Nikshay Jain - MM21B044

Assign 6

```
import os
import numpy as np
import pandas as pd
import seaborn as sns
from scipy.linalg import inv
import matplotlib.pyplot as plt
from collections import Counter
from wordcloud import WordCloud
from unidecode import unidecode
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from datasets import load_dataset, DatasetDict
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,
QuadraticDiscriminantAnalysis
```

```
c:\Users\Nikshay Jain\AppData\Local\Programs\Python\Python38\lib\site-
packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please
update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Get the dataset

```
data = load_dataset("qanastek/MASSIVE", trust_remote_code=True)
print(data)
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'locale', 'partition', 'scenario', 'intent',
'utt', 'annot_utt', 'tokens', 'ner_tags', 'worker_id', 'slot_method',
'judgments'],
        num_rows: 587214
    })
    validation: Dataset({
        features: ['id', 'locale', 'partition', 'scenario', 'intent',
'utt', 'annot_utt', 'tokens', 'ner_tags', 'worker_id', 'slot_method',
'judgments'],
        num_rows: 103683
    })
    test: Dataset({
        features: ['id', 'locale', 'partition', 'scenario', 'intent',
'utt', 'annot_utt', 'tokens', 'ner_tags', 'worker_id', 'slot_method',
'judgments'],
        num_rows: 151674
    })
})

des_col = ['locale', 'partition', 'utt', 'tokens']

data1 = DatasetDict({
    'train': data['train'].select_columns(des_col),
    'validation': data['validation'].select_columns(des_col),
    'test': data['test'].select_columns(des_col)
})

print(data1)

DatasetDict({
    train: Dataset({
        features: ['locale', 'partition', 'utt', 'tokens'],
        num_rows: 587214
    })
    validation: Dataset({
        features: ['locale', 'partition', 'utt', 'tokens'],
        num_rows: 103683
    })
    test: Dataset({
        features: ['locale', 'partition', 'utt', 'tokens'],
        num_rows: 151674
    })
})
```

Task 1

Create files in a folder named utts_by_locale

```
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']

# Directory to store output files
output_dir = "utts_by_locale"
os.makedirs(output_dir, exist_ok=True)
```

```
# Function to extract and save utterances for each locale for all
dataset partitions
def extr_utt(dataset_dict, locales, output_dir, deaccent):
    partitions = ['train', 'validation', 'test']
    for partition in partitions:
        dataset = dataset_dict[partition]
        for locale in locales:
            locale_data = dataset.filter(lambda example:
example['locale'] == locale)

            file_path = os.path.join(output_dir,
f"{locale}_{partition}.txt")
            with open(file_path, 'w', encoding='utf-8') as file:
                for utt in locale_data['utt']:
                    if deaccent:
                        utt = unidecode(utt)
                    file.write(utt + "\n")       # 1 utterance/line

            print(f"Saved {locale} utterances from {partition}
partition to {file_path}")

extr_utt(data1, locales, output_dir, True)
```

```
Saved af-ZA utterances from train partition to utts_by_locale\af-
ZA_train.txt
Saved da-DK utterances from train partition to utts_by_locale\da-
DK_train.txt
Saved de-DE utterances from train partition to utts_by_locale\de-
DE_train.txt
Saved en-US utterances from train partition to utts_by_locale\en-
US_train.txt
Saved es-ES utterances from train partition to utts_by_locale\es-
ES_train.txt
Saved fr-FR utterances from train partition to utts_by_locale\fr-
FR_train.txt
Saved fi-FI utterances from train partition to utts_by_locale\fi-
FI_train.txt
Saved hu-HU utterances from train partition to utts_by_locale\hu-
HU_train.txt
Saved is-IS utterances from train partition to utts_by_locale\is-
IS_train.txt
Saved it-IT utterances from train partition to utts_by_locale\it-
IT_train.txt
Saved jv-ID utterances from train partition to utts_by_locale\jv-
ID_train.txt
Saved lv-LV utterances from train partition to utts_by_locale\lv-
LV_train.txt
Saved ms-MY utterances from train partition to utts_by_locale\ms-
MY_train.txt
Saved nb-NO utterances from train partition to utts_by_locale\nb-
```

```
NO_train.txt
Saved nl-NL utterances from train partition to utts_by_locale\nl-
NL_train.txt
Saved pl-PL utterances from train partition to utts_by_locale\pl-
PL_train.txt
Saved pt-PT utterances from train partition to utts_by_locale\pt-
PT_train.txt
Saved ro-RO utterances from train partition to utts_by_locale\ro-
RO_train.txt
Saved ru-RU utterances from train partition to utts_by_locale\ru-
RU_train.txt
Saved sl-SL utterances from train partition to utts_by_locale\sl-
SL_train.txt
Saved sv-SE utterances from train partition to utts_by_locale\sv-
SE_train.txt
Saved sq-AL utterances from train partition to utts_by_locale\sq-
AL_train.txt
Saved sw-KE utterances from train partition to utts_by_locale\sw-
KE_train.txt
Saved tl-PH utterances from train partition to utts_by_locale\tl-
PH_train.txt
Saved tr-TR utterances from train partition to utts_by_locale\tr-
TR_train.txt
Saved vi-VN utterances from train partition to utts_by_locale\vi-
VN_train.txt
Saved cy-GB utterances from train partition to utts_by_locale\cy-
GB_train.txt
Saved af-ZA utterances from validation partition to utts_by_locale\af-
ZA_validation.txt
Saved da-DK utterances from validation partition to utts_by_locale\da-
DK_validation.txt
Saved de-DE utterances from validation partition to utts_by_locale\de-
DE_validation.txt
Saved en-US utterances from validation partition to utts_by_locale\en-
US_validation.txt
Saved es-ES utterances from validation partition to utts_by_locale\es-
ES_validation.txt
Saved fr-FR utterances from validation partition to utts_by_locale\fr-
FR_validation.txt
Saved fi-FI utterances from validation partition to utts_by_locale\fi-
FI_validation.txt
Saved hu-HU utterances from validation partition to utts_by_locale\hu-
HU_validation.txt
Saved is-IS utterances from validation partition to utts_by_locale\is-
IS_validation.txt
Saved it-IT utterances from validation partition to utts_by_locale\it-
IT_validation.txt
Saved jv-ID utterances from validation partition to utts_by_locale\jv-
ID_validation.txt
```

```
Saved lv-LV utterances from validation partition to utts_by_locale\lv-
LV_validation.txt
Saved ms-MY utterances from validation partition to utts_by_locale\ms-
MY_validation.txt
Saved nb-NO utterances from validation partition to utts_by_locale\nb-
NO_validation.txt
Saved nl-NL utterances from validation partition to utts_by_locale\nl-
NL_validation.txt
Saved pl-PL utterances from validation partition to utts_by_locale\pl-
PL_validation.txt
Saved pt-PT utterances from validation partition to utts_by_locale\pt-
PT_validation.txt
Saved ro-RO utterances from validation partition to utts_by_locale\ro-
RO_validation.txt
Saved ru-RU utterances from validation partition to utts_by_locale\ru-
RU_validation.txt
Saved sl-SL utterances from validation partition to utts_by_locale\sl-
SL_validation.txt
Saved sv-SE utterances from validation partition to utts_by_locale\sv-
SE_validation.txt
Saved sq-AL utterances from validation partition to utts_by_locale\sq-
AL_validation.txt
Saved sw-KE utterances from validation partition to utts_by_locale\sw-
KE_validation.txt
Saved tl-PH utterances from validation partition to utts_by_locale\tl-
PH_validation.txt
Saved tr-TR utterances from validation partition to utts_by_locale\tr-
TR_validation.txt
Saved vi-VN utterances from validation partition to utts_by_locale\vi-
VN_validation.txt
Saved cy-GB utterances from validation partition to utts_by_locale\cy-
GB_validation.txt
Saved af-ZA utterances from test partition to utts_by_locale\af-
ZA_test.txt
Saved da-DK utterances from test partition to utts_by_locale\da-
DK_test.txt
Saved de-DE utterances from test partition to utts_by_locale\de-
DE_test.txt
Saved en-US utterances from test partition to utts_by_locale\en-
US_test.txt
Saved es-ES utterances from test partition to utts_by_locale\es-
ES_test.txt
Saved fr-FR utterances from test partition to utts_by_locale\fr-
FR_test.txt
Saved fi-FI utterances from test partition to utts_by_locale\fi-
FI_test.txt
Saved hu-HU utterances from test partition to utts_by_locale\hu-
HU_test.txt
Saved is-IS utterances from test partition to utts_by_locale\is-
```

```
IS_test.txt
Saved it-IT utterances from test partition to utts_by_locale\it-
IT_test.txt
Saved jv-ID utterances from test partition to utts_by_locale\jv-
ID_test.txt
Saved lv-LV utterances from test partition to utts_by_locale\lv-
LV_test.txt
Saved ms-MY utterances from test partition to utts_by_locale\ms-
MY_test.txt
Saved nb-NO utterances from test partition to utts_by_locale\nb-
NO_test.txt
Saved nl-NL utterances from test partition to utts_by_locale\nl-
NL_test.txt
Saved pl-PL utterances from test partition to utts_by_locale\pl-
PL_test.txt
Saved pt-PT utterances from test partition to utts_by_locale\pt-
PT_test.txt
Saved ro-RO utterances from test partition to utts_by_locale\ro-
RO_test.txt
Saved ru-RU utterances from test partition to utts_by_locale\ru-
RU_test.txt
Saved sl-SL utterances from test partition to utts_by_locale\sl-
SL_test.txt
Saved sv-SE utterances from test partition to utts_by_locale\sv-
SE_test.txt
Saved sq-AL utterances from test partition to utts_by_locale\sq-
AL_test.txt
Saved sw-KE utterances from test partition to utts_by_locale\sw-
KE_test.txt
Saved tl-PH utterances from test partition to utts_by_locale\tl-
PH_test.txt
Saved tr-TR utterances from test partition to utts_by_locale\tr-
TR_test.txt
Saved vi-VN utterances from test partition to utts_by_locale\vi-
VN_test.txt
Saved cy-GB utterances from test partition to utts_by_locale\cy-
GB_test.txt
```

Task 2

Create train,val and test sets for Multinomial Naive Bayes Model

```
# Initialize lists for filtered data and labels for each partition
filtered_train_data, filtered_train_labels = [], []
filtered_val_data, filtered_val_labels = [], []
filtered_test_data, filtered_test_labels = [], []

# Loop through all the files in the output directory and filter by
locale and partition
```

```
for filename in os.listdir(output_dir):
    if filename.endswith(".txt"):
        file_path = os.path.join(output_dir, filename)

        # Extract the locale and partition from the filename
        locale, partition = filename.split('_')[:2]

        # Only process if it's in the desired locales
        if locale in locales:
            with open(file_path, 'r', encoding='utf-8') as file:
                utts = file.readlines()

            # Strip newlines from utterances
            utts = [utt.strip() for utt in utts]

            # Append filtered data and labels based on the partition
            if partition == 'train.txt':
                filtered_train_data.extend(utts)
                filtered_train_labels.extend([locale] * len(utts))
            elif partition == 'validation.txt':
                filtered_val_data.extend(utts)
                filtered_val_labels.extend([locale] * len(utts))
            elif partition == 'test.txt':
                filtered_test_data.extend(utts)
                filtered_test_labels.extend([locale] * len(utts))

# Feature extraction (TF-IDF vectorization)
vectorizer = TfidfVectorizer()

# Vectorize the train, validation, and test data
X_train = vectorizer.fit_transform(filtered_train_data)
y_train = filtered_train_labels

X_val = vectorizer.transform(filtered_val_data)
y_val = filtered_val_labels

X_test = vectorizer.transform(filtered_test_data)
y_test = filtered_test_labels
```

Get the hyperparameter alpha tuned properly by using validation set

```
nb_model = MultinomialNB()        # Multinomial Naive Bayes Model

# Grid search to tune 'alpha'
param_grid = {'alpha': [0.01, 0.033, 0.05, 0.067, 0.1]}
grid_search = GridSearchCV(nb_model, param_grid, cv=5,
scoring='accuracy')
grid_search.fit(X_train, y_train)

# Training model
```

```
nb_model.fit(X_train, y_train)
best_nb_model = grid_search.best_estimator_
print(best_nb_model.get_params())

{'alpha': 0.05, 'class_prior': None, 'fit_prior': True, 'force_alpha':
'warn'}
```

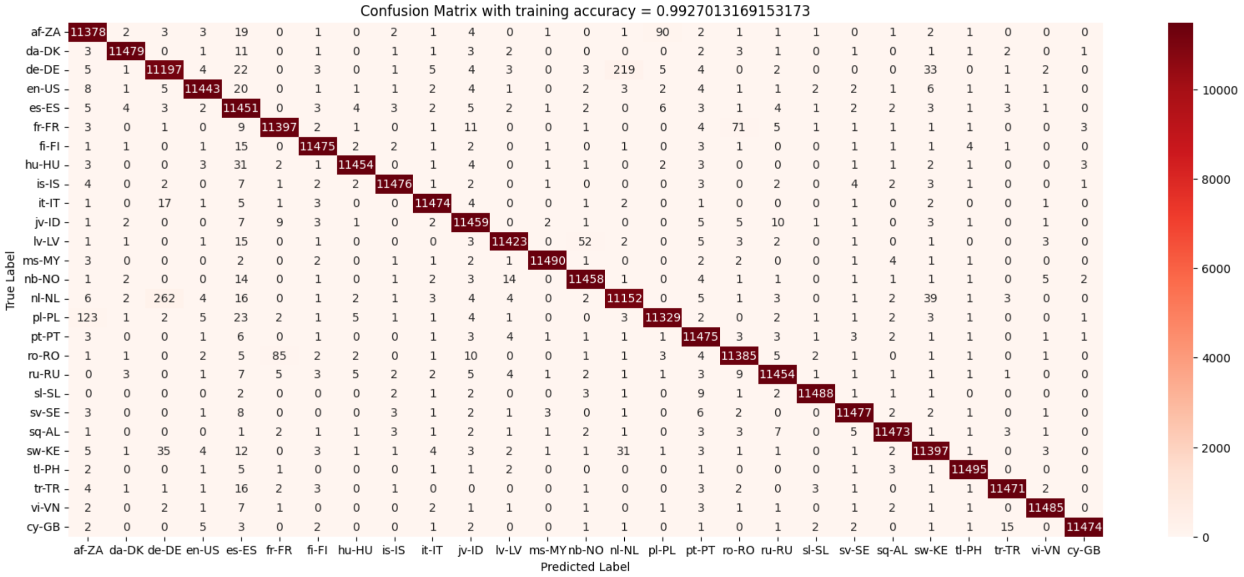
Performance Metrics

```
# Evaluate on training set
y_train_pred = best_nb_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
train_cm = confusion_matrix(y_train, y_train_pred)
print(f"Training Accuracy: {train_accuracy}")

plt.figure(figsize=(20, 8))
sns.heatmap(train_cm, annot=True, fmt='d', cmap='Reds',
xticklabels=locales, yticklabels=locales)

plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title(f'Confusion Matrix with training accuracy =
{train_accuracy}')
plt.show()

Training Accuracy: 0.9927013169153173
```



	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
af-ZA	11378	2	3	3	19	0	1	0	2	1	4	0	1	0	1	90	2	1	1	1	0	1	2	1	0	0	0
da-DK	3	11479	0	1	11	0	1	0	1	1	3	2	0	0	0	0	2	3	1	0	1	0	1	1	2	0	1
de-DE	5	1	11197	4	22	0	3	0	1	5	4	3	0	3	219	5	4	0	2	0	0	0	33	0	1	2	0
en-US	8	1	5	11443	20	0	1	1	1	2	4	1	0	2	3	2	4	1	1	2	2	1	6	1	1	1	0
es-ES	5	4	3	2	11451	0	3	4	3	2	5	2	1	2	0	6	3	1	4	1	2	2	3	1	3	1	0
fr-FR	3	0	1	0	9	11397	2	1	0	1	11	0	0	1	0	0	4	71	5	1	1	1	1	1	0	0	3
fi-FI	1	1	0	1	15	0	11475	2	2	1	2	0	1	0	1	0	3	1	0	0	1	1	1	4	1	0	0
hu-HU	3	0	0	3	31	2	1	11454	0	1	4	0	1	1	0	2	3	0	0	0	1	1	2	1	0	0	3
is-IS	4	0	2	0	7	1	2	2	11476	1	2	0	1	0	0	0	3	0	2	0	4	2	3	1	0	0	1
it-IT	1	0	17	1	5	1	3	0	0	11474	4	0	0	1	2	0	1	0	0	0	1	0	2	0	0	1	0
jv-ID	1	2	0	0	7	9	3	1	0	2	11459	0	2	1	0	0	5	5	10	1	1	0	3	1	0	1	0
lv-LV	1	1	0	1	15	0	1	0	0	0	3	11423	0	52	2	0	5	3	2	0	1	0	1	0	0	3	0
ms-MY	3	0	0	0	2	0	2	0	1	1	2	1	11490	1	0	0	2	2	0	0	1	4	1	1	0	0	0
nb-NO	1	2	0	0	14	0	1	0	1	2	3	14	0	11458	1	0	4	1	1	0	1	1	1	1	0	5	2
nl-NL	6	2	262	4	16	0	1	2	1	3	4	4	0	2	11152	0	5	1	3	0	1	2	39	1	3	0	0
pl-PL	123	1	2	5	23	2	1	5	1	1	4	1	0	0	3	11329	2	0	2	1	1	2	3	1	0	0	1
pt-PT	3	0	0	1	6	0	1	0	0	1	3	4	1	1	1	1	11475	3	3	1	3	2	1	1	0	1	1
ro-RO	1	1	0	2	5	85	2	2	0	1	10	0	0	1	1	3	4	11385	5	2	1	0	1	1	0	1	0
ru-RU	0	3	0	1	7	5	3	5	2	2	5	4	1	2	1	1	3	9	11454	1	1	1	1	1	1	0	0
sl-SL	0	0	0	0	2	0	0	0	2	1	2	0	0	3	1	0	9	1	2	11488	1	1	1	0	0	0	0
sv-SE	3	0	0	1	8	0	0	0	3	1	2	1	3	0	1	0	6	2	0	0	11477	2	2	1	0	1	0
sq-AL	1	0	0	0	1	2	1	1	3	1	2	1	1	2	1	0	3	3	7	0	5	11473	1	1	3	1	0
sw-KE	5	1	35	4	12	0	3	1	1	4	3	2	1	1	31	1	3	1	1	0	1	2	11397	1	0	3	0
tl-PH	2	0	0	1	5	1	0	0	0	1	1	2	0	0	0	0	1	0	0	0	1	3	1	11495	0	0	0
tr-TR	4	1	1	1	16	2	3	0	1	0	0	0	0	1	0	0	3	2	0	3	1	0	1	1	11471	2	0
vi-VN	2	0	2	1	7	1	0	0	0	2	1	1	0	1	0	1	3	1	1	0	1	2	1	1	0	11485	0
cy-GB	2	0	0	5	3	0	2	0	0	1	2	0	0	1	1	0	1	0	1	2	2	0	1	1	15	0	11474

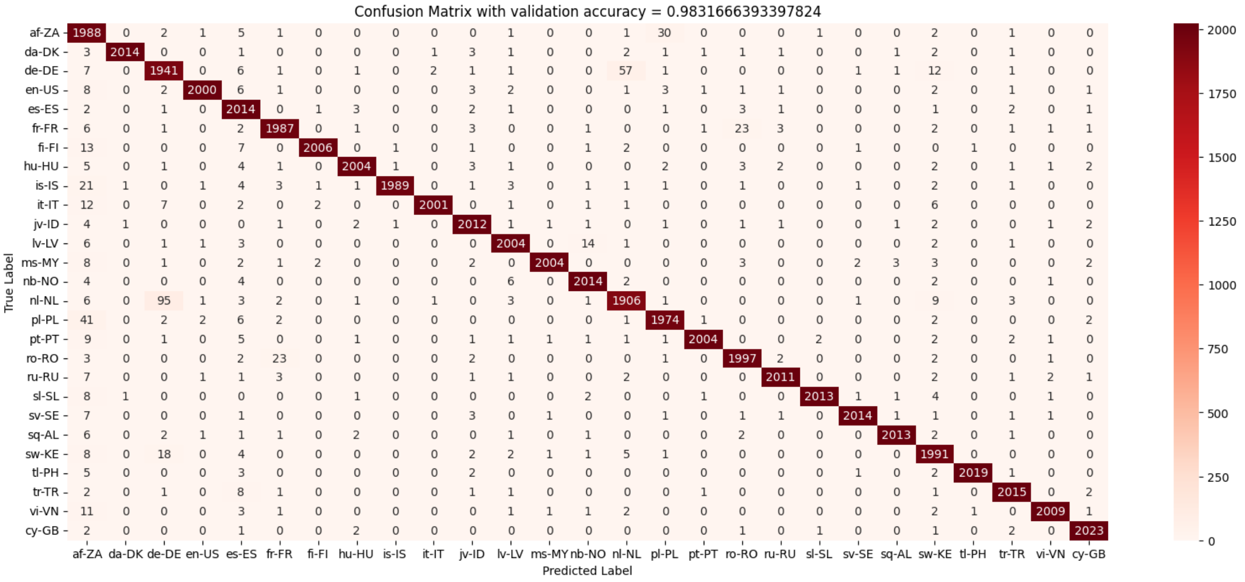
```
# Evaluate on validation set
y_val_pred = best_nb_model.predict(X_val)
val_accuracy = accuracy_score(y_val, y_val_pred)
val_cm = confusion_matrix(y_val, y_val_pred)
```

```
print(f"Validation Accuracy: {val_accuracy}")

plt.figure(figsize=(20, 8))
sns.heatmap(val_cm, annot=True, fmt='d', cmap='Reds',
xticklabels=locales, yticklabels=locales)

plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title(f'Confusion Matrix with validation accuracy =
{val_accuracy}')
plt.show()

Validation Accuracy: 0.9831666393397824
```



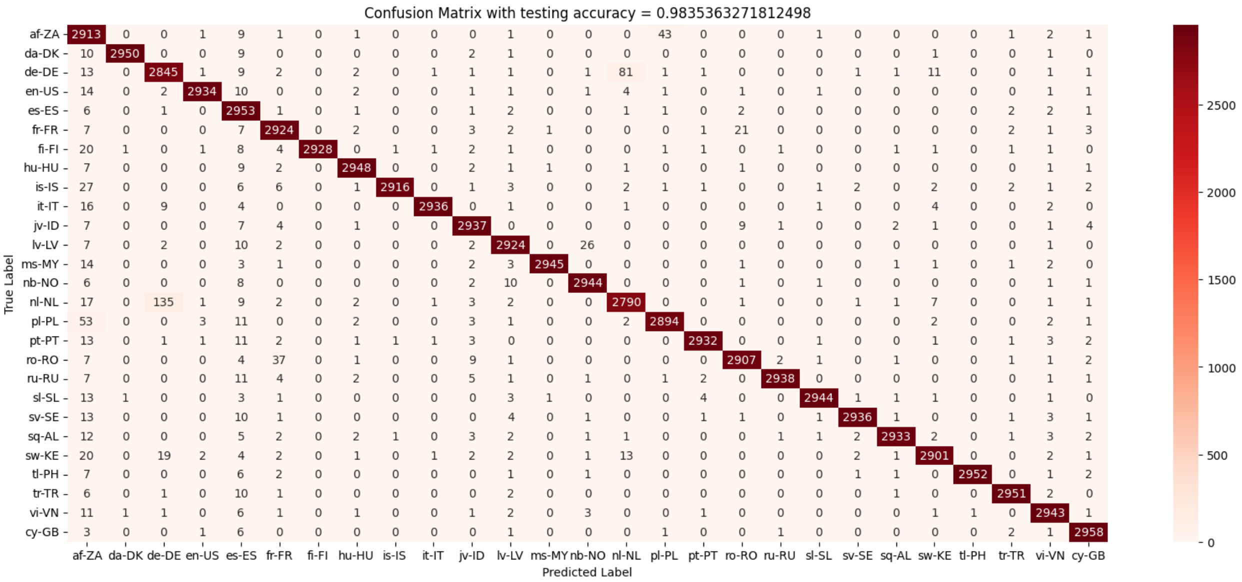
	af-ZA	da-DK	de-DE	en-US	es-ES	fr-FR	fi-FI	hu-HU	is-IS	it-IT	jv-ID	lv-LV	ms-MY	nb-NO	nl-NL	pl-PL	pt-PT	ro-RO	ru-RU	sl-SL	sv-SE	sq-AL	sw-KE	tl-PH	tr-TR	vi-VN	cy-GB
af-ZA	1988	0	2	1	5	1	0	0	0	0	0	1	0	0	1	30	0	0	0	1	0	0	2	0	1	0	0
da-DK	3	2014	0	0	1	0	0	0	0	1	3	1	0	0	2	1	1	1	1	0	0	1	2	0	1	0	0
de-DE	7	0	1941	0	6	1	0	1	0	2	1	1	0	0	57	1	0	0	0	0	1	1	12	0	1	0	0
en-US	8	0	2	2000	6	1	0	0	0	0	3	2	0	0	1	3	1	1	1	0	0	0	2	0	1	0	1
es-ES	2	0	1	0	2014	0	1	3	0	0	2	1	0	0	0	1	0	3	1	0	0	0	1	0	2	0	1
fr-FR	6	0	1	0	2	1987	0	1	0	0	3	0	0	1	0	0	1	23	3	0	0	0	2	0	1	1	1
fi-FI	13	0	0	0	7	0	2006	0	1	0	1	0	0	1	2	0	0	0	0	0	1	0	0	1	0	0	0
hu-HU	5	0	1	0	4	1	0	2004	1	0	3	1	0	0	0	2	0	3	2	0	0	0	2	0	1	1	2
is-IS	21	1	0	1	4	3	1	1	1989	0	1	3	0	1	1	1	0	1	0	0	1	0	2	0	1	0	0
it-IT	12	0	7	0	2	0	2	0	0	2001	0	1	0	1	1	0	0	0	0	0	0	0	6	0	0	0	0
jv-ID	4	1	0	0	0	1	0	2	1	0	2012	1	1	1	0	1	0	1	1	0	0	1	2	0	0	1	2
lv-LV	6	0	1	1	3	0	0	0	0	0	0	2004	0	14	1	0	0	0	0	0	0	0	2	0	1	0	0
ms-MY	8	0	1	0	2	1	2	0	0	0	2	0	2004	0	0	0	0	3	0	0	2	3	3	0	0	0	2
nb-NO	4	0	0	0	4	0	0	0	0	0	0	6	0	2014	2	0	0	0	0	0	0	0	2	0	0	1	0
nl-NL	6	0	95	1	3	2	0	1	0	1	0	3	0	1	1906	1	0	0	0	0	1	0	9	0	3	0	0
pl-PL	41	0	2	2	6	2	0	0	0	0	0	0	0	0	1	1974	1	0	0	0	0	0	2	0	0	0	2
pt-PT	9	0	1	0	5	0	0	1	0	0	1	1	1	1	1	1	2004	0	0	2	0	0	2	0	2	1	0
ro-RO	3	0	0	0	2	23	0	0	0	0	2	0	0	0	0	1	0	1997	2	0	0	0	2	0	0	1	0
ru-RU	7	0	0	1	1	3	0	0	0	0	1	1	0	0	2	0	0	0	2011	0	0	0	2	0	1	2	1
sl-SL	8	1	0	0	0	0	0	1	0	0	0	0	0	2	0	0	1	0	0	2013	1	1	4	0	0	1	0
sv-SE	7	0	0	0	1	0	0	0	0	0	3	0	1	0	0	1	0	1	1	0	2014	1	1	0	1	1	0
sq-AL	6	0	2	1	1	1	0	2	0	0	0	1	0	1	0	0	0	2	0	0	0	2013	2	0	1	0	0
sw-KE	8	0	18	0	4	0	0	0	0	0	2	2	1	1	5	1	0	0	0	0	0	0	1991	0	0	0	0
tl-PH	5	0	0	0	3	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	2	2019	1	0	0
tr-TR	2	0	1	0	8	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	1	0	2015	0	2
vi-VN	11	0	0	0	3	1	0	0	0	0	0	1	1	1	2	0	0	0	0	0	0	0	2	1	0	2009	1
cy-GB	2	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	2	0	2023

```
# Evaluate on test set
y_test_pred = best_nb_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
test_cm = confusion_matrix(y_test, y_test_pred)
print(f"Testing Accuracy: {test_accuracy}")

plt.figure(figsize=(20, 8))
sns.heatmap(test_cm, annot=True, fmt='d', cmap='Reds',
xticklabels=locales, yticklabels=locales)

plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title(f'Confusion Matrix with testing accuracy = {test_accuracy}')
plt.show()

Testing Accuracy: 0.9835363271812498
```



	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
af-ZA	2913	0	0	1	9	1	0	1	0	0	0	1	0	0	0	43	0	0	0	1	0	0	0	0	1	2	1
da-DK	10	2950	0	0	9	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
de-DE	13	0	2845	1	9	2	0	2	0	1	1	1	0	1	81	1	1	0	0	0	1	1	11	0	0	1	1
en-US	14	0	2	2934	10	0	0	2	0	0	1	1	0	1	4	1	0	1	0	1	0	0	0	0	0	1	1
es-ES	6	0	1	0	2953	1	0	1	0	0	1	2	0	0	1	1	0	2	0	0	0	0	0	0	2	2	1
fr-FR	7	0	0	0	7	2924	0	2	0	0	3	2	1	0	0	0	1	21	0	0	0	0	0	0	2	1	3
fi-FI	20	1	0	1	8	4	2928	0	1	1	2	1	0	0	0	1	1	0	1	0	0	1	1	0	1	1	0
hu-HU	7	0	0	0	9	2	0	2948	0	0	2	1	1	0	1	0	0	1	0	0	0	0	0	0	0	1	1
is-IS	27	0	0	0	6	6	0	1	2916	0	1	3	0	0	2	1	1	0	0	1	2	0	2	0	2	1	2
it-IT	16	0	9	0	4	0	0	0	0	2936	0	1	0	0	1	0	0	0	0	1	0	0	4	0	0	2	0
jv-ID	7	0	0	0	7	4	0	1	0	0	2937	0	0	0	0	0	0	9	1	0	0	2	1	0	0	1	4
lv-LV	7	0	2	0	10	2	0	0	0	0	2	2924	0	26	0	0	0	0	0	0	0	0	0	0	0	1	0
ms-MY	14	0	0	0	3	1	0	0	0	0	2	3	2945	0	0	0	0	1	0	0	0	1	1	0	1	2	0
nb-NO	6	0	0	0	8	0	0	0	0	0	2	10	0	2944	0	0	0	1	0	1	0	0	0	0	0	1	1
nl-NL	17	0	135	1	9	2	0	2	0	1	3	2	0	0	2790	0	0	1	0	0	1	1	7	0	0	1	1
pl-PL	53	0	0	3	11	0	0	2	0	0	3	1	0	0	2	2894	0	0	0	0	0	0	2	0	0	2	1
pt-PT	13	0	1	1	11	2	0	1	1	1	3	0	0	0	0	0	2932	0	0	1	0	0	1	0	1	3	2
ro-RO	7	0	0	0	4	37	0	1	0	0	9	1	0	0	0	0	0	2907	2	1	0	1	0	0	1	1	2
ru-RU	7	0	0	0	11	4	0	2	0	0	5	1	0	1	0	1	2	0	2938	0	0	0	0	0	0	1	1
sl-SL	13	1	0	0	3	1	0	0	0	0	0	3	1	0	0	0	4	0	0	2944	1	1	1	0	0	1	0
sv-SE	13	0	0	0	10	1	0	0	0	0	0	4	0	1	0	0	1	1	0	1	2936	1	0	0	1	3	1
sq-AL	12	0	0	0	5	2	0	2	1	0	3	2	0	1	1	0	0	0	1	1	2	2933	2	0	1	3	2
sw-KE	20	0	19	2	4	2	0	1	0	1	2	2	0	1	13	0	0	0	0	0	2	1	2901	0	0	2	1
tl-PH	7	0	0	0	6	2	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	2952	0	1	2
tr-TR	6	0	1	0	10	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0	2951	2	0
vi-VN	11	1	1	0	6	1	0	1	0	0	1	2	0	3	0	0	1	0	0	0	0	0	1	1	0	2943	1
cy-GB	3	0	0	1	6	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	2	1	2958

Task 3

Create a continent mapping for languages

```python
# Define the mapping of languages to continents
continent_mapping = {
    'af-ZA': 'Africa',
    'da-DK': 'Europe',
    'de-DE': 'Europe',
    'en-US': 'North_America',
    'es-ES': 'Europe',
    'fr-FR': 'Europe',
    'fi-FI': 'Europe',
    'hu-HU': 'Europe',
    'is-IS': 'Europe',
    'it-IT': 'Europe',
    'jv-ID': 'Asia',
    'lv-LV': 'Europe',
    'ms-MY': 'Asia',
    'nb-NO': 'Europe',
    'nl-NL': 'Europe',
    'pl-PL': 'Europe',
    'pt-PT': 'Europe',
    'ro-RO': 'Europe',
    'ru-RU': 'Europe',
    'sl-SL': 'Europe',
    'sv-SE': 'Europe',
    'sq-AL': 'Europe',
    'sw-KE': 'Africa',
    'tl-PH': 'Asia',
    'tr-TR': 'Asia',
    'vi-VN': 'Asia',
    'cy-GB': 'Europe'
}

# Dictionary to hold content for continent
continent_files = {continent: [] for continent in
set(continent_mapping.values())}

# Read each language file and append its content to the corresponding
continent list
partitions = ['train', 'validation', 'test']
for partition in partitions:
    for lang_code, continent in continent_mapping.items():
        file_path = os.path.join(output_dir,
f"{lang_code}_{partition}.txt")

        # Check if the file exists before reading
        if os.path.exists(file_path):
            with open(file_path, 'r', encoding='utf-8') as file:
                content = file.readlines()
                continent_files[continent].extend(content)
        else:
            print(f"File {file_path} does not exist.")
```

Save the txt files as per continents

```
# Save content for continents in new locations
lang_dir = 'utts_by_continents'
os.makedirs(lang_dir, exist_ok=True)

for continent, lines in continent_files.items():
    output_file_path = os.path.join(lang_dir, f"{continent}.txt")
    with open(output_file_path, 'w', encoding='utf-8') as output_file:
        output_file.writelines(lines)                # Write all lines
to the continent file
```

Traverse throught the continent folder to get the data in pandas dataframe for the model to operate upon

```
folder_path = output_dir

# List to store DataFrames
train_dfs = []
val_dfs = []
test_dfs = []

# Function to extract the locale from the file name
def extract_locale(file_name):
    return file_name.split('_')[0]
```

```
for file_name in os.listdir(folder_path):
    file_path = os.path.join(folder_path, file_name)

    locale = extract_locale(file_name)  # Extract the locale (e.g.,
'af-ZA')

    if '_train.txt' in file_name:
        df = pd.read_csv(file_path, sep='\t', names=['utt'])
        df['locale'] = locale  # Add locale column
        train_dfs.append(df)

    elif '_validation.txt' in file_name:
        df = pd.read_csv(file_path, sep='\t', names=['utt'])
        df['locale'] = locale  # Add locale column
        val_dfs.append(df)

    elif '_test.txt' in file_name:
        df = pd.read_csv(file_path, sep='\t', names=['utt'])
        df['locale'] = locale  # Add locale column
        test_dfs.append(df)

# Concatenate all DataFrames into one
train_data = pd.concat(train_dfs, ignore_index=True)
validation_data = pd.concat(val_dfs, ignore_index=True)
test_data = pd.concat(test_dfs, ignore_index=True)
```

Vectorize the data and select only relevant features to be trained upon

```
y_train_continent = train_data['locale'].map(continent_mapping)

# Encode target labels as integers for chi-square test
label_encoder = LabelEncoder()
y_train_continent_encoded =
label_encoder.fit_transform(y_train_continent)

# Vectorizer setup (same as before)
vectorizer = TfidfVectorizer(max_features=50000,  # Adjust as needed
                             max_df=0.7,  # Ignore terms that appear
in more than 80% of the documents
                             )

# Fit and transform the training data
X_train_tfidf = vectorizer.fit_transform(train_data['utt'])

# Perform chi-square feature selection
chi2_selector = SelectKBest(chi2, k=1000)  # Choose top 5000 features
based on chi-square test
X_train_selected = chi2_selector.fit_transform(X_train_tfidf,
y_train_continent_encoded)
```

```
# Map validation labels to the continent
y_val_continent = validation_data['locale'].map(continent_mapping)
y_val_continent_encoded = label_encoder.transform(y_val_continent)

# Transform the validation data using the same vectorizer and feature
selector
X_val_tfidf = vectorizer.transform(validation_data['utt'])
X_val_selected = chi2_selector.transform(X_val_tfidf)

# Map test labels to the continent
y_test_continent = test_data['locale'].map(continent_mapping)
y_test_continent_encoded = label_encoder.transform(y_test_continent)

# Transform the test data using the same vectorizer and feature
selector
X_test_tfidf = vectorizer.transform(test_data['utt'])
X_test_selected = chi2_selector.transform(X_test_tfidf)
```

Create the class for RDA using LDA and QDA

```
class RegularizedDiscriminantAnalysis(BaseEstimator, ClassifierMixin):
    def __init__(self, lambda_):
        self.lambda_ = lambda_
        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):
        # Combine predictions from LDA and QDA
        lda_prob = self.lda.predict_proba(X)
        qda_prob = self.qda.predict_proba(X)

        # Weighted average of probabilities of LDA and QDA
        combined_prob = self.lambda_ * lda_prob + (1 - self.lambda_) *
qda_prob

        # Choose the class with the highest probability
        return np.argmax(combined_prob, axis=1)
```

Fit the model on the data with an apt lambda value

```
lambda_range = np.linspace(0, 1, 11)  # Search range for lambda values

best_lambda = None
```

```
best_score = 0

# Loop over lambda values and perform validation
for lambda_ in lambda_range:
    rda = RegularizedDiscriminantAnalysis(lambda_=lambda_)
    rda.fit(X_train_selected.toarray(), y_train_continent_encoded)

    # Predict on the validation data
    y_val_pred = rda.predict(X_val_selected.toarray())

    val_score = accuracy_score(y_val_continent_encoded, y_val_pred)
    if val_score > best_score:
        best_score = val_score
        best_lambda = lambda_

# Output the best lambda value
print(f"Best lambda: {best_lambda}, Best validation accuracy: 
{best_score}")
```

```
c:\Users\Nikshay Jain\AppData\Local\Programs\Python\Python38\lib\site-
packages\sklearn\discriminant_analysis.py:926: UserWarning: Variables
are collinear
  warnings.warn("Variables are collinear")
c:\Users\Nikshay Jain\AppData\Local\Programs\Python\Python38\lib\site-
packages\sklearn\discriminant_analysis.py:926: UserWarning: Variables
are collinear
  warnings.warn("Variables are collinear")
c:\Users\Nikshay Jain\AppData\Local\Programs\Python\Python38\lib\site-
packages\sklearn\discriminant_analysis.py:926: UserWarning: Variables
are collinear
  warnings.warn("Variables are collinear")
c:\Users\Nikshay Jain\AppData\Local\Programs\Python\Python38\lib\site-
packages\sklearn\discriminant_analysis.py:926: UserWarning: Variables
are collinear
  warnings.warn("Variables are collinear")
```

```
rda = RegularizedDiscriminantAnalysis(lambda_=best_lambda)
rda.fit(X_train_selected.toarray(), y_train_continent_encoded)
```

```
c:\Users\Nikshay Jain\AppData\Local\Programs\Python\Python38\lib\site-
packages\sklearn\discriminant_analysis.py:926: UserWarning: Variables
are collinear
  warnings.warn("Variables are collinear")
```

```
RegularizedDiscriminantAnalysis(lambda_=0.7)
```

Output the metrics for the optimum model

```
# Training accuracy
y_train_continent_pred = rda.predict(X_train_selected.toarray())
```

```
train_accuracy_continent = accuracy_score(y_train_continent_encoded,
y_train_continent_pred)
print(f"Training Accuracy: {train_accuracy_continent:.4f}")

# Validation accuracy
y_val_continent_pred = rda.predict(X_val_selected.toarray())
val_accuracy_continent = accuracy_score(y_val_continent_encoded,
y_val_continent_pred)
print(f"Validation Accuracy: {val_accuracy_continent:.4f}")

# Test accuracy
y_test_continent_pred = rda.predict(X_test_selected.toarray())
test_accuracy_continent = accuracy_score(y_test_continent_encoded,
y_test_continent_pred)
print(f"Test Accuracy: {test_accuracy_continent:.4f}")
```

```
Training Accuracy: 0.9515
Validation Accuracy: 0.9492
Test Accuracy: 0.9500
```