AGR9012 - Image and Text Processing for Data Science

Sebnem Keles
27435249

UNIVERSITY OF
LINCOLN

Task 2: Text Processing

Introduction:

First of all, I surveyed the contents of the folders I'll be working on by examining the quantity of texts in each file and subsequently inspected the contents by 'print' each text file with a for loop. The number of text files are 510, 386, 417, 511 and 401 for business, entertainment, politics, sport and tech respectively.

```
# Reading files from a particular folder and indicating how many files there are

files = []
for dirname, _, filenames in os.walk("C:/Users/sbnm9/Downloads/Image Process Assessment 1/bbc"):
    for filename in tqdm(filenames):
        files.append(os.path.join(dirname, filename))
print('There are a total of {} files'.format(len(files)), '\n')
```

```
0it [00:00, ?it/s]
100%|| 510/510 [00:00<00:00, 63546.28it/s]
100%|| 386/386 [00:00<?, ?it/s]
100%|| 417/417 [00:00<?, ?it/s]
100%|| 511/511 [00:00<00:00, 68045.25it/s]
100%|| 401/401 [00:00<00:00, 40013.23it/s]
```

```
There are a total of 2225 files
```

```
# Viewing the text detail files [1]

documents = []
topics = []

directory = os.path.normpath("C:/Users/sbnm9/Downloads/Image Process Assessment 1/bbc")

for subdir, dirs, files in os.walk(directory):
    for file in files:
        if file.endswith(".txt"):
            topics.append(subdir.split('\\')[-1])
            f = open(os.path.join(subdir, file),'r')
            a = f.read()
            documents.append(a)
#              print (documents)
            f.close()
```

Preprocessing:

To work properly with my dataset, I converted the text list into NumPy array using pandas library and labelled my documents and topics.

```
# Converting list into np.array

import pandas as pd
documents = np.array(documents)
pd.DataFrame({'Document':documents, 'Category':topics})
```

```
C:\Users\sbnm9\anaconda3\lib\site-packages\pandas\core\computation\expressions.py:21: UserWarning: Pandas
8.4' or newer of 'numexpr' (version '2.8.3' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
C:\Users\sbnm9\anaconda3\lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires v
er of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

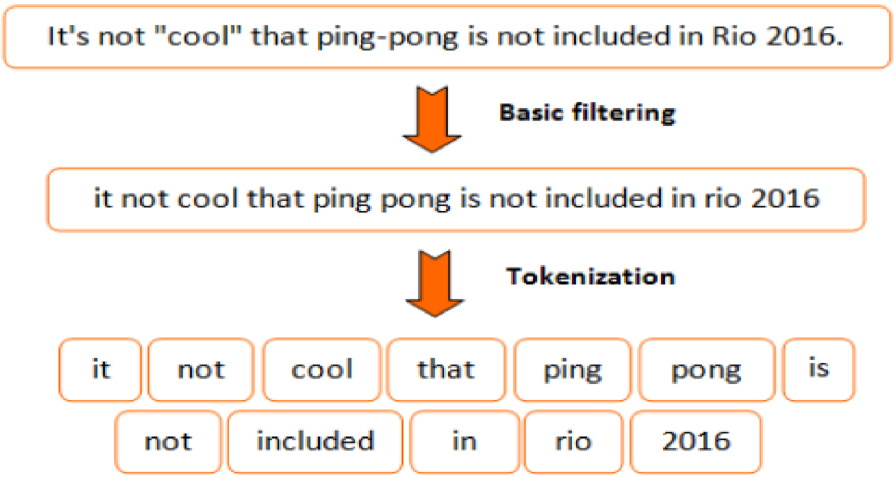
	Document	Category
0		business
1	Dollar gains on Greenspan speech\n\nThe dollar...	business
2	Yukos unit buyer faces loan claim\n\nThe owner...	business
3	High fuel prices hit BA's profits\n\nBritish A...	business
4	Pernod takeover talk lifts Domecq\n\nShares in...	business
...	...	...
2220	BT program to beat dialler scams\n\nBT is intr...	tech
2221	Spam e-mails tempt net shoppers\n\nComputer us...	tech
2222	Be careful how you code\n\nA new European dire...	tech
2223	US cyber security chief resigns\n\nThe man mak...	tech
2224	Losing yourself in online gaming\n\nOnline rol...	tech

2225 rows × 2 columns

Before starting the modelling, I imported necessary libraries and proceeded to the preprocessing stage to refine my data for more accurate predictions. In this phase, I utilized arguments to clean non-alphabetic characters, convert text to lowercase, strip punctuation, word tokenizing, and lemmatize the tokens. These steps were chosen to streamline the process and enhance the quality of the analysis.

```python
# Preprocessing steps

import re
import string
from nltk.stem import WordNetLemmatizer

wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')
wnl = WordNetLemmatizer()

def process_docs(doc):
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I | re.A)
    doc = doc.lower()
    doc = doc.strip()
    tokens = wpt.tokenize(doc)
    filtered_tokens = [token for token in tokens if token not in stop_words]
    lemmatized_tokens = [wnl.lemmatize(token) for token in filtered_tokens]
    doc = ' '.join(lemmatized_tokens)
    return doc
```

Before moving forward, I would like to explain a bit about tokenizing and lemmatization. Tokenization involves breaking down a sentence into individual words, while lemmatization entails reducing words to their base or root form. Examples of both are provided below."



[Text tokenization & multiwords | MeaningCloud]()

[1.5 Stemming, Lemmatization, Stopwords, POS Tagging — Practical NLP with Python (nlplanet.org)]

As a next step, I normalized the corpus to enhance the stability of machine learning.

```
# Normalizing the corpus

corpus_tokens = [process_docs(text).split(' ') for text in documents]
normalize_corpus = np.vectorize(process_docs)
corpus_norm = normalize_corpus(documents)
pd.DataFrame({'Original':documents, 'Normalized':corpus_norm})
```

	Original	Normalized
0		
1	Dollar gains on Greenspan speech\n\nThe dollar...	dollar gain greenspan speech dollar hit highes...
2	Yukos unit buyer faces loan claim\n\nThe owner...	yukos unit buyer face loan claim owner embattl...
3	High fuel prices hit BA's profits\n\nBritish A...	high fuel price hit ba profit british airway b...
4	Pernod takeover talk lifts Domecq\n\nShares in...	pernod takeover talk lift domecq share uk drin...
...	...	...
2220	BT program to beat dialler scams\n\nBT is intr...	bt program beat dialler scam bt introducing tw...
2221	Spam e-mails tempt net shoppers\n\nComputer us...	spam email tempt net shopper computer user acr...
2222	Be careful how you code\n\nA new European dire...	careful code new european directive could put ...
2223	US cyber security chief resigns\n\nThe man mak...	u cyber security chief resigns man making sure...
2224	Losing yourself in online gaming\n\nOnline rol...	losing online gaming online role playing game ...

2225 rows × 2 columns

Modelling & Evaluation:

Once my dataset was prepared for Machine Learning (ML) modelling, I initiated the process by employing Term Frequency-Inverse Document Frequency (TF-IDF) vectorization for feature extraction by TfidfVectorizer command from skit-learn library. Subsequently, I split the data into training and test sets to prediction and evaluation which are the next stages.

```
# TF-IDF vectorization

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(min_df = 0., max_df = 1., use_idf = True, smooth_idf =True)
tv_matrix = tfidf.fit_transform(corpus_norm)
tv_tfidf = pd.DataFrame(tv_matrix.toarray(), columns = tfidf.get_feature_names_out())
```

```
# Splitting data into %70 train and %30 test set [2]

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(tv_tfidf, topics, test_size=0.3, random_state=42)
```

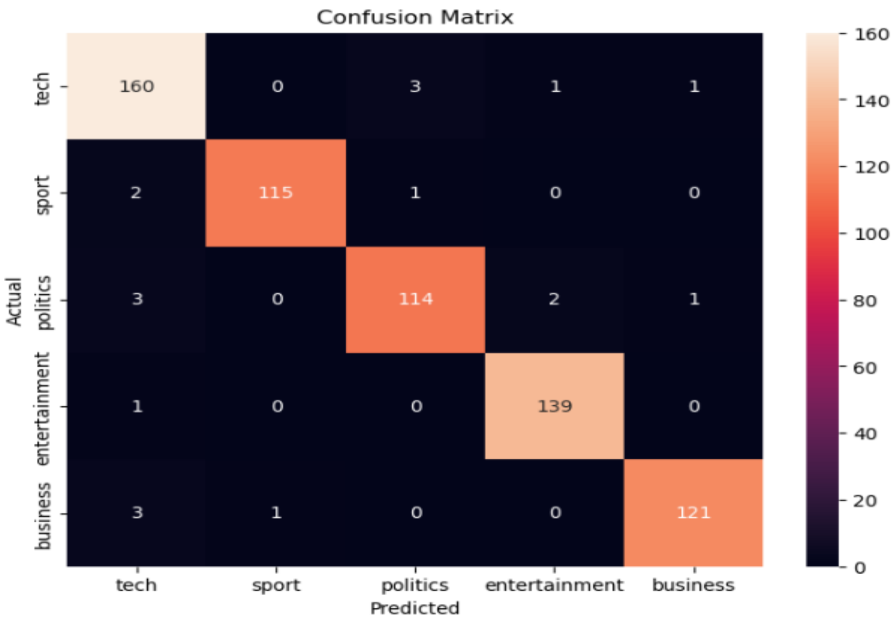
After splitting the dataset, I evaluated the Logistic Regression (LR) model, complemented by TF-IDF vectorization, to assess its accuracy on the testing set and its predictive performance on the training set. I employed GridSearch for hyperparameter tuning to optimize the model and achieve enhanced accuracy. I can display the precision and recall scores for each topic, alongside an impressive test set accuracy of 97%. The 'entertainment' topic boasts the highest precision score at 99%, while 'sport' leads in recall score at the same percentage.

```
# Logistic Regression (LR) with tf-idf vectorization and GridSearch hyperparameter tuning [3]

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

logreg = LogisticRegression()

grid_search = GridSearchCV(logreg, param_grid = {}, cv=10)
grid_search.fit(x_train, y_train)

best_logreg = grid_search.best_estimator_
y_pred = best_logreg.predict(x_test)

print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
                precision    recall  f1-score   support

     business       0.95      0.97      0.96       165
entertainment       0.99      0.97      0.98       118
     politics       0.97      0.95      0.96       120
        sport       0.98      0.99      0.99       140
         tech       0.98      0.97      0.98       125

     accuracy                           0.97       668
    macro avg       0.97      0.97      0.97       668
 weighted avg       0.97      0.97      0.97       668
```

To measure the evaluation performance, I used Confusion Matrix. As (Krstinić et al., 2020) mentioned 'In its simplest form a confusion matrix shows a binary classifier performance in table with two rows and two columns and represents the percentages of four possible classification outcomes: True Positive (TP), False Positive (FP), True Negative (TN) and False negative (FN).' For instance, number of TP for 'sport' is 115.

```
#Confusion matrix for LR-tfidf [4]

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['tech', 'sport', 'politics', 'entertainment', 'business'],
            yticklabels=['tech', 'sport', 'politics', 'entertainment', 'business'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



I experimented with a second model, utilizing Logistic Regression (LR) coupled with Bag-of-Words (BoW) vectorization using the CountVectorizer command. Once more, I divided the dataset into training and testing sets, allocating 70% for training and 30% for testing.

```
# BoW vectorization

import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df = 0., max_df = 1.)
cv_matrix = cv.fit_transform(corpus_norm)
cv_bow = pd.DataFrame(cv_matrix.toarray(), columns = cv.get_feature_names_out())
```

```
# Splitting data into %70 train and %30 test set

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(cv_bow, topics, test_size=0.3, random_state=42)
```

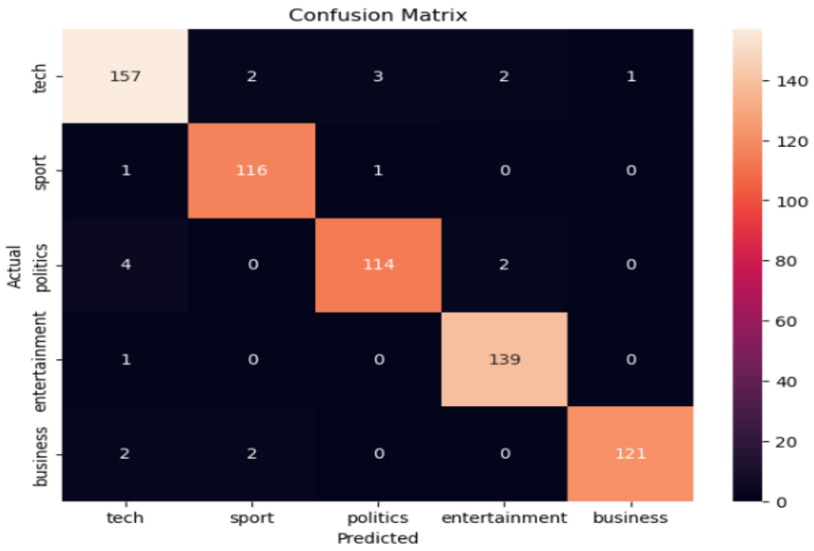
Despite achieving identical accuracy and covering the same topic as the LR model complemented by TF-IDF, it differs with the precision score of 99% is observed in the 'tech' category.

```
# Logistic Regression (LR) with BoW vectorization and GridSearch hyperparameter tuning [3]

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

logreg = LogisticRegression()

grid_search = GridSearchCV(logreg, param_grid = {}, cv=10)
grid_search.fit(x_train, y_train)

best_logreg = grid_search.best_estimator_
y_pred = best_logreg.predict(x_test)

print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

     business       0.95      0.95      0.95       165
entertainment       0.97      0.98      0.97       118
     politics       0.97      0.95      0.96       120
        sport       0.97      0.99      0.98       140
         tech       0.99      0.97      0.98       125

     accuracy                           0.97       668
    macro avg       0.97      0.97      0.97       668
 weighted avg       0.97      0.97      0.97       668
```

Finally, I presented the Confusion Matrix for Logistic Regression (LR) coupled with Bag-of-Words. Notably, the counts of true positives ('tech' and 'sport') differ from the previous matrix, with 157 for 'tech' and 116 for 'sport'. Upon closer examination of the Matrix, it becomes evident that the 'tech' category boasts the highest rating, totalling at 157 out of 165. The most notable misprediction occurred when an item actually categorized as ‘politics’ was mistakenly predicted as ‘tech’, scoring a 4 on the matrix.

```python
# Confusion matrix for LR-BoW

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['tech', 'sport', 'politics', 'entertainment', 'business'],
            yticklabels=['tech', 'sport', 'politics', 'entertainment', 'business'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Conclusion:

In summary, both Logistic Regression models utilizing TF-IDF and Bag-of-Words achieved an impressive accuracy rate of 97%, indicating their suitability for future predictions.

References:

Krstinić, Damir, et al. "Multi-Label Classifier Performance Evaluation with Confusion Matrix." *Computer Science & Information Technology*, 27 June 2020, csitcp.com/paper/10/108csit01.pdf, https://doi.org/10.5121/csit.2020.100801.