DA5400 Assignment 6

MM21B051 - Preethi

In [18]:
```
from datasets import load_dataset
import pandas as pd
import glob
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import os
from unidecode import unidecode

# Load the MASSIVE dataset
dataset = load_dataset("qanastek/MASSIVE", trust_remote_code=True)
```

Task 1

We download the data set and split into different files pertaining to each partition: test, train, validation. We then split the dataset into 27 files each for a locale in the list target_locales

In [3]:
```
# Extract the required fields from the streaming dataset
def extract_relevant_columns(data_split):
    for row in data_split:
        yield {key: row[key] for key in ['locale', 'partition', 'utt', 'tokens']}

# Extract subsets for train, validation, and test
train_data = extract_relevant_columns(dataset['train'])
test_data = extract_relevant_columns(dataset['test'])
validation_data = extract_relevant_columns(dataset['validation'])
```

In [4]:
```
# List of locales
target_locales = [
    'af-ZA', 'da-DK', 'de-DE', 'en-US', 'es-ES', 'fr-FR', 'fi-FI', 'hu-HU',
    'is-IS', 'it-IT', 'jv-ID', 'lv-LV', 'ms-MY', 'nb-NO', 'nl-NL', 'pl-PL',
    'pt-PT', 'ro-RO', 'ru-RU', 'sl-SL', 'sv-SE', 'sq-AL', 'sw-KE', 'tl-PH',
    'tr-TR', 'vi-VN', 'cy-GB'
]

output_dir = "language_utterances_2"
os.makedirs(output_dir, exist_ok=True)
```

In [5]:
```
# Function to process and save data from a specific partition
def process_and_save_data(data_split, target_locales, partition, deaccent=False):
    locale_files = {locale: open(os.path.join(output_dir, f"{locale}_{partition}.tx
```

```
    for row in data_split:
        locale = row['locale']
        if locale in target_locales:
            utt = row['utt']
            if deaccent:
                utt = unidecode(utt)  # Deaccent the utterance if required
            locale_files[locale].write(utt + "\n")

    for file in locale_files.values():
        file.close()

# Process train, test, and validation sets
process_and_save_data(dataset['train'], target_locales, partition='train', deaccent
process_and_save_data(dataset['test'], target_locales, partition='test', deaccent=F
process_and_save_data(dataset['validation'], target_locales, partition='validation'

print(f"Data extraction complete! Utterances saved in {output_dir}/")
```

```
Data extraction complete! Utterances saved in language_utterances_2/
```

In [6]:

```
import os
import pandas as pd

def group_partition_to_csv(target_locales, data_dir, partition, output_csv_path):

    combined_data = []
    for locale in target_locales:
        file_path = os.path.join(data_dir, f"{locale}_{partition}.txt")
        if os.path.exists(file_path):
            with open(file_path, 'r', encoding='utf-8') as file:
                lines = file.readlines()
                for line in lines:
                    combined_data.append({
                        'text': line.strip(),
                        'label': locale
                    })
        else:
            print(f"File not found: {file_path}")

    df = pd.DataFrame(combined_data)
    df.to_csv(output_csv_path, index=False, encoding='utf-8') # Save the dataframe
    print(f"{partition.capitalize()} dataset saved to {output_csv_path}.")

def group_all_partitions_to_csv(target_locales, data_dir, output_dir):

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # Group and save for each partition
    for partition in ['train', 'validation', 'test']:
        output_csv_path = os.path.join(output_dir, f"{partition}.csv")
        group_partition_to_csv(target_locales, data_dir, partition, output_csv_path


target_locales = ['af-ZA', 'da-DK', 'de-DE', 'en-US', 'es-ES', 'fr-FR', 'fi-FI', 'h
```

```
                  'is-IS', 'it-IT', 'jv-ID', 'lv-LV', 'ms-MY', 'nb-NO', 'nl-NL',
                  'pl-PL', 'pt-PT', 'ro-RO', 'ru-RU', 'sl-SL', 'sv-SE', 'sq-AL',
                  'sw-KE', 'tl-PH', 'tr-TR', 'vi-VN', 'cy-GB']

data_dir = "language_utterances_2"
output_dir = "grouped_data_final"
os.makedirs(output_dir, exist_ok=True)

# Group data into separate CSV files for train, test, and validation
group_all_partitions_to_csv(target_locales, data_dir, output_dir)
```

```
Train dataset saved to grouped_data_final\train.csv.
Validation dataset saved to grouped_data_final\validation.csv.
Test dataset saved to grouped_data_final\test.csv.
```

In [7]:
```
def load_data(dataset_type):
    csv_file = os.path.join(output_dir, f'{dataset_type}.csv')
    data = pd.read_csv(csv_file)

    texts = data['text']
    labels = data['label']
    return texts, labels

train_texts, train_labels = load_data('train')
test_texts, test_labels = load_data('test')
validation_texts, validation_labels = load_data('validation')
```

Task 2

We apply the multinomial naive bayes classifier with the best hyper-parameter

In [8]:
```
# Define the pipeline: TF-IDF Vectorizer followed by Multinomial Naive Bayes
pipeline = make_pipeline(TfidfVectorizer(), MultinomialNB())
pipeline.fit(train_texts, train_labels)

# Fine-tuning with GridSearchCV on the validation data
param_grid = {
    'multinomialnb__alpha': [0.5, 1.0, 1.5]
}
grid_search = GridSearchCV(pipeline, param_grid, cv=3)
grid_search.fit(validation_texts, validation_labels)
best_model = grid_search.best_estimator_
```

In [9]:
```
from sklearn.metrics import classification_report

# Predict on training, test, and validation partitions
train_predictions = best_model.predict(train_texts)
test_predictions = best_model.predict(test_texts)
val_predictions = best_model.predict(validation_texts)

# Report metrics for each partition

# 1. Training Partition
print("Training Partition Performance:")
```

```
train_report = classification_report(train_labels, train_predictions, zero_division
print(f"Training Macro average Precision: {train_report['macro avg']['precision']:.
print(f"Training Macro average Recall: {train_report['macro avg']['recall']:.4f}")
print(f"Training Macro average F1 Score: {train_report['macro avg']['f1-score']:.4f

# 2. Test Partition
print("Test Partition Performance:")
test_report = classification_report(test_labels, test_predictions, zero_division=1,
print(f"Test Macro average Precision: {test_report['macro avg']['precision']:.4f}")
print(f"Test Macro average Recall: {test_report['macro avg']['recall']:.4f}")
print(f"Test Macro average F1 Score: {test_report['macro avg']['f1-score']:.4f}\n")

# 3. Validation Partition
print("Validation Partition Performance:")
val_report = classification_report(validation_labels, val_predictions, zero_divisio
print(f"Validation Macro average Precision: {val_report['macro avg']['precision']:.
print(f"Validation Macro average Recall: {val_report['macro avg']['recall']:.4f}")
print(f"Validation Macro average F1 Score: {val_report['macro avg']['f1-score']:.4f
```

```
Training Partition Performance:
Training Macro average Precision: 0.9780
Training Macro average Recall: 0.9762
Training Macro average F1 Score: 0.9766

Test Partition Performance:
Test Macro average Precision: 0.9773
Test Macro average Recall: 0.9753
Test Macro average F1 Score: 0.9758

Validation Partition Performance:
Validation Macro average Precision: 0.9920
Validation Macro average Recall: 0.9920
Validation Macro average F1 Score: 0.9920
```

On the test dataset we get a very good precision of 97.7% for the optimised lambda.

Task 3

We split the dataset based on continents in the following blocks of code

In [6]:
```
from datasets import load_dataset
import pandas as pd

locales = ['af-ZA', 'da-DK', 'de-DE', 'en-US', 'es-ES', 'fr-FR', 'fi-FI', 'hu-HU',
           'jv-ID', 'lv-LV', 'ms-MY', 'nb-NO', 'nl-NL', 'pl-PL', 'pt-PT', 'ro-RO',
           'sv-SE', 'sq-AL', 'sw-KE', 'tl-PH', 'tr-TR', 'vi-VN', 'cy-GB']

# Filter and export the train, validation, and test partitions to CSV
for partition in ['train', 'validation', 'test']:
    partition_data = dataset[partition]
    df = pd.DataFrame(partition_data)
    df_filtered = df[df['locale'].isin(locales)]
    df_filtered = df_filtered[['locale', 'partition', 'utt', 'tokens']]
```

```
    df_filtered.to_csv(rf"C:\Users\preet\Downloads\new_data\MASSIVE_{partition}.csv

print("Filtered data exported to CSV files successfully.")
```

```
Filtered data exported to CSV files successfully.
```

In [10]:
```
import pandas as pd
import os

# Define the language groups by continent
language_groups = {
    'Africa': ['af-ZA', 'sw-KE'],
    'Asia': ['jv-ID', 'ms-MY', 'tl-PH', 'tr-TR', 'vi-VN'],
    'Europe': ['da-DK', 'de-DE', 'es-ES', 'fr-FR', 'fi-FI', 'hu-HU',
               'is-IS', 'it-IT', 'lv-LV', 'nb-NO', 'nl-NL', 'pl-PL', 'pt-PT',
               'ro-RO', 'ru-RU', 'sl-SL', 'sv-SE', 'sq-AL', 'cy-GB'],
    'North America': ['en-US']
}

output_dir = r"C:\Users\preet\Downloads\Continent_Files"
os.makedirs(output_dir, exist_ok=True)
continent_dataframes = {continent: [] for continent in language_groups.keys()}

# Process the files for each partition and append to continent DataFrames
for partition in ['train', 'validation', 'test']:
    for language in sum(language_groups.values(), []):
        try:
            df = pd.read_csv(rf"C:\Users\preet\Downloads\new_data\MASSIVE_{partitio
            df_language = df[df['locale'] == language]
            for continent, languages in language_groups.items():
                if language in languages:
                    continent_dataframes[continent].append(df_language)
        except FileNotFoundError:
            print(f"File for {language} not found for partition {partition}. Skippi

# Combine and save the continent DataFrames into single CSV files
for continent, dfs in continent_dataframes.items():
    if dfs:
        combined_df = pd.concat(dfs, ignore_index=True)
        combined_df = combined_df[['locale', 'utt', 'tokens']]
        combined_df.to_csv(os.path.join(output_dir, f"{continent}_data.csv"), index

print("Continent files created successfully.")
```

```
Continent files created successfully.
```

In [11]:
```
from sklearn.model_selection import train_test_split
import os
# Combine datasets for all continents
output_dir = r"C:\Users\preet\Downloads\Continent_Files"
continent_files = ['Africa_data.csv', 'Asia_data.csv', 'Europe_data.csv', 'North Am
df_list = [pd.read_csv(os.path.join(output_dir, file)) for file in continent_files]
combined_data = pd.concat(df_list, ignore_index=True)

# feature and target variable
X = combined_data['utt']  # Input features (utterances)
y = combined_data['locale']  # Target labels (language locale)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [12]:
```
import os
from scipy.sparse import save_npz
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import load_npz

# Create TF-IDF features
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

save_npz('X_train_tfidf.npz', X_train_tfidf)
save_npz('X_test_tfidf.npz', X_test_tfidf)
```

In [14]:
```
import os
from scipy.sparse import save_npz
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import load_npz

X_train_tfidf = load_npz('X_train_tfidf.npz')
X_test_tfidf = load_npz('X_test_tfidf.npz')
```

One of the main issues is that while it's feasible to store and work with the sparse dataset, RDA requires converting it into a dense format, which is extremely memory-inefficient. The dense matrix ends up being over 400GB, making it impossible to use directly. To overcome this, a common and effective strategy is to apply low-frequency pruning by selecting the top 1000 features. Additionally, we reduce the matrix size using Singular Value Decomposition (SVD) by retaining only the top 300 components, which helps compress the data while maintaining critical information.

In [15]:
```
import numpy as np
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDisc
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.metrics import classification_report, accuracy_score
import os

#Truncated SVD for dimensionality reduction
def apply_truncated_svd(X_train, X_test, n_components=300):
    svd = TruncatedSVD(n_components=n_components)
    X_train_reduced = svd.fit_transform(X_train)
    X_test_reduced = svd.transform(X_test)

    return X_train_reduced, X_test_reduced

#defining Regularized Discriminant Analysis(RDA) model
class RegularizedDiscriminantAnalysis(BaseEstimator, ClassifierMixin):
    def __init__(self, lambda_param=0.5):
        self.lambda_param = lambda_param
```

```
        self.lda = LinearDiscriminantAnalysis()
        self.qda = QuadraticDiscriminantAnalysis()

    def fit(self, X, y):
        self.lda.fit(X, y)
        self.qda.fit(X, y)
        return self

    def predict(self, X):
        lda_probs = self.lda.predict_proba(X)
        qda_probs = self.qda.predict_proba(X)
        final_probs = (1 - self.lambda_param) * lda_probs + self.lambda_param * qda
        return np.argmax(final_probs, axis=1)

#dense matrices
X_train_tfidf = load_npz('X_train_tfidf.npz')
X_test_tfidf = load_npz('X_test_tfidf.npz')
X_train_reduced, X_test_reduced = apply_truncated_svd(X_train_tfidf, X_test_tfidf,

#training RDA model
rda_model = RegularizedDiscriminantAnalysis(lambda_param=0.5)
```

In [16]:

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)  # Encodes y_train as numeri
y_test_encoded = label_encoder.transform(y_test)
# train the RDA model with encoded labels
rda_model.fit(X_train_reduced, y_train_encoded)
y_pred_encoded = rda_model.predict(X_test_reduced)
#convert predictions to strings
y_pred = label_encoder.inverse_transform(y_pred_encoded)

# Convert y_test_encoded to strings comparison
y_test = label_encoder.inverse_transform(y_test_encoded)

print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))
```

```
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
```

```
Accuracy: 0.9525859170085412
              precision    recall  f1-score   support

       af-ZA       0.93      0.95      0.94      3367
       cy-GB       0.97      0.98      0.98      3235
       da-DK       0.90      0.80      0.85      3311
       de-DE       0.97      0.97      0.97      3358
       en-US       0.93      0.97      0.95      3294
       es-ES       0.93      0.96      0.94      3297
       fi-FI       0.94      0.95      0.95      3375
       fr-FR       0.98      0.97      0.98      3220
       hu-HU       0.98      0.95      0.97      3330
       is-IS       0.96      0.97      0.96      3247
       it-IT       0.96      0.96      0.96      3308
       jv-ID       0.97      0.92      0.94      3354
       lv-LV       0.93      0.97      0.95      3356
       ms-MY       0.94      0.97      0.95      3329
       nb-NO       0.85      0.87      0.86      3295
       nl-NL       0.94      0.92      0.93      3354
       pl-PL       0.97      0.96      0.96      3287
       pt-PT       0.97      0.94      0.95      3247
       ro-RO       0.99      0.95      0.97      3325
       ru-RU       0.91      0.99      0.95      3259
       sl-SL       0.99      0.95      0.97      3361
       sq-AL       0.99      0.98      0.98      3291
       sv-SE       0.91      0.96      0.93      3311
       sw-KE       0.97      0.99      0.98      3303
       tl-PH       0.97      0.98      0.98      3249
       tr-TR       0.99      0.96      0.97      3275
       vi-VN       0.99      0.99      0.99      3276

    accuracy                           0.95     89214
   macro avg       0.95      0.95      0.95     89214
weighted avg       0.95      0.95      0.95     89214
```

In [17]:

```python
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDisc
from sklearn.metrics import accuracy_score, classification_report
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
import pandas as pd
from sklearn.base import BaseEstimator, ClassifierMixin

train_data = pd.read_csv(r'C:\Users\preet\Downloads\new_data\MASSIVE_train.csv')
validation_data = pd.read_csv(r'C:\Users\preet\Downloads\new_data\MASSIVE_validatio
test_data = pd.read_csv(r'C:\Users\preet\Downloads\new_data\MASSIVE_test.csv')

X_train = train_data['utt']
y_train = train_data['locale']

X_val = validation_data['utt']
y_val = validation_data['locale']

X_test = test_data['utt']
```

```
y_test = test_data['locale']

vectorizer = TfidfVectorizer(max_features=10000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_val_tfidf = vectorizer.transform(X_val)
X_test_tfidf = vectorizer.transform(X_test)

n_components = 300
svd = TruncatedSVD(n_components=n_components, random_state=42)

X_train_reduced = svd.fit_transform(X_train_tfidf)
X_val_reduced = svd.transform(X_val_tfidf)
X_test_reduced = svd.transform(X_test_tfidf)

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.transform(y_val)
y_test_encoded = label_encoder.transform(y_test)

class RegularizedDiscriminantAnalysisCV(BaseEstimator, ClassifierMixin):
    def __init__(self, lambda_param=0.5):
        self.lambda_param = lambda_param
        self.lda = LinearDiscriminantAnalysis()
        self.qda = QuadraticDiscriminantAnalysis()

    def fit(self, X, y):
        self.lda.fit(X, y)
        self.qda.fit(X, y)
        return self

    def predict(self, X):
        lda_probs = self.lda.predict_proba(X)
        qda_probs = self.qda.predict_proba(X)
        final_probs = (1 - self.lambda_param) * lda_probs + self.lambda_param * qda
        return np.argmax(final_probs, axis=1)

lambda_values = np.linspace(0, 1, 10)
best_lambda = None
best_accuracy = 0

for lambda_param in lambda_values:
    rda_model = RegularizedDiscriminantAnalysisCV(lambda_param=lambda_param)
    rda_model.fit(X_train_reduced, y_train_encoded)
    y_val_pred_encoded = rda_model.predict(X_val_reduced)
    y_val_pred = label_encoder.inverse_transform(y_val_pred_encoded)

    accuracy = accuracy_score(y_val_encoded, y_val_pred_encoded)
    print(f"Lambda: {lambda_param}, Validation Accuracy: {accuracy}")

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_lambda = lambda_param

print(f"Best lambda: {best_lambda} with validation accuracy: {best_accuracy}")

final_rda_model = RegularizedDiscriminantAnalysisCV(lambda_param=best_lambda)
```

```
final_rda_model.fit(X_train_reduced, y_train_encoded)

y_test_pred_encoded = final_rda_model.predict(X_test_reduced)

y_test_pred = label_encoder.inverse_transform(y_test_pred_encoded)

print(f"Test Accuracy with lambda {best_lambda}: {accuracy_score(y_test, y_test_pre
print(classification_report(y_test, y_test_pred))
```

```
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Lambda: 0.0, Validation Accuracy: 0.853910477127398
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Lambda: 0.1111111111111111, Validation Accuracy: 0.857736240913811
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Lambda: 0.2222222222222222, Validation Accuracy: 0.8624911187626387
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Lambda: 0.3333333333333333, Validation Accuracy: 0.8730028602138784
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Lambda: 0.4444444444444444, Validation Accuracy: 0.9163615164598933
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Lambda: 0.5555555555555556, Validation Accuracy: 0.9421580951339928
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Lambda: 0.6666666666666666, Validation Accuracy: 0.9420670055200306
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Lambda: 0.7777777777777777, Validation Accuracy: 0.9420852234428231
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Lambda: 0.8888888888888888, Validation Accuracy: 0.9420670055200306
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Lambda: 1.0, Validation Accuracy: 0.9420487875972382
Best lambda: 0.5555555555555556 with validation accuracy: 0.9421580951339928
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\dis
criminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
```

```
Test Accuracy with lambda 0.5555555555555556: 0.9420035368253257
              precision    recall  f1-score   support

       af-ZA       0.94      0.94      0.94      2974
       cy-GB       0.99      0.97      0.98      2974
       da-DK       0.89      0.78      0.83      2974
       de-DE       0.98      0.96      0.97      2974
       en-US       0.92      0.96      0.94      2974
       es-ES       0.94      0.94      0.94      2974
       fi-FI       0.99      0.92      0.95      2974
       fr-FR       0.99      0.97      0.98      2974
       hu-HU       0.98      0.92      0.95      2974
       is-IS       0.98      0.95      0.97      2974
       it-IT       0.97      0.96      0.96      2974
       jv-ID       0.97      0.91      0.94      2974
       lv-LV       0.98      0.95      0.96      2974
       ms-MY       0.94      0.96      0.95      2974
       nb-NO       0.84      0.86      0.85      2974
       nl-NL       0.95      0.92      0.93      2974
       pl-PL       0.98      0.93      0.96      2974
       pt-PT       0.96      0.93      0.95      2974
       ro-RO       0.99      0.95      0.97      2974
       ru-RU       0.62      1.00      0.77      2974
       sl-SL       0.99      0.94      0.96      2974
       sq-AL       0.99      0.97      0.98      2974
       sv-SE       0.92      0.95      0.93      2974
       sw-KE       0.98      0.97      0.98      2974
       tl-PH       0.98      0.99      0.98      2974
       tr-TR       0.99      0.94      0.96      2974
       vi-VN       1.00      0.99      1.00      2974

    accuracy                           0.94     80298
   macro avg       0.95      0.94      0.94     80298
weighted avg       0.95      0.94      0.94     80298
```

Using RDA, we obtain a test accuracy of 95% on the optimised hyper-parameter lambda.