LTU_PhD_CyberSecAI_ProgTask_SMondal_v0.003

August 10, 2024

1 Programming Task:

1.1 ## Language Recognition Using Distributed High Dimensional Representations

Author Details -

Name: **Sharadananda Mondal**
Email: sharadananda.mondal@gmail.com
Mobile: +91-8709331235

1.1.1 Objective:

Classify the language of an input text. The language recognition will be done for 21 European languages. The list of languages being - Bulgarian, Czech, Danish, German, Greek, English, Estonian, Finnish, French, Hungarian, Italian, Latvian, Lithuanian, Dutch, Polish, Portuguese, Romanian, Slovak, Slovene, Spanish, Swedish. The training and testing data is based on the Wortschatz Corpora: https://wortschatz.uni-leipzig.de/en/download

1.1.2 Approach:

- Smaller size subsets of original data sets were used for faster processing. In fact selected randomly 500 rows from each language corpora and combined together to form train data set for the classification data model.
- Data preprocessing was necessary. The data procured from Leipzig corpora had some noise in it. Used Notepad++ for data cleaning. Removed symbols, punctuation marks, digits from the data. Later checked for punctuation marks using Jupyter notebook
- A 70-30 split was done for model training and testing purpose.
- Using tri-grams were memory intensive especially when used with Count Vectorizier, a sklearn library
- Ridge Classification was used to train the model, an implementation from sklearn library
- Test data were taken from euparl repository
- Confusion matrix, accuracy and F1-scores were calculated and plotted

1.1.3 Model Building

Step-1: Import the necessary Libraries

[1]:
```
import re
import string
```

```
import itertools
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import RidgeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
from sklearn import metrics
```

Step-2: Load data

Data was preprocessed. It was obtained by combining news data in selected languages from the Leipzig Corpora repository.

[2]:
```
data = pd.read_csv("../input/leipzig21_500_sub.csv")
data.head(5)
```

[2]:

	Text	Language
0	…	Bulgarian
1	…	Bulgarian
2	…	Bulgarian
3	…	Bulgarian
4	…	Bulgarian

Outline of data. It shows no of rows and memory usage.

[3]:
```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10500 entries, 0 to 10499
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Text      10500 non-null  object
 1   Language  10500 non-null  object
dtypes: object(2)
memory usage: 164.2+ KB
None
```

Rows by language type.

[4]:
```
data['Language'].value_counts()
```

[4]:
```
Language
Bulgarian     500
Greek         500
```

```
Swedish       500
Spanish       500
Slovenian     500
Slovak        500
Romanian      500
Portuguese    500
Polish        500
English       500
Dutch         500
Czech         500
Lithuanian    500
Lativian      500
Italian       500
French        500
Finnish       500
Estonian      500
German        500
Danish        500
Hungarian     500
Name: count, dtype: int64
```

Step-3: Separating Label from the data

Labels are the predictor variables, dependent variable. While the other features are independent variables.

[5]:
```
X = data["Text"]
y = data["Language"]
```

Label Encoding

[6]:
```
le = LabelEncoder()
y = le.fit_transform(y)
```

Step-4: Text Preprocessing

[7]:
```
# creating a list for appending the preprocessed text
data_list = []
# iterating through all the text
for text in X:
        # removing the symbols and numbers
         text = re.sub(r'[!@#$(),n"%^*?:;~`0-9®©$]', ' ', text)
         text = re.sub(r'[[]]', ' ', text)
         # converting the text to lower case
         text = text.lower()
         # removing punctuation
         text = ''.join([j for j in text if j not in string.punctuation])
         # appending to data_list
         data_list.append(text)
```

```
C:\Users\shara\AppData\Local\Temp\ipykernel_26108\1572569825.py:7:
FutureWarning: Possible nested set at position 1
  text = re.sub(r'[[]]', ' ', text)
```

Bag of Words

Output feature and input feature should be of the numerical form. So we are converting text into numerical form by creating a Bag of Words model using CountVectorizer.

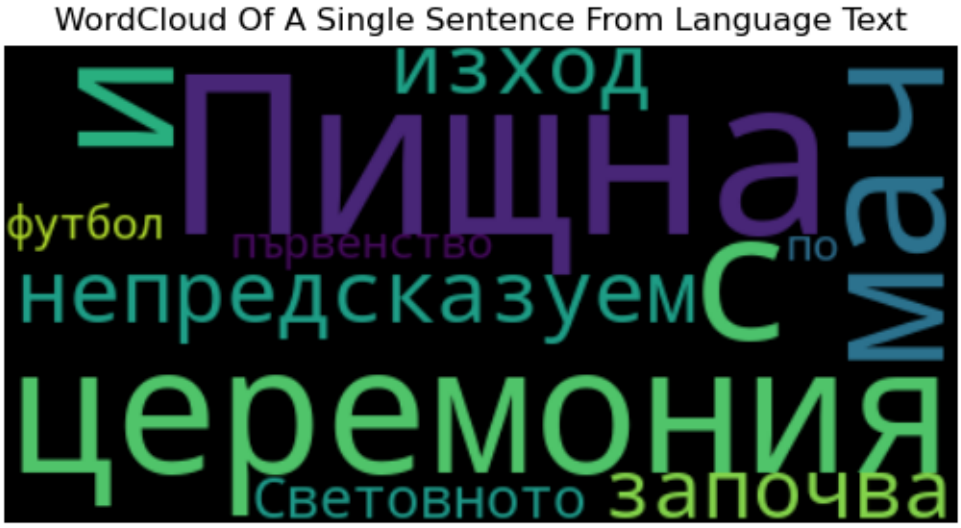
[8]:
```
cv = CountVectorizer(ngram_range=(1,1), analyzer="word", min_df=1, max_df=1.0)␣
↪# unigram (1,1) for word analyzer
X = cv.fit_transform(data_list).toarray()
X.shape #(10500, 157042)
```

[8]:
```
(10500, 64952)
```

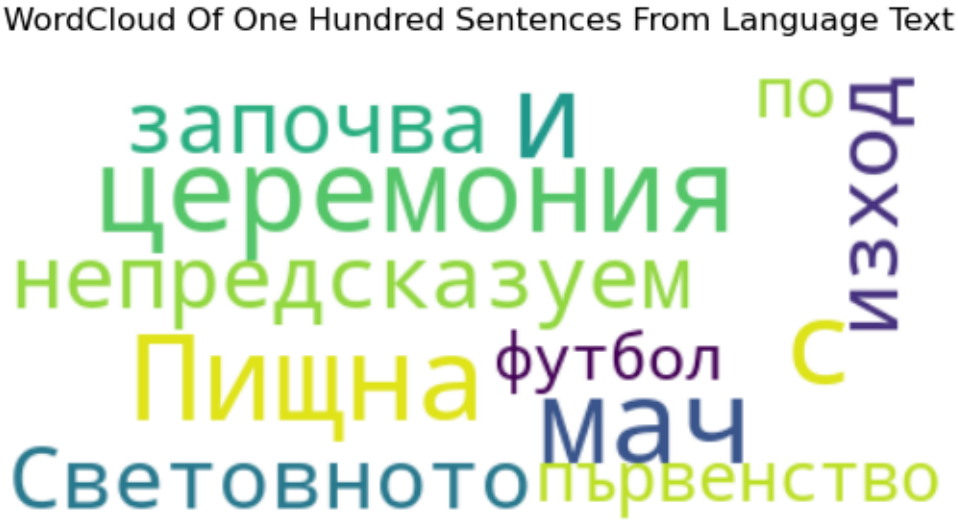
1.1.4 Visualization: WordCloud

[9]:
```
from wordcloud import WordCloud
```

[10]:
```
# Start with one review:
text = data.Text[0]

# Create and generate a word cloud image:
wordcloud = WordCloud().generate(text)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("WordCloud Of A Single Sentence From Language Text")
plt.show()
```



[11]:
```
# lower max_font_size, change the maximum number of word and lighten the␣
 ↪background:
wordcloud = WordCloud(max_font_size=50, max_words=100,␣
 ↪background_color="white", random_state=100).generate(text)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("WordCloud Of One Hundred Sentences From Language Text")
plt.show()
```



[12]:
```
text = " ".join(review for review in data.Text)
print ("There are {} words in the combination of all review.".format(len(text)))
```

```
There are 1125189 words in the combination of all review.
```

[13]:
```
from wordcloud import STOPWORDS
```

[14]:
```
# Create stopword list:
stopwords = set(STOPWORDS)
stopwords.update(["la", "de", "v", "na", "le", "je", "o", "en", "z", "y", "al",␣
 ↪"ha", "un", "si", "ja", "lo", "az", "po","nu",
                  "c", "e", "w", "il", "ir", "da", "se", "te", "el", "em",␣
 ↪"di", "um", "et"]),

# Create and generate a word cloud image:
wordcloud = WordCloud(stopwords=stopwords, width= 800, height = 500,␣
 ↪random_state= 42,
```

```
                      max_font_size= 100, background_color="white",␣
↪max_words=5000).generate(text)

# Display the generated image:
# the matplotlib way:
plt.figure(figsize = (15, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title("WordCloud of Language Text from 21 European Languages")
plt.axis("off")
#plt.axes(linewidth=1.0)
plt.show()
```

[15]:
```
# Save the image in the img folder:
wordcloud.to_file("../img/five_thousand_sentences.png")
```

[15]: <wordcloud.wordcloud.WordCloud at 0x254bd5500b0>

Step-5: Train Test Data Splitting

[16]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.30,␣
↪random_state=0)
```

Step-6: Model Training and Prediction

[17]:
```
model = RidgeClassifier()
model.fit(x_train, y_train)
```

[17]:
```
RidgeClassifier()
```

Prediction

[18]:
```
y_pred = model.predict(x_test)
```

Step-7: Model Evaluation

Model Accuracy

[19]:
```
lang_labels = le.inverse_transform(np.unique(y_test))
```

[20]:
```
ac = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
clf_report = classification_report(y_test, y_pred, target_names=lang_labels)
print("Accuracy is :",ac)
```

```
Accuracy is : 0.9482539682539682
```

[21]:
```
print(clf_report)
```

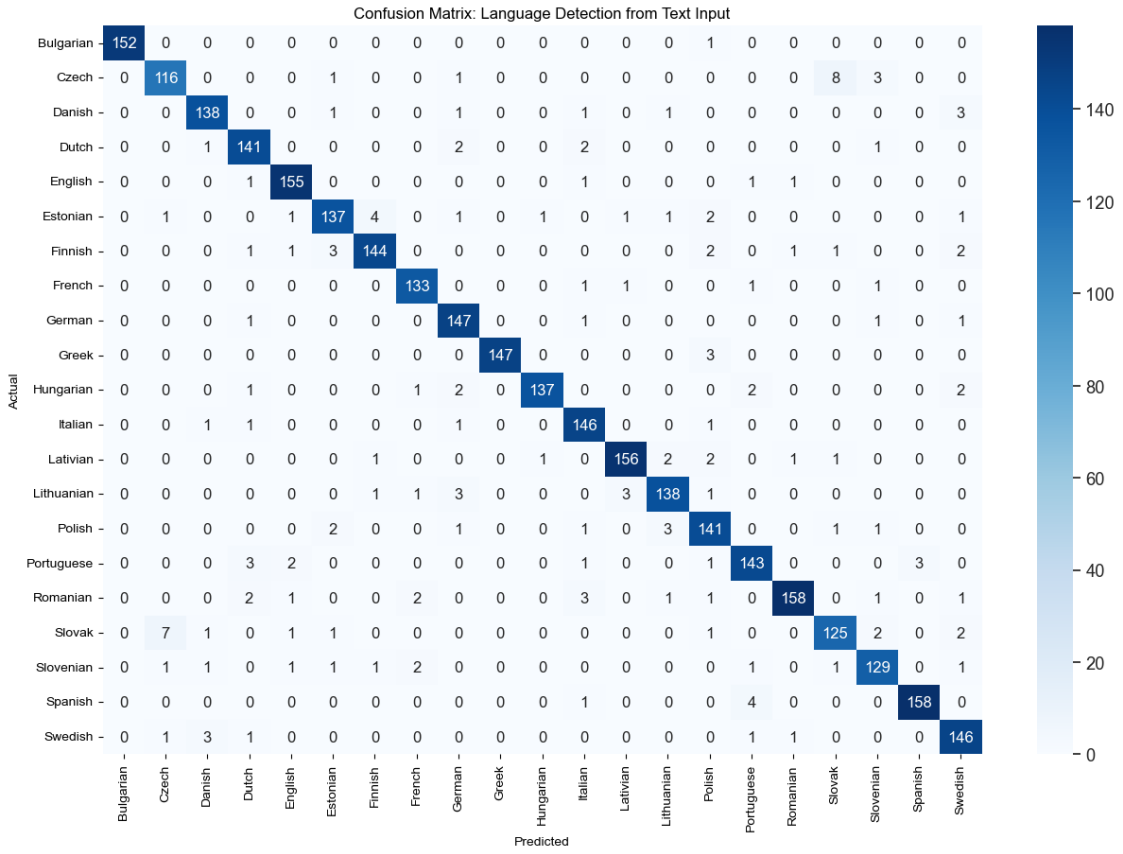
```
              precision    recall  f1-score   support

   Bulgarian       1.00      0.99      1.00       153
       Czech       0.92      0.90      0.91       129
      Danish       0.95      0.95      0.95       145
       Dutch       0.93      0.96      0.94       147
     English       0.96      0.97      0.97       159
    Estonian       0.94      0.91      0.93       150
     Finnish       0.95      0.93      0.94       155
      French       0.96      0.97      0.96       137
      German       0.92      0.97      0.95       151
       Greek       1.00      0.98      0.99       150
   Hungarian       0.99      0.94      0.96       145
     Italian       0.92      0.97      0.95       150
    Lativian       0.97      0.95      0.96       164
  Lithuanian       0.95      0.94      0.94       147
      Polish       0.90      0.94      0.92       150
  Portuguese       0.93      0.93      0.93       153
    Romanian       0.98      0.93      0.95       170
       Slovak       0.91      0.89      0.90       140
   Slovenian       0.93      0.93      0.93       139
     Spanish       0.98      0.97      0.98       163
     Swedish       0.92      0.95      0.94       153

    accuracy                           0.95      3150
   macro avg       0.95      0.95      0.95      3150
```

```
weighted avg       0.95      0.95      0.95      3150
```

Confusion Matrix

[22]:
```
#df_cm = pd.DataFrame(cm, columns=np.unique(y_test), index = np.unique(y_test))
df_cm = pd.DataFrame(cm, columns=lang_labels, index = lang_labels)
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (15,10))
plt.title("Confusion Matrix: Language Detection from Text Input")
sns.set(font_scale=1.2)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 12}, fmt="d")#font size
```

[22]:
```
<Axes: title={'center': 'Confusion Matrix: Language Detection from Text Input'},
xlabel='Predicted', ylabel='Actual'>
```



Step-8: Model Validation

[23]:
```
def predict(text):
    x = cv.transform([text]).toarray() # converting text to bag of words model␣
 ↪(Vector)
    lang = model.predict(x) # predicting the language
    lang = le.inverse_transform(lang) # finding the language corresponding the␣
 ↪the predicted value
    print("The langauge is in",lang[0]) # printing the language
```

[24]:
```
predict("                                                                    ␣
 ↪ ")
```

```
The langauge is in Bulgarian
```

[25]:
```
predict("komandų žaidėjai rungtynes pradėjo be didesnės žvalgybos")
```

```
The langauge is in Lithuanian
```

1.2 Data Pre-Processing Step: Read & Combine data to form a single csv file for building the classification model

[26]:
```
data_files = ['bul_news_2022_30K_sentences',␣
 ↪'ces_news_2022_30K_sentences','dan_news_2022_30K_sentences',
              'deu_news_2023_30K_sentences',␣
 ↪'est_news_2022_30K_sentences','fin_news_2022_30K_sentences',
             ␣
 ↪'fra_news_2023_30K_sentences','ita_news_2023_30K_sentences','lav_news_2020_30K_sentences',
             ␣
 ↪'lit_news_2020_30K_sentences','nld_news_2022_30K_sentences','ell_news_2022_30K_sentences',
             ␣
 ↪'eng_news_2023_30K_sentences','pol_news_2023_30K_sentences','por_news_2022_30K_sentences',
             ␣
 ↪'ron_news_2022_30K_sentences','slk_news_2020_30K_sentences','slv_news_2020_30K_sentences',
             ␣
 ↪'spa_news_2023_30K_sentences','swe_news_2022_30K_sentences','hun_news_2023_30K_sentences']
languages = ['bul', 'ces','dan','deu','est','fin','fra','ita','lav','lit',␣
 ↪'nld', 'ell','eng','pol','por','ron','slk','slv','spa','swe','hun']
df_dic = {}
```

[27]:
```
for filename, language in zip(data_files, languages):
    df_name = language + '_df'
    file = "../input/data/{}.csv".format(filename)
    df_dic[df_name] = pd.read_csv(file)
```

[28]:
```
df = pd.concat(df_dic.values(), ignore_index=True)
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 630000 entries, 0 to 629999
```

```
Data columns (total 2 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   Text      630000 non-null  object
 1   Language  630000 non-null  object
dtypes: object(2)
memory usage: 9.6+ MB
None
```

[29]:
```
df['Language'].value_counts()
```

[29]:
```
Language
Bulgarian     30000
Greek         30000
Swedish       30000
Spanish       30000
Slovenian     30000
Slovak        30000
Romanian      30000
Portuguese    30000
Polish        30000
English       30000
Dutch         30000
Czech         30000
Lithuanian    30000
Lativian      30000
Italian       30000
French        30000
Finnish       30000
Estonian      30000
German        30000
Danish        30000
Hungarian     30000
Name: count, dtype: int64
```

1.3 Creating a subset of data by selecting a few rows randomly

[30]:
```
dfsub_dic = {}
for name, dat in df_dic.items():
    dfsub_name = name + '_subdf'
    dfsub_dic[dfsub_name] = dat.sample(n=500, random_state=3).
 ↪reset_index(drop=True)
```

[31]:
```
subdf = pd.concat(dfsub_dic.values(), ignore_index=True)
print(subdf.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10500 entries, 0 to 10499
```

```
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Text      10500 non-null  object
 1   Language  10500 non-null  object
dtypes: object(2)
memory usage: 164.2+ KB
None
```

[32]:
```
subdf['Language'].value_counts()
```

[32]:
```
Language
Bulgarian     500
Greek         500
Swedish       500
Spanish       500
Slovenian     500
Slovak        500
Romanian      500
Portuguese    500
Polish        500
English       500
Dutch         500
Czech         500
Lithuanian    500
Lativian      500
Italian       500
French        500
Finnish       500
Estonian      500
German        500
Danish        500
Hungarian     500
Name: count, dtype: int64
```

[33]:
```
subdf.to_csv("../input/leipzig21_500_sub.csv", index=False)
```

[34]:
```
df_ = pd.read_csv("../input/leipzig21_500_sub.csv")
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 630000 entries, 0 to 629999
Data columns (total 2 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   Text      630000 non-null  object
 1   Language  630000 non-null  object
dtypes: object(2)
```

```
memory usage: 9.6+ MB
None
```

[35]:
```
df_.head()
```

[35]:

	Text	Language
0	…	Bulgarian
1	…	Bulgarian
2	…	Bulgarian
3	…	Bulgarian
4	…	Bulgarian

[ ]:
```

```