Analyzing Language Bias Between French and English in Conventional Multilingual Sentiment Analysis Models

March 31, 2024

1 Calculating Bias for Multilingual Support Vector Machine and Naive-Bayes for Sentiment Analysis

1.1 Library Imports

Libraries Used

- **Sklearn**: Used to use the Multinomial Naive-Bayes and Support Vector Machine Model, build the Tf-Idf Matrix, use proper train test splitting, and build accuracy reports.
- **Pandas**: Used for building DataFrames
- **Numpy**: Provides operations for the DataFrames
- **FairLearn**: Builds specified bias metrics in models

[ ]:
```
!pip install fairlearn
```

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, classification_report
from fairlearn.metrics import MetricFrame, demographic_parity_difference,␣
↪equalized_odds_difference, selection_rate, equalized_odds_ratio,␣
↪demographic_parity_ratio
from google.colab import drive
drive.mount('/content/drive')
```

1.2 Pre-Processing

Installing spaCy's French and English stop words and pre-processing tools, removing taggers as dataset is unordered.

[ ]:
```
import spacy
!python -m spacy download en_core_web_sm
```

```
!python -m spacy download fr_core_news_sm
```

[ ]:
```
nlp_en = spacy.load("en_core_web_sm", disable=["ner", "parser", "tagger"])
nlp_fr = spacy.load("fr_core_news_sm", disable=["ner", "parser", "tagger"])
```

1.3 Pre-Processing Functions

- **Review Parsing** - Parses through the dataset and reconstructs the text from the word frequencies provided, along with the sentiment labels as well.
- **Batch Pre-Processing** - Pre-processes the text via spaCy using batch pre-processing with a batch size of 100.
- **Load Dataset** - The fully pre-process data is placed in a dataframe.

[ ]:
```
def review_parsing(line):
    """
    Parses lines from the dataset to reconstruct the text by multiplying by the
    proper word frequencies and extracting the
    sentiment labels as well.

    Args:
      String: The lines in the dataset

    Returns:
      Dict {str:str}: A dictionary that has text and sentiment as the keys and
      the reconstructed text and sentiment as values.
    """
    words = []
    parts = line.strip().split()
    sentiment = None

    for part in parts:
        if part.startswith("#label#"):
            sentiment = part.split(":")[1]
        else:
            word, freq = part.split(":")
            words.extend([word] * int(freq))

    reconstructed_text = " ".join(words)
    return {'text': reconstructed_text, 'sentiment': sentiment}

def batch_preprocess_en(texts):
    """
    Batch pre-process English texts.

    Args:
      List[str]: All of the English texts.
```

```
    Return:
      List[str]: All of the English texts fully pre-processed.
    """
    processed_texts = []
    for doc in nlp_en.pipe(texts, batch_size=100):
        tokens = [token.text.lower() for token in doc if token.is_alpha and not␣
 ↪token.is_stop]
        processed_texts.append(' '.join(tokens))
    return processed_texts

def batch_preprocess_fr(texts):
    """
    Batch pre-process French texts.

    Args:
      List[str]: All of the French texts.

    Return:
      List[str]: All of the French texts fully pre-processed.
    """
    processed_texts = []
    for doc in nlp_fr.pipe(texts, batch_size=20):
        tokens = [token.text.lower() for token in doc if token.is_alpha and not␣
 ↪token.is_stop]
        processed_texts.append(' '.join(tokens))
    return processed_texts


def load_dataset_to_dataframe_en(file_path):
    """
    Transforming the English pre-processed data into a dataframe.

    Args:
      FILE: The CSV file with all the English data.

    Return:
      DataFrame: A dataframe that has the pre-processed texts along with the
      sentiments.
    """
    texts, sentiments = [], []

    with open(file_path, 'r') as file:
        for line in file:
            parsed_line = review_parsing(line)
            texts.append(parsed_line['text'])
            sentiments.append(parsed_line['sentiment'])
```

```
    processed_texts = batch_preprocess_en(texts)

    df = pd.DataFrame({
        'ProcessedText': processed_texts,
        'Sentiment': sentiments
    })

    return df

def load_dataset_to_dataframe_fr(file_path):
    """
    Transforming the French pre-processed data into a dataframe.

    Args:
      FILE: The CSV file with all the French data.

    Return:
      DataFrame: A dataframe that has the pre-processed texts along with the
      sentiments.
    """
    texts, sentiments = [], []

    with open(file_path, 'r') as file:
        for line in file:
            parsed_line = review_parsing(line)
            texts.append(parsed_line['text'])
            sentiments.append(parsed_line['sentiment'])

    processed_texts = batch_preprocess_fr(texts)

    df = pd.DataFrame({
        'ProcessedText': processed_texts,
        'Sentiment': sentiments
    })

    return df
```

1.4 Loading the Particular Datasets

The file path to the Webis-CLS-10 Dataset

[ ]:
```
FILE_PATH_EN = '/content/drive/My Drive/dataset/en/music/unlabeled.processed'
FILE_PATH_FR = '/content/drive/My Drive/dataset/fr/music/unlabeled.processed'

FILE_PATH_EN_dvd = '/content/drive/My Drive/dataset/en/dvd/unlabeled.processed'
FILE_PATH_FR_dvd = '/content/drive/My Drive/dataset/fr/dvd/unlabeled.processed'
```

```
FILE_PATH_EN_books = '/content/drive/My Drive/dataset/en/books/unlabeled.
 ↪processed'
FILE_PATH_FR_books = '/content/drive/My Drive/dataset/fr/books/unlabeled.
 ↪processed'
```

```
Mounted at /content/drive
```

Loading the French and English dataset from the Webis-CLS-10 Dataset. We are taking the all three sub-categories of the dataset which are the dvd, music, and books categories.

[ ]:
```
data_en = load_dataset_to_dataframe_en(FILE_PATH_EN)
data_fr = load_dataset_to_dataframe_fr(FILE_PATH_FR)

data_en_dvd = load_dataset_to_dataframe_en(FILE_PATH_EN_dvd)
data_fr_dvd = load_dataset_to_dataframe_fr(FILE_PATH_FR_dvd)

data_en_books = load_dataset_to_dataframe_en(FILE_PATH_EN_books)
data_fr_books = load_dataset_to_dataframe_fr(FILE_PATH_FR_books)
```

1.5 Caching the Datasets

Caching the datasets so there is no need to keep pre-processing the data

[ ]:
```
data_en.to_csv("/content/drive/My Drive/dataset/en/music/unlabeled.csv",␣
 ↪index=False)
data_fr.to_csv("/content/drive/My Drive/dataset/fr/music/unlabeled.csv",␣
 ↪index=False)

data_en_dvd.to_csv("/content/drive/My Drive/dataset/en/dvd/unlabeled.csv",␣
 ↪index=False)
data_fr_dvd.to_csv("/content/drive/My Drive/dataset/fr/dvd/unlabeled.csv",␣
 ↪index=False)

data_en_books.to_csv("/content/drive/My Drive/dataset/en/books/unlabeled.csv",␣
 ↪index=False)
data_fr_books.to_csv("/content/drive/My Drive/dataset/fr/books/unlabeled.csv",␣
 ↪index=False)
```

[4]:
```
data_en = pd.read_csv("/content/drive/My Drive/dataset/en/music/unlabeled.csv")
data_fr = pd.read_csv("/content/drive/My Drive/dataset/fr/music/unlabeled.csv")

data_en_dvd = pd.read_csv("/content/drive/My Drive/dataset/en/dvd/unlabeled.
 ↪csv")
data_fr_dvd = pd.read_csv("/content/drive/My Drive/dataset/fr/dvd/unlabeled.
 ↪csv")
```

```
data_en_books = pd.read_csv("/content/drive/My Drive/dataset/en/books/unlabeled.
  ↪csv")
data_fr_books = pd.read_csv("/content/drive/My Drive/dataset/fr/books/unlabeled.
  ↪csv")
```

2 The Dataset Pre-Processed

[ ]:
```
display(data_en)
```

```
                                              ProcessedText Sentiment
0       pretty pretty pretty good good good records re…  negative
1       classical alive years narration peter children…  positive
2       chamillionaire chamillionaire chamillionaire c…  negative
3       perfect perfect giants giants world world linc…  positive
4       playing playing playing playing autoharp autoh…  positive
…                                                     …         …
25215   num num num num num num num num num num num nu…  positive
25216   album album album excuses excuses words words …  positive
25217   destiny destiny music music stone stone beauti…  negative
25218   best best best tha tha tha rap rap heard heard…  positive
25219   lm cd wonder devil peace entitled selling kam …  negative

[25220 rows x 2 columns]
```

[ ]:
```
display(data_fr)
```

```
                                              ProcessedText Sentiment
0       ringard ringard ringard esrt aufray rêver fair…  negative
1       l l l d d jamais jamais n n peur indépendance …  negative
2       muse muse muse muse muse muse muse qu qu qu qu…  negative
3       num num num num l l l groupe groupe groupe évo…  negative
4       d d d d d d d sympa sympa sympa sympa sympa sy…  negative
…                                                     …         …
15935   d d rappelles dizaine j années cd petits adore…  positive
15936   vallenatos vallenatos vallenatos rien rien mod…  negative
15937   sambora sambora connaissait guitariste album p…  positive
15938   disco disco n n compil qualité énorme passez c…  negative
15939   médiocre cd traditionnels pistes trouve qu mus…  negative

[15940 rows x 2 columns]
```

2.1 Creating the Multi-Lingual Dataset

Sample Data A function that ensures equal number of Englsih and French Reviews in the Dataframe.

[ ]:
```python
def sample_data(df_en, df_fr, perc_en, perc_fr):
    """
    Adjusts sampling to ensure an equal number of English and French samples,
    maximizing the amount of data used while respecting the specified␣
 ↪percentages.

    Args:
      Dataframe: The Pre-Processed English DataFrame
      Dataframe: The Pre-Proecessed French DataFrame
      Float: The percentage of English reviews in the dataset
      Float: The percentage of French reviews in the dataset

    Returns:
      Dataframe: The multilingual dataset
    """
    # Determine the maximum number of samples we can take equally from both␣
 ↪datasets
    max_samples_en = int(len(df_en))
    max_samples_fr = int(len(df_fr))

    max_total = max_samples_en + max_samples_fr

    # The actual number of samples to take from each is the minimum of these␣
 ↪two numbers
    if min(max_samples_en, max_samples_fr) < perc_en * max_total:
      actual_samples = min(max_samples_en, max_samples_fr)
    else:
      actual_samples = perc_en * max_total

    # Sample these numbers from each dataframe
    sample_en = df_en.sample(n=actual_samples, random_state=42)
    sample_fr = df_fr.sample(n=actual_samples, random_state=42)

    # Mark each sample with its language
    sample_en['Language'] = 'English'
    sample_fr['Language'] = 'French'

    # Combine and return the samples
    return pd.concat([sample_en, sample_fr], ignore_index=True)
```

2.1.1 Building the Tf-Idf Matrix

A Tf-Idf Matrix must be built to have the SVM and Naive Bayes to work.

[ ]:
```python
def preprocess_and_vectorize(df):
    """
    Pre-Processes and vectorizes the text data in the DataFrame.
```

```
    Args:
      DataFrame: The Multi-Lingual Dataset

    Returns:
      DataFrame: Tf-Idf of the shape of the samples and feature representing the
      vectorized text data.
      DataFrame: The sentiment labels associated with each text
      TfidfVectorizer: Contains the vocabulary and idf scores of each term
      NumpyArray: An array of every text entry's language
    """
    tfidf = TfidfVectorizer(max_features=10000)
    X = tfidf.fit_transform(df['ProcessedText'])
    y = df['Sentiment'].values
    return X, y, tfidf, df['Language'].values
```

2.2 Calculating the Specified Bias Metrics, Training the SVM and Naive Bayes, and Splitting the Data

Ensure Binary Labels: Ensures the specified labels are binary, so the bias metrics function works as intended.

Calculate Bias Metrics: Calculates the demographic parity ratiom equalized odds ratio, demographic parity difference, and equalized odds difference.

Map Labels: Maps the sentiment labels to have binary classification.

Train and Evaluate: Trains the SVM and Naive Bayes models, and calculating the corresponding precision, recall and f1-scores for each langugage. Also outputs the bias metrics for the models.

[ ]:
```
def ensure_binary_labels(y):
    """
    Ensures the specified labels are binary, so the bias metrics function works
    as intended

    Args:
      DataFrame: The sentiment dataframe
    """
    unique_labels = np.unique(y)
    if set(unique_labels) == {0, 1} or set(unique_labels) == {-1, 1}:
        return np.where(y == -1, 0, y)
    else:
        raise ValueError("Labels must be binary and in {0, 1} or {-1, 1}.")

def calculate_bias_metrics(y_true, y_pred, sensitive_features):
    """
    Calculates the demographic parity ratiom equalized odds ratio,
    demographic parity difference, and equalized odds difference.
```

```
    Args:
      DataFrame: Dataframe the contains the actual sentiment labels for the
      specified text in the dataset
      DataFrame: Dataframe the contains the predicted sentiment labels for the
      specified text in the dataset
      DataFrame: Contains the dataframe with the languages corresponding to
      the sentiment labels
    """
    y_true_binary = ensure_binary_labels(y_true)
    y_pred_binary = ensure_binary_labels(y_pred)

    m_dpr = demographic_parity_ratio(y_true_binary, y_pred_binary,␣
 ↪sensitive_features=sensitive_features)
    m_eqo = equalized_odds_ratio(y_true_binary, y_pred_binary,␣
 ↪sensitive_features=sensitive_features)
    m_dpr_2 = demographic_parity_difference(y_true_binary, y_pred_binary,␣
 ↪sensitive_features=sensitive_features)
    m_eqo_2 = equalized_odds_difference(y_true_binary, y_pred_binary,␣
 ↪sensitive_features=sensitive_features)

    print(f"The demographic parity ratio is {m_dpr}")
    print(f"The equalized odds ratio is {m_eqo}")
    print(f"The demographic parity differece is {m_dpr_2}")
    print(f"The equalized odds difference is {m_eqo_2}")

def map_labels(y):
    """
    Maps the positve and negative setiments to be binary

    Args:
      DataFrame: The dataset without binary labels for sentiment

    Return:
      DataFrame: The dataset with binary labels

    """

    return np.where(y == 'positive', 1, 0)

def train_and_evaluate(X_train, y_train, X_test, y_test, languages_test, model,␣
 ↪model_name="Model"):
    """
    Trains the SVM and Naive Bayes models, and calculating the corresponding
    precision, recall, and f1-scores for each language. Also outputs the
    bias metrics for the models.

    Args:
```

```
    DataFrame: The training dataframe with data other than sentiment
    DataFrame: The training dataframe with the sentiment data
    DataFrame: The testing dataframe with data other than sentiment
    DataFrame: The testing dataframe with the sentiment data
    DataFrame: A language dataframe that corresponds to the y_test sentiment␣
↪data
    SkLearn Model: The specified model to be trained
  """

  y_train_mapped = map_labels(y_train)
  y_test_mapped = map_labels(y_test)

  model.fit(X_train, y_train_mapped)
  y_pred_mapped = model.predict(X_test)

  bias_metrics = calculate_bias_metrics(y_test_mapped, y_pred_mapped,␣
↪languages_test)

  y_pred = np.where(y_pred_mapped == 1, 'positive', 'negative')
  print(f"Results for {model_name}:")
  print("Overall Accuracy:", accuracy_score(y_test, y_pred))
  print("Overall Classification Report:")
  print(classification_report(y_test, y_pred))

  for language in ['English', 'French']:
      idx = languages_test == language
      y_test_lang = y_test[idx]
      y_pred_lang = y_pred[idx]

      print(f"Accuracy on {language}: {accuracy_score(y_test_lang,␣
↪y_pred_lang)}")
      print(f"Classification Report for {language}:")
      print(classification_report(y_test_lang, y_pred_lang))

  print("-------------------------------------------------------")
```

3 Training the Music Data Using SVM and Naive Bayes

Splitting up the data, using an 80-20 split and calling the corresponding functions for a properly trained model.

[ ]:
```
df_sampled = sample_data(data_en, data_fr, 0.5, 0.5)

X, y, tfidf, languages = preprocess_and_vectorize(df_sampled)

# Split the data, ensuring languages array is split consistently with X and y
```

```
X_train, X_test, y_train, y_test, languages_train, languages_test =␣
 ↪train_test_split(X, y, languages, test_size=0.2, random_state=42)
```

[ ]:
```
# Initialize models
svm_model = SVC(kernel='linear')
nb_model = MultinomialNB()

# Train and evaluate models, including language-specific performance
print("Evaluating SVM...")
train_and_evaluate(X_train, y_train, X_test, y_test, languages_test, svm_model,␣
 ↪"SVM")

print("Evaluating Naive Bayes...")
train_and_evaluate(X_train, y_train, X_test, y_test, languages_test, nb_model,␣
 ↪"Naive Bayes")
```

```
Evaluating SVM…
The demographic parity ratio is 0.9867867840290437
The equalized odds ratio is 0.5070733863837312
The demographic parity differece is 0.006797809078653905
The equalized odds difference is 0.07155386326687774
Results for SVM:
Overall Accuracy: 0.8922521957340025
Overall Classification Report:
              precision    recall  f1-score   support

    negative       0.89      0.89      0.89      3120
    positive       0.89      0.89      0.89      3256

    accuracy                           0.89      6376
   macro avg       0.89      0.89      0.89      6376
weighted avg       0.89      0.89      0.89      6376

Accuracy on English: 0.8666871921182266
Classification Report for English:
              precision    recall  f1-score   support

    negative       0.87      0.85      0.86      1612
    positive       0.86      0.88      0.87      1636

    accuracy                           0.87      3248
   macro avg       0.87      0.87      0.87      3248
weighted avg       0.87      0.87      0.87      3248

Accuracy on French: 0.9187979539641944
Classification Report for French:
              precision    recall  f1-score   support
```

```
    negative       0.91      0.93      0.92      1508
    positive       0.93      0.91      0.92      1620

    accuracy                           0.92      3128
   macro avg       0.92      0.92      0.92      3128
weighted avg       0.92      0.92      0.92      3128

--------------------------------------------------
Evaluating Naive Bayes…
The demographic parity ratio is 0.8326970359662453
The equalized odds ratio is 0.31491146318732527
The demographic parity differece is 0.09596226046640549
The equalized odds difference is 0.15724736887139557
Results for Naive Bayes:
Overall Accuracy: 0.8655897114178168
Overall Classification Report:
              precision    recall  f1-score   support

    negative       0.87      0.85      0.86      3120
    positive       0.86      0.88      0.87      3256

    accuracy                           0.87      6376
   macro avg       0.87      0.87      0.87      6376
weighted avg       0.87      0.87      0.87      6376

Accuracy on English: 0.8420566502463054
Classification Report for English:
              precision    recall  f1-score   support

    negative       0.90      0.77      0.83      1612
    positive       0.80      0.91      0.85      1636

    accuracy                           0.84      3248
   macro avg       0.85      0.84      0.84      3248
weighted avg       0.85      0.84      0.84      3248

Accuracy on French: 0.8900255754475703
Classification Report for French:
              precision    recall  f1-score   support

    negative       0.86      0.93      0.89      1508
    positive       0.93      0.85      0.89      1620

    accuracy                           0.89      3128
   macro avg       0.89      0.89      0.89      3128
weighted avg       0.89      0.89      0.89      3128
```

---

4 Training the DVD Data Using SVM and Naive-Bayes

[ ]:
```
df_sampled = sample_data(data_en_dvd, data_fr_dvd, 0.5, 0.5)
X, y, tfidf, languages = preprocess_and_vectorize(df_sampled)

# Split the data, ensuring languages array is split consistently with X and y
X_train, X_test, y_train, y_test, languages_train, languages_test =␣
 ↪train_test_split(X, y, languages, test_size=0.2, random_state=42)

# Initialize models
svm_model = SVC(kernel='linear')
nb_model = MultinomialNB()

# Train and evaluate models, including language-specific performance
print("Evaluating SVM...")
train_and_evaluate(X_train, y_train, X_test, y_test, languages_test, svm_model,␣
 ↪"SVM")

print("Evaluating Naive Bayes...")
train_and_evaluate(X_train, y_train, X_test, y_test, languages_test, nb_model,␣
 ↪"Naive Bayes")
```

```
Evaluating SVM…
The demographic parity ratio is 0.9997391279034616
The equalized odds ratio is 0.8206236956293261
The demographic parity differece is 0.00013303111100959608
The equalized odds difference is 0.027384292368063945
Results for SVM:
Overall Accuracy: 0.8699252136752137
Overall Classification Report:
              precision    recall  f1-score   support

    negative       0.88      0.86      0.87      1870
    positive       0.86      0.88      0.87      1874

    accuracy                           0.87      3744
   macro avg       0.87      0.87      0.87      3744
weighted avg       0.87      0.87      0.87      3744

Accuracy on English: 0.8649214659685864
Classification Report for English:
              precision    recall  f1-score   support

    negative       0.88      0.85      0.87       976
    positive       0.85      0.88      0.86       934
```

```
    accuracy                           0.86      1910
   macro avg       0.87      0.87      0.86      1910
weighted avg       0.87      0.86      0.86      1910

Accuracy on French: 0.8751363140676118
Classification Report for French:
              precision    recall  f1-score   support

    negative       0.87      0.87      0.87       894
    positive       0.88      0.88      0.88       940

    accuracy                           0.88      1834
   macro avg       0.88      0.88      0.88      1834
weighted avg       0.88      0.88      0.88      1834

--------------------------------------------------
Evaluating Naive Bayes…
The demographic parity ratio is 0.9508331761373087
The equalized odds ratio is 0.6591532649529357
The demographic parity differece is 0.025149731368507566
The equalized odds difference is 0.055527285730003303
Results for Naive Bayes:
Overall Accuracy: 0.8624465811965812
Overall Classification Report:
              precision    recall  f1-score   support

    negative       0.86      0.86      0.86      1870
    positive       0.86      0.86      0.86      1874

    accuracy                           0.86      3744
   macro avg       0.86      0.86      0.86      3744
weighted avg       0.86      0.86      0.86      3744

Accuracy on English: 0.856020942408377
Classification Report for English:
              precision    recall  f1-score   support

    negative       0.88      0.84      0.86       976
    positive       0.84      0.88      0.86       934

    accuracy                           0.86      1910
   macro avg       0.86      0.86      0.86      1910
weighted avg       0.86      0.86      0.86      1910

Accuracy on French: 0.8691384950926936
Classification Report for French:
              precision    recall  f1-score   support
```

```
    negative       0.85      0.89      0.87       894
    positive       0.89      0.85      0.87       940

    accuracy                           0.87      1834
   macro avg       0.87      0.87      0.87      1834
weighted avg       0.87      0.87      0.87      1834

----------------------------------------------------
```

5 Training the Books Data Using SVM and Naive-Bayes

[ ]:
```
df_sampled = sample_data(data_en_books, data_fr_books, 0.5, 0.5)
X, y, tfidf, languages = preprocess_and_vectorize(df_sampled)

X_train, X_test, y_train, y_test, languages_train, languages_test =␣
 ↪train_test_split(X, y, languages, test_size=0.2, random_state=42)

svm_model = SVC(kernel='linear')
nb_model = MultinomialNB()

print("Evaluating SVM...")
train_and_evaluate(X_train, y_train, X_test, y_test, languages_test, svm_model,␣
 ↪"SVM")

print("Evaluating Naive Bayes...")
train_and_evaluate(X_train, y_train, X_test, y_test, languages_test, nb_model,␣
 ↪"Naive Bayes")
```

```
Evaluating SVM…
The demographic parity ratio is 0.9938528914599009
The equalized odds ratio is 0.8958006279434851
The demographic parity differece is 0.0031101386888131577
The equalized odds difference is 0.020157858718604205
Results for SVM:
Overall Accuracy: 0.8786127167630058
Overall Classification Report:
              precision    recall  f1-score   support

    negative       0.89      0.87      0.88      6660
    positive       0.87      0.89      0.88      6488

    accuracy                           0.88     13148
   macro avg       0.88      0.88      0.88     13148
weighted avg       0.88      0.88      0.88     13148

Accuracy on English: 0.8699124558439564
```

```
Classification Report for English:
              precision    recall  f1-score   support

    negative       0.88      0.86      0.87      3300
    positive       0.86      0.88      0.87      3211

    accuracy                           0.87      6511
   macro avg       0.87      0.87      0.87      6511
weighted avg       0.87      0.87      0.87      6511

Accuracy on French: 0.8871478077444629
Classification Report for French:
              precision    recall  f1-score   support

    negative       0.90      0.88      0.89      3360
    positive       0.88      0.90      0.89      3277

    accuracy                           0.89      6637
   macro avg       0.89      0.89      0.89      6637
weighted avg       0.89      0.89      0.89      6637

--------------------------------------------------
Evaluating Naive Bayes…
The demographic parity ratio is 0.9714034311355366
The equalized odds ratio is 0.7669034464614575
The demographic parity differece is 0.014792388870451978
The equalized odds difference is 0.038354978354978364
Results for Naive Bayes:
Overall Accuracy: 0.869257681776696
Overall Classification Report:
              precision    recall  f1-score   support

    negative       0.88      0.85      0.87      6660
    positive       0.86      0.88      0.87      6488

    accuracy                           0.87     13148
   macro avg       0.87      0.87      0.87     13148
weighted avg       0.87      0.87      0.87     13148

Accuracy on English: 0.8573183842727692
Classification Report for English:
              precision    recall  f1-score   support

    negative       0.88      0.84      0.86      3300
    positive       0.84      0.88      0.86      3211

    accuracy                           0.86      6511
   macro avg       0.86      0.86      0.86      6511
```

```
weighted avg       0.86      0.86      0.86      6511

Accuracy on French: 0.8809703179147205
Classification Report for French:
              precision    recall  f1-score   support

    negative       0.89      0.87      0.88      3360
    positive       0.87      0.89      0.88      3277

    accuracy                           0.88      6637
   macro avg       0.88      0.88      0.88      6637
weighted avg       0.88      0.88      0.88      6637

--------------------------------------------------
```

6 Transforming the Notebook into a Latex File

[ ]:
```
!apt-get -q install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[8]:
```
!jupyter nbconvert --to pdf "/content/drive/My Drive/Colab Notebooks/COMP396.
↪ipynb"
```

```
[NbConvertApp] Converting notebook /content/drive/My Drive/Colab
Notebooks/COMP396.ipynb to pdf
[NbConvertApp] Writing 77226 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 83905 bytes to /content/drive/My Drive/Colab
Notebooks/COMP396.pdf
```

[9]:
```
import os
from google.colab import files
files.download(f"/content/drive/My Drive/Colab Notebooks/COMP396.pdf")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```