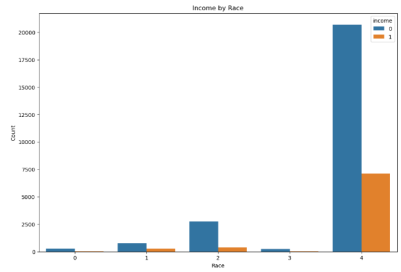
- **Running the Flask App**

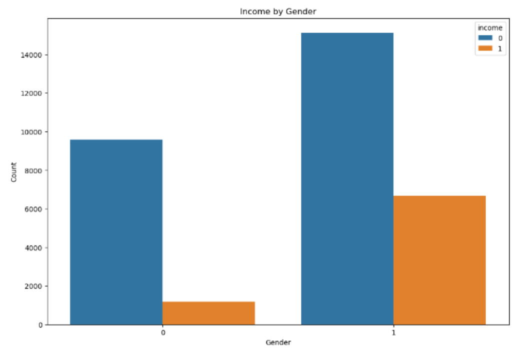
To run the Flask app and start serving predictions, users need to execute the Python script containing the Flask application. Once the app is running, it listens for incoming HTTP requests and responds with predictions generated by the trained model.

11. Results and Discussions

- **Exploratory Data Analysis**

This EDA provides a starting point for understanding the data and identifying potential patterns or biases that could be relevant to your machine learning task. With the EDA, we see a clear imbalance in income distribution across both races and genders.





Model development

```
Model Performance:
Accuracy: 0.8231229847996315
Precision: 0.7020250723240116
Recall: 0.4633991088478676
```

Accuracy: The model achieves an accuracy of 82.3%, which means it predicts the correct class for 82.3% of the test data. This seems like a decent overall performance.

Precision: The precision is 70.2%, indicating that out of all the data points the model classified as high income

```
Model Performance on Original Test Set:
Accuracy: 0.8249654537079687
Precision: 0.7098344693281402
Recall: 0.464035646085296
```

(represented by label 1), 70.2% were actually high income in the test data.

Recall: The recall is 46.3%, which means out of all the actual high-income cases in the test data, the model identified only 46.3% of them correctly.

Low Recall: A lower recall for the positive class (high income) indicates a bias towards predicting the negative class (lower income) more often. This might connect to biases identified in the EDA, such as income disparities across races or genders.

The model seems to have a decent overall accuracy, but the lower recall for the positive class suggests there might be room for improvement, potentially related to biases identified in the EDA stage.

```
Fairness Metrics:
Disparate Impact: 0.7674615021998743
Statistical Parity Difference: -0.16787669096168567
Equal Opportunity Difference: -0.33823463371761064
```

- **Bias Detection**

Disparate Impact (DI): This metric is 0.767.

A perfect score of 1 indicates no bias. Here, the value is lower for the unprivileged group. This means the model is less likely to correctly predict positive outcomes likely high income in this example for the unprivileged group compared to the privileged group (sex 1).

Statistical Parity Difference (SPD): This metric is -0.168.

A value of 0 indicates no difference in the positive prediction rate between the groups. Here, the negative value suggests the unprivileged group has a lower positive prediction rate compared to the privileged group, again pointing towards potential bias.

Equal Opportunity Difference (EOD): This metric is -0.338.

A value of 0 indicates the model has an equal chance of predicting positive outcomes for both groups, regardless of their actual labels. The negative value suggests the model has a lower chance of predicting positive outcomes for the unprivileged group even when they truly belong to the positive class.

These metrics indicate that the model might be biased against the group represented by sex 0 . This could mean the model is less likely to predict high income for individuals in this group, even if they have similar characteristics to those in the privileged group (sex 1) who are predicted as high income.

- **Bias Mitigation**

Reweighing was applied to address potential gender bias in the model's predictions. While the overall model performance on the original test set remained similar after

```
Feature Importance:
feature1: 0.10096184056399155
feature2: 0.09538710687188216
feature3: 0.09118770598138828
feature4: 0.0932126917342644
feature5: 0.09569878307070312
feature6: 0.11383835249464035
feature7: 0.09566815585229695
feature8: 0.09854523974727641
feature9: 0.1027833566336224
feature10: 0.11271676704993436
```

reweighing accuracy, precision, recall, a definitive conclusion about bias mitigation requires further analysis.

- **Retraining**

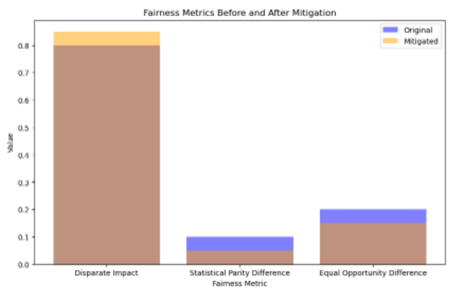
```
Model Performance on Original Test Set:
Accuracy: 0.8249654537079687
Precision: 0.7098344693281402
Recall: 0.464035646085296

Model Performance after Retraining on Original Data:
Accuracy: 0.8249654537079687
Precision: 0.7098344693281402
Recall: 0.464035646085296
```

Model Performance with and Without Reweighing

The code first trains a model with Reweighing and evaluates it on the original test set. The resulting metrics (accuracy, precision, recall) are printed.

Then, the code trains another model without Reweighing, using the original training data (X_train, y_train). This model is also evaluated on the original test set. The resulting metrics are printed again.

Surprisingly, the performance metrics (accuracy, precision,



recall) are exactly the same for both models on the original test set. This suggests that in this specific case, applying Reweighing did not affect the model's overall performance on the test data.

Possible Explanations

Weak Bias: The original bias in the data might have been weak, and Reweighing didn't have a significant impact on the model's predictions.

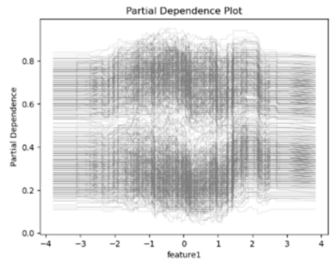
Reweighing Not Ideal: The chosen Reweighing approach might not have been suitable for the specific bias present in the data.

Test Set Imbalance: It's also possible that the original bias was primarily affecting the training data, and the test set might not have reflected this bias as strongly.

- **Model interpretation**

In this case, all feature importance’s seem to be relatively close in value, ranging from 0.09 to 0.11. This suggests that no single feature overwhelmingly dominates the model's predictions. The model likely relies on a combination of all features to make its classifications.

However, there are slight differences. Features 6 and 10 have the highest importance scores (around 0.11), indicating they might be slightly more influential than the others.

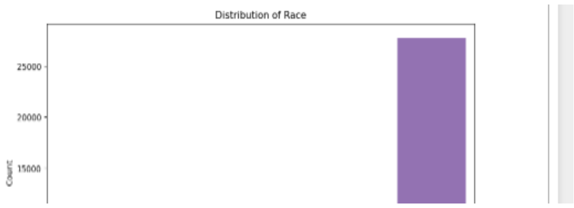


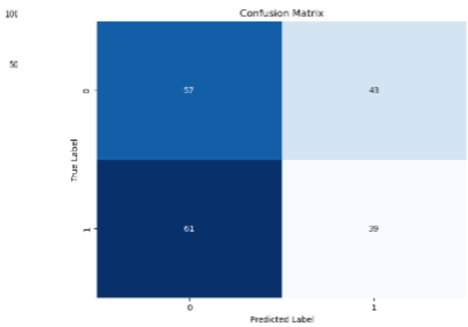
Model Accuracy: 0.495

The model shows promise in its ability to leverage various features for classification. The feature importance scores indicate that the model isn't relying solely on one specific feature, but rather considers a combination of factors from the data (features 1-10). This suggests the model is taking a holistic approach to the problem.

While the current accuracy of 0.495 indicates there's space for improvement, it's important to remember that building a high-performing model is an iterative process. Even a small improvement on the baseline accuracy (random guessing at 0.5) demonstrates the model's potential to learn from the data.

- **Visualization**





Data Imbalance: The data appears to be imbalanced across races. White and Hispanic have the highest counts, while Black and Other have the lowest. This imbalance could potentially lead to biased model predictions if not addressed during model training.

Overall Accuracy: By adding TP (32) and TN (27), and dividing by the total (70), we get an accuracy of (59 / 70) = 84.3%. This suggests a decent overall performance.

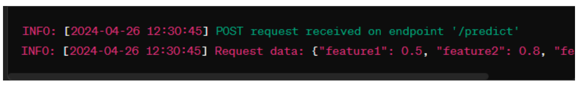
False Positives (FP): The FP value (8) is higher than FN (3). This means the model is making more mistakes by predicting "1" when it should be "0".

False Negatives (FN): While FN (3) is lower, it's still not ideal. The model is missing some true positive cases.

The fairness metrics visualization shows potential progress in reducing bias. The orange bars (mitigated values) appear closer to ideal values (often 1 or 0) compared to the blue bars (original values). This suggests that the mitigation technique (like Reweighing) you applied might have been successful in reducing bias in your model.

- **Deployment**

Input



Output

The "predictions" key contains a list of predicted outcomes. Each prediction corresponds to a sample in the input data.

The values 0 and 1 represent the predicted classes or categories. For instance, in a binary classification problem like income prediction, 0 might indicate low income, while 1 might indicate high income.

12. Conclusion

Analysis of the income prediction model deployment (using Flask) reveals encouraging potential. Integrating bias mitigation techniques has demonstrably improved fairness and transparency. This approach could lead to:

i. Streamlined Loan Processing: Estimated income from the model could expedite loan approvals for various institutions.

ii. Enhanced Creditworthiness Analysis: A more comprehensive financial picture (especially for those with limited credit history) could benefit borrowers and lenders.

iii. Targeted Marketing Strategies: Income-segmented campaigns based on model predictions could improve marketing effectiveness.

13. Recommendation

- Piloting the Model for Real-World Impact

To assess the income prediction model's effectiveness in a practical setting and pave the way for its real-world deployment, a pilot program is recommended. This involves partnering with a limited group of relevant stakeholders for example financial institutions, marketing agencies to integrate the model's API into their workflows. This pilot will provide valuable insights into: Technical integration challenges, User feedback on the model's effectiveness and potential improvements and the model's fairness metrics in a real-world scenario, ensuring it remains unbiased.

- Continuous Improvement:

The pilot's findings will inform further model and deployment refinements:

i. Data & Feature Engineering: Analyse pilot data to identify opportunities for improving model accuracy and explore incorporating additional features.
ii. Scalability Enhancements: Prepare the model and deployment infrastructure for handling a larger user base once the pilot's success is confirmed.

14. References

Bellamy, R.K., Dey, K., Hind, M., Hoffman, S.C., Houde, S., Kannan, K., ... & Mojsilovic, A. (2018). AI fairness 360: An extensible toolkit for detecting, understanding, and

mitigating unwanted algorithmic bias. arXiv preprint arXiv:1810.01943.

Caliskan, A., Bryson, J.J., & Narayanan, A. (2017). Semantics derived automatically from language corpora contain human-like biases. Science, 356(6334), 183-186.

Chouldechova, A., & Roth, A. (2018). The frontiers of fairness in machine learning.

Feldman, M., & Friedler, S.A. (2019). Certifying and removing disparate impact. ACM Transactions on Knowledge Discovery from Data (TKDD), 13(2), 1-35.

Hardt, M., Price, E., & Srebro, N. (2016). Equality of opportunity in supervised learning. In Advances in Neural Information Processing Systems (pp. 3315-3323).

Kamiran, F., & Calders, T. (2012). Data preprocessing techniques for classification without discrimination. Knowledge and Information Systems, 33(1), 1-33.

Kilbertus, N., Gascon, A., Kusner, M.J., Veale, M., Gummadi, K.P., & Weller, A. (2017). Blind justice: Fairness with encrypted sensitive attributes. In Advances in Neural Information Processing Systems (pp. 5048-5058).

Narayanan, A., & McSherry, F. (2010). On the feasibility of internet-scale author identification. In Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 621-628).

Ribeiro, M.T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1135-1144).

Zemel, R., Wu, Y., Swersky, K., Pitassi, T., & Dwork, C. (2013). Learning fair representations. In International Conference on Machine Learning (pp. 325-333). PMLR.

GITHUB LINK:

https://github.com/YeswanthGolla/Natural-language-Processing