CS 621 - Deep Learning for NLP
Assignment 1
Supervised by Dr. Fawaz Al-Salmi

Assignment Report

Github repository of project: https://github.com/madnanrizqu/cs681-assignment-1

Results

We found each dataset has its own best performer combination of machine learning model + vectorization method. The IMDB has Logistic Regression + TF-IDF perform the best with 0.9007 accuracy. On the other hand, 20newsgroup has Naive Bayes + BoW perform the best with 0.9957 accuracy.

Another thing is that each dataset has its own preferred machine learning model. On 20newsgroup, Naive Bayes consistently outperforms Logistic Regression regardless of vectorization method, while IMDB demonstrates the opposite pattern, with Logistic Regression typically achieving better results.

Finally, Word2Vec performed the worst. One possible explanation is that we do not use pre-trained embeddings, instead we create custom embeddings for each dataset. Word2vec probably needs more data to capture semantic relationships better.

IMDB Dataset

Name	Accuracy
Logistic Regression + BoW	0.8848
Logistic Regression + Tf-IDF	0.9007
Logistic Regression + Word2vec	0.8762
Naive Bayes + BoW	0.8578
Naive Bayes + Tf-IDF	0.8648
Naive Bayes + Word2vec	0.7597

20newsgroup

Name	Accuracy
Logistic Regression + BoW	0.9829

Logistic Regression + Tf-IDF	0.9744
Logistic Regression + Word2vec	0.9145
Naive Bayes + BoW	0.9957
Naive Bayes + Tf-IDF	0.9915
Naive Bayes + Word2vec	0.8120

Methodology

In this assignment, we use:

1. Python 3.11
2. Conda: managing python environment
3. Jupyter notebook
4. Well known Python libraries. Most notably: pandas, numpy, matplotlib, sklearn, nltk and gensim. The details are capture in this lockfile
5. Github

Then we have 2 datasets:

1. IMDB: we will do sentiment analysis
2. 20newsgroup: only the 'rec.autos', 'comp.graphics' categories. Hence will do text classification to categorize news into the correct category

The datasets goes through the same methodology to get the results, which are:

1. Dataset download
2. Dataset exploration
3. Preprocessing
4. Dataset split
5. Vectorization
6. Model training
7. Testing

Dataset download

In this step goal is to make the dataset ready to be used in the Jupyter notebook runtime. the dataset gets downloaded into the runtime of the Jupyter notebook for further processing. It was particularly more difficult to get the IMDB data ready since it was originally a tar file. After the dataset is inside the runtime then further exploration can be performed.

The most difficult part was to make the Jupyter notebook download deterministic for the IMDB dataset. Here is what was done:

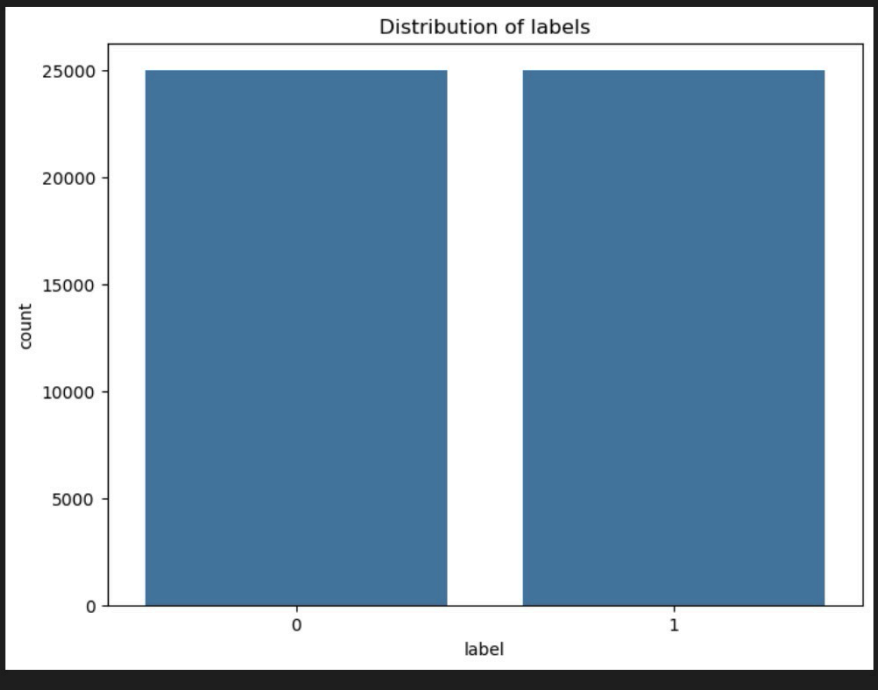
1. Put dataset tar file into a fast blob storage, we used Github
2. Download the file programmatically in Jupyter notebook
3. Extract the tar file
4. Extract data and labels by crawling all files inside train>pos, train>neg, test>pos and test>neg
5. Change labels from pos->1 and neg->0
6. Saves the extracted data and labels to a pandas dataframe
7. Save the dataframe into .csv file to bypass dataset download on future notebook runs

Dataset exploration

In this step the goal is to know how best to preprocess the dataset. We also want to know whether there are any empty values and whether the dataset has balanced class distribution.

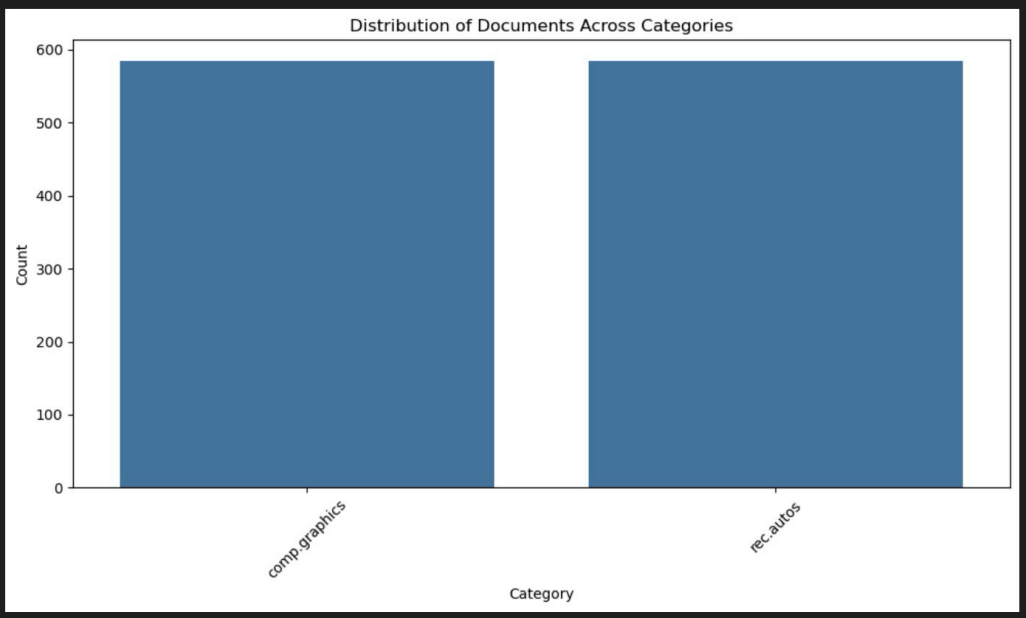
Here are the results for IMDB. One highlight find from this dataset is that some rows has HTML tags which most likely don't benefit towards sentiment analysis:

```
Dataset shape: (50000, 2)

First 5 rows:
                                                text  label
0  For a movie that gets no respect there sure ar...      1
1  Bizarre horror movie filled with famous faces ...      1
2  A solid, if unremarkable film. Matthau, as Ein...      1
3  It's a strange feeling to sit alone in a theat...      1
4  You probably all already know this by now, but...      1

Columns and data types:
text     object
label     int64
dtype: object

Missing values per column:
text     0
label    0
dtype: int64

Label distribution:
label
1    25000
0    25000
Name: count, dtype: int64
```



Here are the results for 20newsgroups. The highlight is that the data is semi structured in which it has attributes like “From”, “Subject”, “Distribution”, etc. Ideally any preprocessing should make use of these attributes and extract key values from them. Furthermore, each attribute might have an unique way to best extract the value.

```
Dataset shape: (1168, 3)

Columns and data types:
data          object
label          int64
label_text    object
dtype: object

Missing values per column:
data          0
label         0
label_text    0
dtype: int64
```



Preprocessing

In this step the goal is to remove unwanted characters from the dataset, the ones that might negatively impact or at least have no value in the machine learning task.

Here are the preprocessing steps done for IMDB:

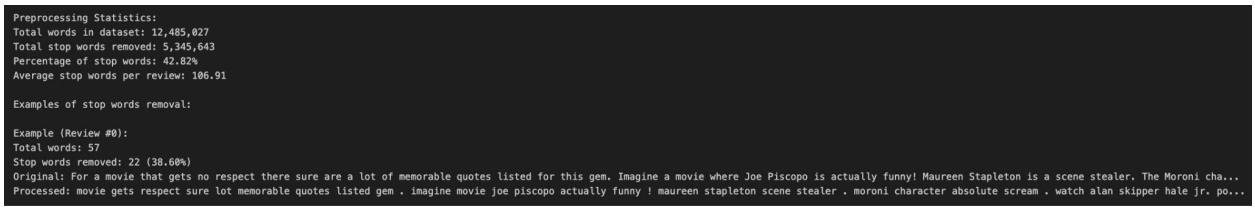
1. Remove HTML tag

```
Preprocessing Statistics:
Total characters in dataset: 65,471,551
Total HTML chars filtered: 1,212,813
Percentage of HTML chars: 1.85%
Average HTML chars per review: 24.26

Examples where HTML was found:

Example (Review #3):
Total chars: 2596
HTML chars removed: 96 (3.70%)
Original: It's a strange feeling to sit alone in a theater occupied by parents and their rollicking kids. I felt like instead of a movie ticket, I should have been given a NAMBLA membership.<br /><br />Based up...
Processed: It's a strange feeling to sit alone in a theater occupied by parents and their rollicking kids. I felt like instead of a movie ticket, I should have been given a NAMBLA membership.Based upon Thomas Ro...
```

2. Remove special char which is any char besides alphabets, numbers, spaces and basic punctuation (“.”, “,”, “!”, “?”)

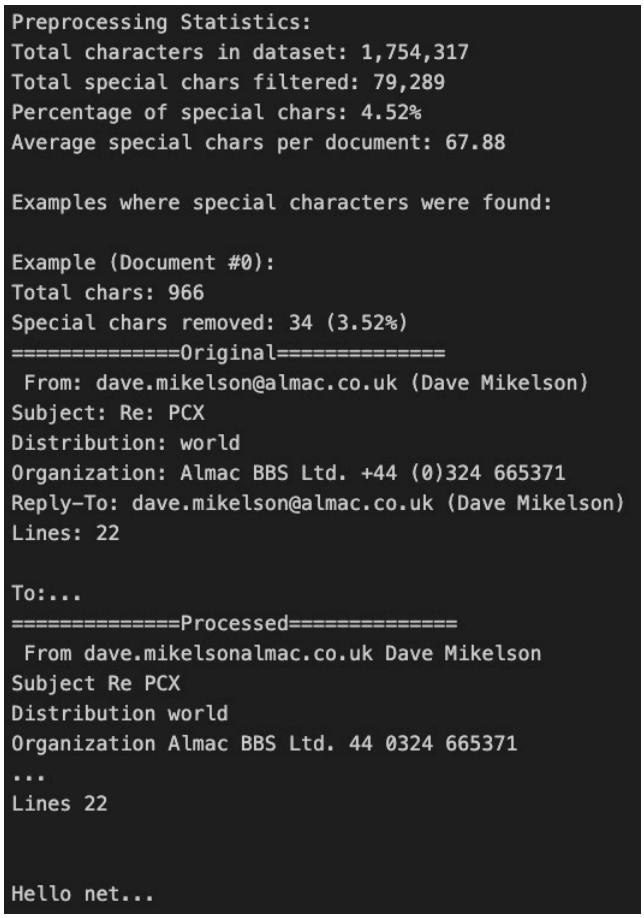
```
Preprocessing Statistics:
Total characters in dataset: 64,258,738
Total special chars filtered: 623,938
Percentage of special chars: 0.97%
Average special chars per review: 12.48

Examples where special characters were found:

Example (Review #0):
Total chars: 284
Special chars removed: 2 (0.70%)
Original: For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni cha...
Processed: For a movie that gets no respect there sure are a lot of memorable quotes listed for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapleton is a scene stealer. The Moroni cha...
```

3. Remove English stop words. We use nltk list of stop words to filter out low meaning words



Here are the preprocessing steps done for 20newsgroups:

1. Remove special char which is any char besides alphabets, numbers, spaces and basic punctuation (“.”, “,”, “!”, “?”)



2. Remove English stop words. We use nltk list of stop words to filter out low meaning words

Dataset split

In this step we want to have a split where more data is allocated to the training set but enough data is there to evaluate the model properly in the test set. We split the dataset into a ratio of 4:1 with 4 being the training split and 1 being the test split. We use sklearn's train_test_split.

Vectorization

In this step we transform the text into numerical representations that can be consumed by the computer and by extension the machine learning algorithms.

In tWe use 3 vectorization:

1. BoW using sklearn's CountVectorizer
2. Tf-IDF using sklearn's TfidfVectorizer
3. Word2Vec using gensim.models

CountVectorizer and Tf-IDF are executed with X_train only to ensure no testing data is leaked to the embeddings. For the word2vec we don't use any pretrained embeddings, instead we create the embeddings by training the word2vec with the X_train split only. Hence all embeddings only make use of X_train, which will make a more fair comparison.

For BoW we use a max features count of 20000 to save RAM usage.

IMDB BoW:

```
Vectorization Statistics:
Number of documents: 50000
Vocabulary size: 20000
Total words: 5,805,122

Top 10 most frequent terms:
         term  count
11819   movie  86993
6860     film  77615
12516     one  53009
10503    like  40113
7749     good  29674
18089    time  25033
6297     even  24832
19796   would  24217
14407  really  23082
15721     see  22981
```

20newsgroup BoW:

```
Vectorization Statistics:
Number of documents: 1168
Vocabulary size: 20000
Total words: 166,757

Top 10 most frequent terms:
              term  count
7022           edu   1991
5255           com   1288
16939      subject   1238
11204        lines   1236
13465  organization   1152
4765            car    871
3559        article    738
19578         would    726
19603        writes    723
18464    university    626
```

Model training

In this step the goal is to train Logistic Regression and Naive Bayes for task classification. For the IMDB dataset we particularly do Sentiment Analysis.

Important thing to highlight are:

1. We use standardized algorithms from sklearn
2. We use LogisticRegression with only max_iter=200 hyper parameter
3. We use MultinomialNB out of the possible choice at sklearn

Model testing

In this step the goal is to find out the effects of vector embedding choice with model accuray on task classification. Here are the results for IMDB:

```
Fitting Logistic Regression to dataset...
Fitting Logistic Regression with TF-IDF features...
Generating word2vec representation of training dataset...
Fitting Logistic Regression with Word2Vec features...

Model Comparison:
CountVectorizer + LogisticRegression accuracy: 0.8848
TF-IDF + LogisticRegression accuracy: 0.9007
Word2Vec + LogisticRegression accuracy: 0.8762
```

```
Fitting MultinomialNB to dataset...
Fitting MultinomialNB with TF-IDF features...
Fitting MultinomialNB with Word2Vec features...

Model Comparison:
CountVectorizer + MultinomialNB accuracy: 0.8578
TF-IDF + MultinomialNB accuracy: 0.8648
Word2Vec + MultinomialNB accuracy: 0.7597
```

And here are the results for Naive Bayes:

```
Fitting Logistic Regression to dataset...
Fitting Logistic Regression with TF-IDF features...
Generating word2vec representation of training dataset...
Fitting Logistic Regression with Word2Vec features...

Model Comparison:
CountVectorizer + LogisticRegression accuracy: 0.9829
TF-IDF + LogisticRegression accuracy: 0.9744
Word2Vec + LogisticRegression accuracy: 0.9145
```

```
Fitting MultinomialNB to dataset...
Fitting MultinomialNB with TF-IDF features...
Fitting MultinomialNB with Word2Vec features...

Model Comparison:
CountVectorizer + MultinomialNB accuracy: 0.9957
TF-IDF + MultinomialNB accuracy: 0.9915
Word2Vec + MultinomialNB accuracy: 0.8120
```