Feature Engineering

Papangkorn Inkeaw, Ph.D.

Feature Extraction for Text

Lab 7

Load dataset

Dataset: Twitter Sentiment Dataset (https://www.kaggle.com/datasets/saurabhshahane/twitter-sentiment-dataset)

The dataset has three sentiments namely, negative(-1), neutral(0), and positive(+1). It contains two fields for the tweet and label.

- Import libraries

```
import pandas as pd
import numpy as np
import string
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
nltk.download('punkt')
```

- Load the twitter sentiment dataset

```
data = pd.read_csv('Twitter_Data.csv’)
data.head()
```

Text Preprocessing

- Read a text from the dataset

```
text = data['clean_text’][0]
```

- Make all characters in text to lower case

```
text = text = text.lower()
```

- Tokenize the text

```
tokens = nltk.tokenize.word_tokenize(text)
```

- Remove nuisance characters

```
punctuationSet = [c for c in string.punctuation]+['’','“','”']
tokenRemovedNuisance = []
for token in tokens:
    if token not in punctuationSet:
            tokenRemovedNuisance.append(token)
```

Text Preprocessing

- Iteratively change each word to its base form by stemming

```
pst = nltk.stem.PorterStemmer()
StemmingWord = []
for word in tokenRemovedNuisance:
    stw = pst.stem(word)
    StemmingWord.append(stw)
```

Text Preprocessing

- Pack all preprocessing steps in a function

```
def my_preprocessing(text):
    text = text = text.lower()
    tokens = nltk.tokenize.word_tokenize(text)
    punctuationSet = [c for c in string.punctuation]+['’','“','”']
    tokenRemovedNuisance = []
    for token in tokens:
            if token not in punctuationSet:
                    tokenRemovedNuisance.append(token)
    pst = nltk.stem.PorterStemmer()
    StemmingWord = []
    for word in tokenRemovedNuisance:
        stw = pst.stem(word)
        StemmingWord.append(stw)
    return ' '.join(StemmingWord)
```

Bag of Words

- Convert data in clean_text column to list

```
copus = data["clean_text"].values.tolist()
```

- Extract BoW vectors (Simple)

```
cv = CountVectorizer(copus)
count_vector = cv.fit_transform(copus)
```

- Extract BoW vectors (Custom dictionary)

```
vocab_dicts = {'cat': 0, 'dog': 1,  'fish':2, 'tiger':3}
cv = CountVectorizer(copus, vocabulary=vocab_dicts)
count_vector = cv.fit_transform(copus)
```

- Extract BoW vectors (Custom tokenization)

```
def my_tokenizer(text):
    # create a space between special characters
    text=re.sub("(\\W)"," \\1 ",text)
    # split based on whitespace
    return re.split("\\s+",text)
cv = CountVectorizer(copus, tokenizer=my_tokenizer)
count_vector=cv.fit_transform(copus)
```

Bag of Words

- Extract BoW vectors (Custom stop word list)

```
stop_words = ["all","in","the","is","and"])
cv = CountVectorizer(copus,stop_words = stop_words)
count_vector=cv.fit_transform(copus)
```

- Extract BoW vectors (Custom preprocessing)

```
cv = CountVectorizer(copus, preprocessor = my_preprocessing)
count_vector=cv.fit_transform(copus)
```

- Extract BoW vectors (Working with n-grams)

```
cv = CountVectorizer(copus, ngram_range=(1,2))
count_vector=cv.fit_transform(copus)
```

- Extract BoW vectors (Working with n-grams and limiting vocabulary size)

```
cv = CountVectorizer(copus, ngram_range=(1,2), max_features=20)
count_vector=cv.fit_transform(copus)
print(cv.vocabulary_)
```

TF-IDF

- Extract TD-IDF vectors (Simple)

```
tfidfv = TfidfVectorizer(copus)
tfidf_vector = tfidfv.fit_transform(copus)
```

- Extract TD-IDF vectors (Custom dictionary)

```
vocab_dicts = {'cat': 0, 'dog': 1,  'fish':2, 'tiger':3}
tfidfv = TfidfVectorizer(copus, vocabulary=vocab_dicts)
tfidf_vector = tfidfv.fit_transform(copus)
```

- Extract TD-IDF vectors (Custom tokenization)

```
tfidfv = TfidfVectorizer(copus, tokenizer=my_tokenizer)
tfidf_vector = tfidfv.fit_transform(copus)
```

TF-IDF

- Extract BoW vectors (Custom stop word list)

```
stop_words = ["all","in","the","is","and"])
tfidfv = TfidfVectorizer(copus,stop_words = stop_words)
tfidf_vector = tfidfv.fit_transform(copus)
```

- Extract BoW vectors (Custom preprocessing)

```
tfidfv = TfidfVectorizer(copus, preprocessor = my_preprocessing)
tfidf_vector = tfidfv.fit_transform(copus)
```

- Extract BoW vectors (Working with n-grams)

```
tfidfv = TfidfVectorizer(copus, ngram_range=(1,2))
tfidf_vector = tfidfv.fit_transform(copus)
```

- Extract BoW vectors (Working with n-grams and limiting vocabulary size)

```
tfidfv = TfidfVectorizer(copus, ngram_range=(1,2), max_features=20)
tfidf_vector = tfidfv.fit_transform(copus)
print(cv.vocabulary_)
```

Your work!

1. Load the SMS Spam Collection Dataset from https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset
2. Split the dataset into training and test dataset (Hint: use *train_test_split* method in sklearn library)
3. Extract feature vectors of samples on both training and test sets
4. Construct a classifier using the training samples for identifying the class of sms (i.e., spam or ham)
5. Evaluate performance of the classifier on the test set
6. Submit your program to the assignment submission system (http://hw.cs.science.cmu.ac.th/).

Note:

- Put your name and student ID in the first cell using comment tag.
- Name your python notebook file with the pattern Lab_07_XXXXXXXXX.py (XXXXXXXXX is your student ID)

References & Study Resources

- https://www.kaggle.com/datasets/saurabhshahane/twitter-sentiment-dataset
- https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset
- https://kavita-ganesan.com/how-to-use-countvectorizer/#.YxhBFexBxAd
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html