DA24M011

September 26, 2024

#####

DA5401 Data Analytics Labarotary

#####

Assignment 6 - Submitted by: DA24M011 - Nandhakishore C S

Task 1 [40 Points] Given the dataset (Amazon's MASSIVE dataset) - has data from 51 Languages in the latest version. For the given question, we are downloading the 27 files corresponding to languages which use the roman / latin script.

Importing Libraries

[1]:
```
import os
import numpy as np
from datasets import load_dataset
from unidecode import unidecode # type: ignore

import unicodedata
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay, accuracy_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import pandas as pd
```

0.0.1 TASK 1

Defining the languages which use Latin Script and downloading the corresponsing files from the main dataset.

[2]:
```
# Define the locales we are interested in (i.e.) the languages which use Latin␣
 ↪Script
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'
]
```

[3]:
```
# from datasets import load_dataset - using Huggingface load dataset module /␣
↪function
df_raw = load_dataset("AmazonScience/massive", "all")
```

[4]:
```
df_raw
```

[4]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'locale', 'partition', 'scenario', 'intent', 'utt',
'annot_utt', 'worker_id', 'slot_method', 'judgments'],
        num_rows: 587214
    })
    validation: Dataset({
        features: ['id', 'locale', 'partition', 'scenario', 'intent', 'utt',
'annot_utt', 'worker_id', 'slot_method', 'judgments'],
        num_rows: 103683
    })
    test: Dataset({
        features: ['id', 'locale', 'partition', 'scenario', 'intent', 'utt',
'annot_utt', 'worker_id', 'slot_method', 'judgments'],
        num_rows: 151674
    })
})
```

[5]:
```
# Downloading and storing datafiles for 27 languages

output_dir = 'language_files'
os.makedirs(output_dir, exist_ok=True)

# Extract utterances for each locale
for locale in locales:
    file_path = os.path.join(output_dir, f'{locale}.txt')
    with open(file_path, 'w', encoding='utf-8') as f:
        for partition in ['train', 'validation', 'test']:
                for example in df_raw[partition]:
                        if example['locale'] == locale:
                                utt = example['utt']
                                f.write(utt + '\n')
```

0.0.2 TASK 2

The dataset is split into three parttions:

1. Train
2. Validation

3. Test

From the training partition, we get the 'utt' column as the input features (the sentences with / without) accents and the corresponsing country name in the 'locale' column as the label. Instead of using the tokens from the dataset, Tokens are generated using CountVectoriser from sklearn's feature extraction module.

Using '.filter' function to get the data corresponding to the country names

[6]:
```
df_filtered = df_raw.filter(lambda x: x['locale'] in locales)
df_filtered
```

[6]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'locale', 'partition', 'scenario', 'intent', 'utt',
'annot_utt', 'worker_id', 'slot_method', 'judgments'],
        num_rows: 310878
    })
    validation: Dataset({
        features: ['id', 'locale', 'partition', 'scenario', 'intent', 'utt',
'annot_utt', 'worker_id', 'slot_method', 'judgments'],
        num_rows: 54891
    })
    test: Dataset({
        features: ['id', 'locale', 'partition', 'scenario', 'intent', 'utt',
'annot_utt', 'worker_id', 'slot_method', 'judgments'],
        num_rows: 80298
    })
})
```

Building Multinomial Naive Bayes Model - without removing accents

[7]:
```
# No accent removal
df1_train = df_filtered['train']
df1_val = df_filtered['validation']
df1_test = df_filtered['test']
```

[8]:
```
pipeline = Pipeline([
    ('vectoriser', CountVectorizer()),
    ('classifier', MultinomialNB())
])

pipeline.fit(df1_train['utt'], df1_train['locale'])
```

[8]:
```
Pipeline(steps=[('vectoriser', CountVectorizer()),
                ('classifier', MultinomialNB())])
```

Model's perfomance metrics for Train Partition

```
[9]: train_predictions = pipeline.predict(df1_train['utt'])
print("Training Data Performance:")
print(classification_report(df1_train['locale'], train_predictions))
print("Training Accuracy:", accuracy_score(df1_train['locale'],␣
 ↪train_predictions))
```

```
Training Data Performance:
              precision    recall  f1-score   support

       af-ZA       0.98      0.98      0.98     11514
       cy-GB       1.00      1.00      1.00     11514
       da-DK       0.97      0.97      0.97     11514
       de-DE       1.00      0.99      0.99     11514
       en-US       0.96      0.99      0.98     11514
       es-ES       0.99      0.99      0.99     11514
       fi-FI       1.00      0.99      0.99     11514
       fr-FR       0.99      0.99      0.99     11514
       hu-HU       1.00      0.99      1.00     11514
       is-IS       1.00      1.00      1.00     11514
       it-IT       0.99      0.99      0.99     11514
       jv-ID       0.99      0.99      0.99     11514
       lv-LV       1.00      1.00      1.00     11514
       ms-MY       0.99      0.99      0.99     11514
       nb-NO       0.97      0.96      0.97     11514
       nl-NL       0.99      0.98      0.98     11514
       pl-PL       0.99      0.99      0.99     11514
       pt-PT       0.99      0.99      0.99     11514
       ro-RO       1.00      0.99      0.99     11514
       ru-RU       1.00      1.00      1.00     11514
       sl-SL       1.00      0.99      1.00     11514
       sq-AL       1.00      0.99      1.00     11514
       sv-SE       0.99      0.99      0.99     11514
       sw-KE       1.00      1.00      1.00     11514
       tl-PH       1.00      1.00      1.00     11514
       tr-TR       1.00      0.99      1.00     11514
       vi-VN       1.00      1.00      1.00     11514

    accuracy                           0.99    310878
   macro avg       0.99      0.99      0.99    310878
weighted avg       0.99      0.99      0.99    310878

Training Accuracy: 0.9909160506693945
```

Model's perfomance metrics for Validation Partition

```
[10]: val_predictions = pipeline.predict(df1_val['utt'])
print("Validation Data Performance:")
print(classification_report(df1_val['locale'], val_predictions))
```

```
print("Validation Accuracy:", accuracy_score(df1_val['locale'],␣
↪val_predictions))
```

```
Validation Data Performance:
              precision    recall  f1-score   support

       af-ZA       0.91      0.98      0.94      2033
       cy-GB       1.00      0.99      0.99      2033
       da-DK       0.94      0.96      0.95      2033
       de-DE       1.00      0.98      0.99      2033
       en-US       0.96      0.99      0.98      2033
       es-ES       0.98      0.98      0.98      2033
       fi-FI       1.00      0.98      0.99      2033
       fr-FR       0.99      0.99      0.99      2033
       hu-HU       1.00      0.98      0.99      2033
       is-IS       1.00      0.99      0.99      2033
       it-IT       0.98      0.99      0.99      2033
       jv-ID       0.99      0.98      0.99      2033
       lv-LV       1.00      0.99      0.99      2033
       ms-MY       0.99      0.99      0.99      2033
       nb-NO       0.96      0.94      0.95      2033
       nl-NL       0.98      0.97      0.98      2033
       pl-PL       0.99      0.98      0.98      2033
       pt-PT       0.98      0.98      0.98      2033
       ro-RO       1.00      0.99      0.99      2033
       ru-RU       1.00      0.99      1.00      2033
       sl-SL       1.00      0.99      0.99      2033
       sq-AL       1.00      0.99      0.99      2033
       sv-SE       0.97      0.98      0.97      2033
       sw-KE       1.00      0.99      1.00      2033
       tl-PH       0.99      0.99      0.99      2033
       tr-TR       1.00      0.99      0.99      2033
       vi-VN       1.00      1.00      1.00      2033

    accuracy                           0.98     54891
   macro avg       0.98      0.98      0.98     54891
weighted avg       0.98      0.98      0.98     54891

Validation Accuracy: 0.9838589204058953
```

Model's perfomance metrics for Test Partition

[11]:
```
test_predictions = pipeline.predict(df1_test['utt'])
print("Testing Data Performance:")
print(classification_report(df1_test['locale'], test_predictions))
print("Testing Accuracy:", accuracy_score(df1_test['locale'], test_predictions))
```

```
Testing Data Performance:
```

```
              precision    recall  f1-score   support

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

    accuracy                           0.98     80298
   macro avg       0.98      0.98      0.98     80298
weighted avg       0.98      0.98      0.98     80298

Testing Accuracy: 0.9833868838576303
```

Now, building a MultinomialNB model with accent removal from the data. For the "unicodedata" library is used.

[28]:
```
# ACCENT REMOVAL

df2 = df_filtered
# Function to deaccent characters
def deaccent(text):
    return ''.join(c for c in unicodedata.normalize('NFD', text) if unicodedata.
 ↪category(c) != 'Mn')

# Create a directory to store the files
```

```python
# Iterate through the dataset and save sentences to respective files
for locale in locales:
    for partition in ['train', 'validation', 'test']:
           for item in df2[partition]:
                  if item['locale'] == locale:
                          sentence = item['utt']
                          sentence = deaccent(sentence)
```

[29]:
```python
df2_train = df2['train']
df2_val = df2['validation']
df2_test = df2['test']
```

Building Multinomial Naive Bayes Model - with accents

[30]:
```python
pipeline = Pipeline([
    ('vectoriser', CountVectorizer()),
    ('classifier', MultinomialNB())
])

pipeline.fit(df2_train['utt'], df2_train['locale'])
```

[30]:
```
Pipeline(steps=[('vectoriser', CountVectorizer()),
                ('classifier', MultinomialNB())])
```

Model's perfomance metrics for Train Partition

[31]:
```python
train_predictions = pipeline.predict(df2_train['utt'])
print("Training Data Performance:")
print(classification_report(df2_train['locale'], train_predictions))
print("Training Accuracy:", accuracy_score(df2_train['locale'],␣
 ↪train_predictions))
```

```
Training Data Performance:
              precision    recall  f1-score   support

       af-ZA       0.98      0.98      0.98     11514
       cy-GB       1.00      1.00      1.00     11514
       da-DK       0.97      0.97      0.97     11514
       de-DE       1.00      0.99      0.99     11514
       en-US       0.96      0.99      0.98     11514
       es-ES       0.99      0.99      0.99     11514
       fi-FI       1.00      0.99      0.99     11514
       fr-FR       0.99      0.99      0.99     11514
       hu-HU       1.00      0.99      1.00     11514
       is-IS       1.00      1.00      1.00     11514
       it-IT       0.99      0.99      0.99     11514
       jv-ID       0.99      0.99      0.99     11514
```

```
       lv-LV       1.00      1.00      1.00     11514
       ms-MY       0.99      0.99      0.99     11514
       nb-NO       0.97      0.96      0.97     11514
       nl-NL       0.99      0.98      0.98     11514
       pl-PL       0.99      0.99      0.99     11514
       pt-PT       0.99      0.99      0.99     11514
       ro-RO       1.00      0.99      0.99     11514
       ru-RU       1.00      1.00      1.00     11514
       sl-SL       1.00      0.99      1.00     11514
       sq-AL       1.00      0.99      1.00     11514
       sv-SE       0.99      0.99      0.99     11514
       sw-KE       1.00      1.00      1.00     11514
       tl-PH       1.00      1.00      1.00     11514
       tr-TR       1.00      0.99      1.00     11514
       vi-VN       1.00      1.00      1.00     11514

    accuracy                           0.99    310878
   macro avg       0.99      0.99      0.99    310878
weighted avg       0.99      0.99      0.99    310878

Training Accuracy: 0.9909160506693945
```

Model's perfomance metrics for Validation Partition

[32]:
```
val_predictions = pipeline.predict(df2_val['utt'])
print("Validation Data Performance:")
print(classification_report(df2_val['locale'], val_predictions))
print("Validation Accuracy:", accuracy_score(df2_val['locale'],
 ↪val_predictions))
```

```
Validation Data Performance:
              precision    recall  f1-score   support

       af-ZA       0.91      0.98      0.94      2033
       cy-GB       1.00      0.99      0.99      2033
       da-DK       0.94      0.96      0.95      2033
       de-DE       1.00      0.98      0.99      2033
       en-US       0.96      0.99      0.98      2033
       es-ES       0.98      0.98      0.98      2033
       fi-FI       1.00      0.98      0.99      2033
       fr-FR       0.99      0.99      0.99      2033
       hu-HU       1.00      0.98      0.99      2033
       is-IS       1.00      0.99      0.99      2033
       it-IT       0.98      0.99      0.99      2033
       jv-ID       0.99      0.98      0.99      2033
       lv-LV       1.00      0.99      0.99      2033
       ms-MY       0.99      0.99      0.99      2033
       nb-NO       0.96      0.94      0.95      2033
```

```
      nl-NL       0.98      0.97      0.98      2033
      pl-PL       0.99      0.98      0.98      2033
      pt-PT       0.98      0.98      0.98      2033
      ro-RO       1.00      0.99      0.99      2033
      ru-RU       1.00      0.99      1.00      2033
      sl-SL       1.00      0.99      0.99      2033
      sq-AL       1.00      0.99      0.99      2033
      sv-SE       0.97      0.98      0.97      2033
      sw-KE       1.00      0.99      1.00      2033
      tl-PH       0.99      0.99      0.99      2033
      tr-TR       1.00      0.99      0.99      2033
      vi-VN       1.00      1.00      1.00      2033

   accuracy                           0.98     54891
  macro avg       0.98      0.98      0.98     54891
weighted avg       0.98      0.98      0.98     54891

Validation Accuracy: 0.9838589204058953
```

Model's perfomance metrics for Test Partition

[33]:
```python
test_predictions = pipeline.predict(df2_test['utt'])
print("Testing Data Performance:")
print(classification_report(df2_test['locale'], test_predictions))
print("Testing Accuracy:", accuracy_score(df2_test['locale'], test_predictions))
```

```
Testing Data Performance:
              precision    recall  f1-score   support

      af-ZA       0.89      0.98      0.94      2974
      cy-GB       1.00      0.99      1.00      2974
      da-DK       0.94      0.95      0.95      2974
      de-DE       0.99      0.99      0.99      2974
      en-US       0.94      0.99      0.97      2974
      es-ES       0.98      0.98      0.98      2974
      fi-FI       0.99      0.98      0.99      2974
      fr-FR       0.99      0.99      0.99      2974
      hu-HU       1.00      0.98      0.99      2974
      is-IS       1.00      0.99      0.99      2974
      it-IT       0.98      0.99      0.99      2974
      jv-ID       0.99      0.98      0.99      2974
      lv-LV       0.99      0.99      0.99      2974
      ms-MY       0.99      0.99      0.99      2974
      nb-NO       0.96      0.93      0.95      2974
      nl-NL       0.99      0.97      0.98      2974
      pl-PL       1.00      0.98      0.99      2974
      pt-PT       0.98      0.98      0.98      2974
      ro-RO       1.00      0.99      0.99      2974
```

```
       ru-RU       1.00      0.99      1.00      2974
       sl-SL       1.00      0.99      0.99      2974
       sq-AL       1.00      0.99      0.99      2974
       sv-SE       0.99      0.97      0.98      2974
       sw-KE       1.00      0.99      1.00      2974
       tl-PH       0.99      0.99      0.99      2974
       tr-TR       0.99      0.99      0.99      2974
       vi-VN       1.00      1.00      1.00      2974

    accuracy                           0.98     80298
   macro avg       0.98      0.98      0.98     80298
weighted avg       0.98      0.98      0.98     80298

Testing Accuracy: 0.9833868838576303
```

From the above results, We can see that keeping accents in the data is useful for language classification. Accuracy for model without removing accents is 99% and accuracy for model without accents is 98%. Note the fact that, the Precision for model with accents is higher than the precision for the model wihtout accents.

TASK 3 Collapsing the choosen 27 languages into a 4 label dataset, where the labels are the continent they are spoken in. Building a Regularised Discriminent Analyser (RDA) over the above modified data with Linear Discriminent Analyser (LDA) and Quadratic Discriminent Analyser (QDA) with a parameter which balances the prfedictions of both.

[18]:
```python
continent_groups = {
    'af-ZA': 'Africa', 'sw-KE': 'Africa',
    'da-DK': 'Europe', 'de-DE': 'Europe', 'es-ES': 'Europe', 'fr-FR': 'Europe',␣
 ↪'fi-FI': 'Europe',
    'hu-HU': 'Europe', 'is-IS': 'Europe', 'it-IT': 'Europe', 'lv-LV': 'Europe',␣
 ↪'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', 'cy-GB': 'Europe',
    'jv-ID': 'Asia', 'ms-MY': 'Asia', 'tl-PH': 'Asia', 'tr-TR': 'Asia', 'vi-VN':
 ↪ 'Asia',
    'en-US': 'North America'
}
```

Creating a extral column in the dataset and adding the continent value to the corresponding language and then passing the new column as the label for the RDA Model

[19]:
```python
def assign_continent(row):
    locale = row['locale']
    return continent_groups.get(locale, 'NA')
```

Using 'lambda' functions in python to apply the above function to add the continent names.

```
[20]: train_data = df_filtered['train'].map(lambda row: {'continent':␣
       ↪assign_continent(row)})
      val_data = df_filtered['validation'].map(lambda row: {'continent':␣
       ↪assign_continent(row)})
      test_data = df_filtered['test'].map(lambda row: {'continent':␣
       ↪assign_continent(row)})
```

Class for Regularised Discriminent Analysis

```
[34]: from sklearn.base import BaseEstimator, ClassifierMixin # to get two␣
       ↪classifiers inside a single class
      from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,␣
       ↪QuadraticDiscriminantAnalysis
      from sklearn.metrics import accuracy_score

      class RegularizedDiscriminantAnalysis(BaseEstimator, ClassifierMixin):
          __slots__ = '_lambda'
          def __init__(self, _lambda=0.5):
              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):
              lda_pred = self.lda.predict_proba(X)
              qda_pred = self.qda.predict_proba(X)
              combined_pred = ((1 - self._lambda) * lda_pred) + (self._lambda *␣
       ↪qda_pred)
              return np.argmax(combined_pred, axis=1)
```

Using min frequency prunning (min_df = 10) and limiting the features to 250, the model is build and tunned for hyper parameter lambda

```
[55]: X_train_vec = CountVectorizer(max_features=400, min_df=10).
       ↪fit_transform(train_data['utt'])
      y_train = LabelEncoder().fit_transform(train_data['continent'])

      X_test_vec = CountVectorizer(max_features=400, min_df=10).
       ↪fit_transform(test_data['utt'])
      y_test = LabelEncoder().fit_transform(test_data['continent'])

      X_val_vec = CountVectorizer(max_features=400, min_df=10).
       ↪fit_transform(val_data['utt'])
```

```
y_val = LabelEncoder().fit_transform(val_data['continent'])
```

[56]:
```
# Initialize and train the RDA model
rda = RegularizedDiscriminantAnalysis(_lambda=0)
rda.fit(X_train_vec.toarray(), y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/sklearn/discriminant_analysis.py:926: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
```

[56]:
```
RegularizedDiscriminantAnalysis(_lambda=0)
```

[57]:
```
y_test = LabelEncoder().fit_transform(y_test)
# Predict and evaluate the model
y_pred = rda.predict(X_test_vec.toarray())
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.6566913248150639
```

[40]:
```
# Define the parameter grid
parameter_grid =  [0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1]
best_accuracy = float('-inf')
best_parameter = 0

for i in range(0, len(parameter_grid)):
    model = RegularizedDiscriminantAnalysis(_lambda = parameter_grid[i]).
 ↪fit(X_train_vec.toarray(), y_train)
    y_pred = model.predict(X_val_vec.toarray())
    accuracy = accuracy_score(y_val, y_pred)
    if(accuracy > best_accuracy):
        best_parameter = parameter_grid[i]
        best_accuracy = accuracy
    print(f'Iteration{i} Done')
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/sklearn/discriminant_analysis.py:926: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")

Iteration0 Done

/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/sklearn/discriminant_analysis.py:926: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")

Iteration1 Done
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/sklearn/discriminant_analysis.py:926: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
```

```
Iteration2 Done
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/sklearn/discriminant_analysis.py:926: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
```

```
Iteration3 Done
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/sklearn/discriminant_analysis.py:926: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
```

```
Iteration4 Done
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/sklearn/discriminant_analysis.py:926: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
```

```
Iteration5 Done
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/sklearn/discriminant_analysis.py:926: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
```

```
Iteration6 Done
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/sklearn/discriminant_analysis.py:926: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
```

```
Iteration7 Done
```

[41]:
```
best_accuracy
```

[41]: 0.733599315006103

[42]:
```
best_parameter
```

[42]: 0

After hyperparameter tuning, the best parameter is lambda = 0. With the best parameter, the accuracy is 73.5% with max_df = 10 and max_features = 400 in counter vectoriser.