ASSIGNMENT 3

import pandas as pd

# Load the dataset

file\_path = r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\Assignment 3\data\_tshikama\_xls-1 (1).xlsx'

# Replace with your file path

df = pd.read\_excel(file\_path)

# Ensure columns are named correctly

df.columns = ['ciluba', 'french', 'score', 'sentiment', 'nature']

df



import pandas as pd

from deep\_translator import MyMemoryTranslator

from tqdm import tqdm

import numpy as np

# Translation function using MyMemoryTranslator

def translate\_text(text, source\_language, target\_language):

    try:

        translator = MyMemoryTranslator(source=source\_language, target=target\_language)

        translation = translator.translate(text)

        return translation

    except Exception as e:

        return f"Error: {e}"

# Function to translate a row

def translate\_row(row):

    try:

        # Translate from French to English

        english = translate\_text(row['french'], source\_language='fr-FR', target\_language='en-GB')

        # Translate from English to Afrikaans

        afrikaans = translate\_text(english, source\_language='en-GB', target\_language='af-ZA')

        # Translate from English to Zulu

        zulu = translate\_text(english, source\_language='en-GB', target\_language='zu-ZA')

        # Translate from English to Xhosa

        xhosa = translate\_text(english, source\_language='en-GB', target\_language='xh-ZA')

        return pd.Series({'English': english, 'Afrikaans': afrikaans, 'Zulu': zulu, 'Xhosa': xhosa})

    except Exception as e:

        return pd.Series({'English': None, 'Afrikaans': None, 'Zulu': None, 'Xhosa': None})

if \_\_name\_\_ == '\_\_main\_\_':

    # Load the dataset

    file\_path = r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\Assignment 3\data\_tshikama\_xls-1 (1).xlsx'

    df = pd.read\_excel(file\_path)

    # Ensure columns are named correctly

    df.columns = ['ciluba', 'french', 'score', 'sentiment', 'nature']

    # Define batch size

    batch\_size = 500

    start\_index = 1000  # Starting from the 501st row

    end\_index = 1500   # Ending at the 1000th row

    # Slice the DataFrame to process only the desired batch

    df\_batch = df.iloc[start\_index:end\_index]

    # Apply the translation row by row with a progress bar

    tqdm.pandas()

    translations = df\_batch.progress\_apply(translate\_row, axis=1)

    df\_translated\_batch = pd.concat([df\_batch, translations], axis=1)

    # Save the translated batch to Excel

    output\_file = r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\lexicon\_expanded\_batch\_1001\_to\_1500.xlsx'

    df\_translated\_batch.to\_excel(output\_file, index=False)

    print(f"Success! Translated batch saved to {output\_file}")

#Sentiment analysis 1 - textblob

from textblob import TextBlob

import pandas as pd

# Load the lexicon file

lexicon\_df = pd.read\_excel(r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\Assignment 3\lexicon\_expanded.xlsx')

# Define a function to calculate sentiment

def get\_sentiment(text):

    text = str(text)  # Ensure the text is a string

    blob = TextBlob(text)

    polarity = blob.sentiment.polarity

    sentiment\_label = 'Positive' if polarity > 0 else 'Negative' if polarity < 0 else 'Neutral'

    return polarity, sentiment\_label

# Apply sentiment analysis for each language

for lang in ['English', 'Afrikaans', 'Zulu', 'Xhosa']:

    lexicon\_df[f'{lang}\_sentiment\_score'], lexicon\_df[f'{lang}\_sentiment'] = zip(

        \*lexicon\_df[lang].apply(lambda x: get\_sentiment(x) if pd.notnull(x) else (0, 'Neutral'))

    )

# Save the expanded lexicon with sentiment analysis

lexicon\_df.to\_excel(r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\Assignment 3\lexicon\_expanded\_with\_sentiment\_textblob.xlsx', index=False)

# Display the first few rows to verify

print(lexicon\_df.head())

A screen shot of a computer

Description automatically generated

.

#Sentiment 2 - vader

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Load the lexicon file

lexicon\_df = pd.read\_excel(r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\Assignment 3\lexicon\_expanded.xlsx')

# Initialize VADER sentiment analyzer

analyzer = SentimentIntensityAnalyzer()

# Define a function to calculate sentiment using VADER

def get\_vader\_sentiment(text):

    sentiment\_scores = analyzer.polarity\_scores(str(text))  # Ensure text is a string

    polarity = sentiment\_scores['compound']  # Compound score is a normalized composite score

    sentiment\_label = 'Positive' if polarity > 0 else 'Negative' if polarity < 0 else 'Neutral'

    return polarity, sentiment\_label

# Apply sentiment analysis for each language

for lang in ['English', 'Afrikaans', 'Zulu', 'Xhosa']:

    lexicon\_df[f'{lang}\_sentiment\_score'], lexicon\_df[f'{lang}\_sentiment'] = zip(

        \*lexicon\_df[lang].apply(get\_vader\_sentiment)

    )

# Save the expanded lexicon with sentiment analysis

lexicon\_df.to\_excel(r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\Assignment 3\vaderAnalysis.xlsx', index=False)

# Display the first few rows to verify

print(lexicon\_df.head())

A screen shot of a computer

Description automatically generated

#Graphs for the sentiment analysis

#Graphs for sentiment for each language:

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the data files

textblob\_df = pd.read\_excel(r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\Assignment 3\lexicon\_expanded\_with\_sentiment\_textblob.xlsx')

vader\_df = pd.read\_excel(r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\Assignment 3\vaderAnalysis.xlsx')

# Function to plot sentiment distribution bar charts

def plot\_sentiment\_distribution(data, analysis\_name):

    fig, axes = plt.subplots(2, 2, figsize=(15, 10))

    fig.suptitle(f'Sentiment Distribution by Language - {analysis\_name}')

    languages = ['English', 'Afrikaans', 'Zulu', 'Xhosa']

    for ax, lang in zip(axes.flatten(), languages):

        sns.countplot(data=data, x=f'{lang}\_sentiment', ax=ax, palette='viridis')

        ax.set\_title(f'{lang} Sentiment Count')

        ax.set\_xlabel('Sentiment')

        ax.set\_ylabel('Count')

    plt.tight\_layout(rect=[0, 0, 1, 0.96])

    plt.show()

# Function to plot sentiment score distribution box plots

def plot\_sentiment\_score\_distribution(data, analysis\_name):

    fig, axes = plt.subplots(2, 2, figsize=(15, 10))

    fig.suptitle(f'Sentiment Score Distribution by Language - {analysis\_name}')

    languages = ['English', 'Afrikaans', 'Zulu', 'Xhosa']

    for ax, lang in zip(axes.flatten(), languages):

        sns.boxplot(data=data, y=f'{lang}\_sentiment\_score', ax=ax, palette='viridis')

        ax.set\_title(f'{lang} Sentiment Score')

        ax.set\_ylabel('Sentiment Score')

    plt.tight\_layout(rect=[0, 0, 1, 0.96])

    plt.show()

# Plot TextBlob Sentiment Distribution

plot\_sentiment\_distribution(textblob\_df, 'TextBlob')

# Plot TextBlob Sentiment Score Distribution

plot\_sentiment\_score\_distribution(textblob\_df, 'TextBlob')

# Plot VADER Sentiment Distribution

plot\_sentiment\_distribution(vader\_df, 'VADER')

# Plot VADER Sentiment Score Distribution

plot\_sentiment\_score\_distribution(vader\_df, 'VADER')

A group of graphs showing different types of numbers

Description automatically generated

A group of white squares with black lines

Description automatically generated

A group of graphs showing different types of data

Description automatically generated with medium confidence

A screenshot of a graph

Description automatically generated

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score, roc\_curve

import matplotlib.pyplot as plt

from sklearn.preprocessing import label\_binarize

# Load the dataset

file\_path = r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\lexicon\_expanded.xlsx'  # Path to the expanded lexicon

df = pd.read\_excel(file\_path)

# Ensure columns are named correctly and include English and South African languages

df.columns = ['ciluba', 'french', 'score', 'sentiment', 'nature', 'english', 'afrikaans', 'zulu', 'xhosa']

# Step 1: Translation functions

def translate\_text\_using\_lexicon(text, lexicon):

    words = text.lower().split()

    translated\_words = [lexicon.get(word, word) for word in words]

    return ' '.join(translated\_words)

# Create translation lexicon from the dataset

translation\_lexique = dict(zip(df['french'].str.lower(), df['ciluba']))

# Step 2: Sentiment Analysis Function

lexique = dict(zip(df['ciluba'].str.lower(), df['score']))

def analyse\_sentiment(text):

    words = text.lower().split()

    word\_scores = {word: lexique.get(word, 0) for word in words}

    score = sum(word\_scores.values())

    if score > 0.05:

        sentiment = "Positive"

    elif score < -0.05:

        sentiment = "Negative"

    else:

        sentiment = "Neutral"

    return score, sentiment, word\_scores

# Step 3: Machine Learning Pipeline

def preprocess\_and\_train\_ml\_models(df):

    # Prepare the dataset for machine learning

    df = df.dropna(subset=['french', 'score', 'sentiment'])  # Drop rows with missing values

    X = df['french']  # Features (text)

    y = df['sentiment']  # Target (sentiment labels)

    # Binarize the output for multi-class ROC

    y\_bin = label\_binarize(y, classes=['Negatif', 'Neutre', 'Positif'])

    n\_classes = y\_bin.shape[1]

    # Split into training and testing datasets

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

    # Feature extraction using TF-IDF

    vectorizer = TfidfVectorizer(max\_features=5000)  # Limit to top 5000 features

    X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

    X\_test\_tfidf = vectorizer.transform(X\_test)

    # Initialize machine learning models

    models = {

        "Logistic Regression": LogisticRegression(max\_iter=1000),

        "Decision Tree": DecisionTreeClassifier(),

        "Random Forest": RandomForestClassifier(),

        "Support Vector Machine": SVC(probability=True)

    }

    # Train and evaluate each model

    for model\_name, model in models.items():

        model.fit(X\_train\_tfidf, y\_train)

        y\_pred = model.predict(X\_test\_tfidf)

        y\_proba = model.predict\_proba(X\_test\_tfidf) if hasattr(model, "predict\_proba") else None

        # Print classification report

        print(f"Classification Report for {model\_name}:")

        print(classification\_report(y\_test, y\_pred, target\_names=['Negatif', 'Neutre', 'Positif']))

        # Confusion matrix

        cm = confusion\_matrix(y\_test, y\_pred)

        plt.figure(figsize=(6, 6))

        plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

        plt.title(f'Confusion Matrix for {model\_name}')

        plt.colorbar()

        tick\_marks = range(n\_classes)

        plt.xticks(tick\_marks, ['Negatif', 'Neutre', 'Positif'], rotation=45)

        plt.yticks(tick\_marks, ['Negatif', 'Neutre', 'Positif'])

        plt.ylabel('True label')

        plt.xlabel('Predicted label')

        plt.show()

        # ROC Curve

        if y\_proba is not None:

            # Compute ROC curve and ROC area for each class

            fpr = dict()

            tpr = dict()

            roc\_auc = dict()

            for i in range(n\_classes):

                fpr[i], tpr[i], \_ = roc\_curve(label\_binarize(y\_test, classes=['Negatif', 'Neutre', 'Positif'])[:, i], y\_proba[:, i])

                roc\_auc[i] = roc\_auc\_score(label\_binarize(y\_test, classes=['Negatif', 'Neutre', 'Positif'])[:, i], y\_proba[:, i])

            # Plot ROC curve for each class

            plt.figure()

            for i in range(n\_classes):

                plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc\_auc[i]:.2f})')

            plt.plot([0, 1], [0, 1], linestyle='--')

            plt.title(f'ROC Curve for {model\_name}')

            plt.xlabel('False Positive Rate')

            plt.ylabel('True Positive Rate')

            plt.legend(loc='lower right')

            plt.show()

# Run the updated machine learning pipeline

preprocess\_and\_train\_ml\_models(df)

A screenshot of a graph

Description automatically generated

A screenshot of a graph

Description automatically generated

A screenshot of a graph

Description automatically generated

A screenshot of a graph

Description automatically generated

import seaborn as sns

import matplotlib.pyplot as plt

# Plot word count distribution for Ciluba

plt.figure(figsize=(10, 6))

sns.countplot(data=df, x='ciluba', palette='viridis', order=df['ciluba'].value\_counts().index)

plt.title('Count of Ciluba Words')

plt.xticks(rotation=90)

plt.show()

# Plot word count distribution for French

plt.figure(figsize=(10, 6))

sns.countplot(data=df, x='french', palette='viridis', order=df['french'].value\_counts().index)

plt.title('Count of French Words')

plt.xticks(rotation=90)

plt.show()

A screenshot of a computer

Description automatically generated

# Plot sentiment distribution (positive, negative, neutral)

plt.figure(figsize=(8, 6))

sns.countplot(data=df, x='sentiment', palette='Blues')

plt.title('Count of Sentiments')

plt.show()

A graph with blue and white bars

Description automatically generated with medium confidence

# Plot score distribution

plt.figure(figsize=(8, 6))

sns.histplot(data=df, x='score', bins=20, kde=True, palette='Greens')

plt.title('Score Distribution')

plt.show()

A graph of a number of bars

Description automatically generated with medium confidence

# Plot distribution of "nature" (categorical variable)

plt.figure(figsize=(8, 6))

sns.countplot(data=df, x='nature', palette='Purples')

plt.title('Count of Nature')

plt.xticks(rotation=45)

plt.show()

A graph with numbers and a bar

Description automatically generated

# Load your dataset

file\_path = r'C:\Users\AlexGreyling\OneDrive - Agile Bridge\Honours\Semester 2\INF 791\Assignment 3\coherent\_english\_sentences\_dataset\_v2.xlsx'

df\_sentences = pd.read\_excel(file\_path)

# Print column names to verify them

print("Columns in the dataset:", df.columns)

# Ensure columns are named correctly

# Uncomment and adjust the column names as needed based on the output

# df.columns = ['ciluba', 'french', 'score', 'sentiment', 'nature']

# Display first few rows to inspect data structure

print(df\_sentences.head())

A screen shot of a computer

Description automatically generated

import requests

from deep\_translator import MyMemoryTranslator

from textblob import TextBlob

# Function to translate text using MyMemoryTranslator with retries and bypass SSL verification

def translate\_text(text, source\_language, target\_language, retries=3):

    try:

        translator = MyMemoryTranslator(source=source\_language, target=target\_language)

        # Bypass SSL verification by overriding requests behavior

        original\_get = requests.get

        def new\_get(\*args, \*\*kwargs):

            kwargs['verify'] = False  # Disable SSL verification

            return original\_get(\*args, \*\*kwargs)

        requests.get = new\_get

        # Perform the translation

        translation = translator.translate(text)

        # Restore the original get method

        requests.get = original\_get

        return translation

    except Exception as e:

        if retries > 0:

            print(f"Retrying translation for {text}... ({3 - retries} retries left)")

            return translate\_text(text, source\_language, target\_language, retries - 1)

        return f"Error: {e}"

# Function to perform sentiment analysis and compute sentiment score

def analyze\_sentiment(text):

    blob = TextBlob(text)

    # Get polarity score (-1 to 1)

    polarity = blob.sentiment.polarity

    # Convert polarity to the desired -9 to +9 scale

    sentiment\_score = int(polarity \* 9)

    # Label sentiment based on the score

    if sentiment\_score > 0:

        sentiment\_label = 'Positive'

    elif sentiment\_score < 0:

        sentiment\_label = 'Negative'

    else:

        sentiment\_label = 'Neutral'

    return sentiment\_score, sentiment\_label

# Function to prompt user for language selection and translate input sentence, with sentiment analysis

def interactive\_translation():

    # Define supported languages

    languages = {

        '1': ('af-ZA', 'Afrikaans'),

        '2': ('zu-ZA', 'Zulu'),

        '3': ('xh-ZA', 'Xhosa')

    }

    # Prompt user for language choice

    print("Select the language you want to translate to:")

    print("1. Afrikaans")

    print("2. Zulu")

    print("3. Xhosa")

    choice = input("Enter the number of the language (1, 2, or 3): ")

    # Ensure valid choice

    if choice not in languages:

        print("Invalid choice. Please select 1, 2, or 3.")

        return

    # Get the language code and name

    target\_language\_code, target\_language\_name = languages[choice]

    # Prompt user for the sentence in English

    sentence = input(f"Enter the sentence in English to translate into {target\_language\_name}: ")

    # Translate the sentence

    translated\_sentence = translate\_text(sentence, source\_language='en-GB', target\_language=target\_language\_code)

    # Perform sentiment analysis on both original and translated sentence

    original\_sentiment\_score, original\_sentiment\_label = analyze\_sentiment(sentence)

    translated\_sentiment\_score, translated\_sentiment\_label = analyze\_sentiment(translated\_sentence)

    # Display the results

    print(f"\nOriginal sentence: {sentence}")

    print(f"Sentiment (Original): {original\_sentiment\_label} (Score: {original\_sentiment\_score})")

    print(f"Translated into {target\_language\_name}: {translated\_sentence}")

    print(f"Sentiment (Translated): {translated\_sentiment\_label} (Score: {translated\_sentiment\_score})")

# Call the interactive translation function

interactive\_translation()

