Natural Language processing

TP 2 – Sentiment analysis

Prérequis

```
pip install nltk
pip install gensim
```

1. Data preparation

1. Importer Dependencies

```
import re    # for regular expressions
import nltk  # for text manipulation
import string
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore", category=DeprecationWarning)

%matplotlib inline
```

2. Reading data

```
train  = pd.read_csv('./data/train_tweets.csv')
test = pd.read_csv('./data/test_tweets.csv')
```

Text is a highly unstructured form of data, various types of noise are present in it and the data is not readily analyzable without any pre-processing. The entire process of cleaning and standardization of text, making it noise-free and ready for analysis is known as text preprocessing.

3. Let's check out a few non racist/sexist tweets.

```
train[train['label'] == 0].head(10)
```

4. Now check out a few racist/sexist tweets.

```
train[train['label'] == 1].head(10)
```

5. Let's check dimensions of the train and test dataset

```
train.shape, test.shape
```

6. Let's have a glimpse at label-distribution in the train dataset.

```
train["label"].value_counts()
```

7. Now we will check the distribution of length of the tweets, in terms of words, in both train and test data.

```
length_train = train['tweet'].str.len()
length_test = test['tweet'].str.len()
plt.hist(length_train, bins=20, label="train_tweets")
plt.hist(length_test, bins=20, label="test_tweets")
plt.legend()
plt.show()
```

Data cleaning

8. Before we begin cleaning, let's first combine train and test datasets. Combining the datasets will make it convenient for us to preprocess the data. Later we will split it back into train and test data.

```
combi = pd.concat([train,test], axis=0, ignore_index=True)
combi.shape
#Given below is a user-defined function to remove
#unwanted text patterns from the tweets.
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.sub(i, '', input_txt)
    return input_txt
```

9. Removing Twitter Handles

```
combi['tidy_tweet'] = 1
combi['tidy_tweet'] = combi['tweet'].replace('@[\w]*','',regex=True)
```

10. Removing Punctuations, Numbers, and Special Characters

Questions : remove puntuations from `combi['tidy_tweet']` column

11. Remove shrot words : just to filter more tweets, I randomly select to remove words with 3 or less characters

```
combi['tidy_tweet'] = combi['tidy_tweet'].replace(value=' ', regex=r'\b[a-z]{1,3}\b')
combi.head(3)
```

Question : What's the impact of not doing this step ?

12. Define a function to tokenize

```
import nltk
def tokenize(column):
    """Tokenizes a Pandas dataframe column and returns a list of tokens.

    Args:
        column: Pandas dataframe column (i.e. df['text']).

    Returns:
        tokens (list): Tokenized list, i.e. [Donald, Trump, tweets]
    """

    tokens = nltk.word_tokenize(column)
    return [w for w in tokens if w.isalpha()]
```

13. Running tokenization on tweets

```
combi['tidy_tweet'] = combi['tidy_tweet'].astype(str)
tokenized_tweet = combi.apply(lambda x: tokenize(x['tidy_tweet2']), axis=1)
tokenized_tweet.head()
```

14.Normalize tokens with Stemming

```
from nltk.stem.porter import *
stemmer = PorterStemmer()
# stemming
tokenized_tweet = tokenized_tweet.apply(lambda x: [stemmer.stem(i) for i in x])
```

14. Now let's stitch these tokens back together, to do some analysis

```
combi['tidy_tweet'] = tokenized_tweet

combi.head(3)
```

Descriptive analysis of words

15. Understanding the common words used in the tweets: WordCloud

```
from wordcloud import WordCloud

all_words = ' '.join([text for text in combi['tidy_tweet']])
wordcloud = WordCloud(width=800, height=500,
                      random_state=21, max_font_size=110).generate(all_words)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```

16. Words in non racist/sexist tweets : labelling with 0

```
normal_words =' '.join([text for text in combi['tidy_tweet'][combi['label'] == 0]])
wordcloud = WordCloud(width=800, height=500,
                      random_state=21, max_font_size=110).generate(normal_words)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```

17. Words in racist/sexist tweets : labelling with 1

```
normal_words =' '.join([text for text in combi['tidy_tweet'][combi['label'] == 1]])
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(normal_words)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```

18. Understanding the impact of Hashtags on tweets sentiment

```
# function to collect hashtags
def hashtag_extract(x):
    hashtags = []
    # Loop over the words in the tweet
    for i in x:
        ht = re.findall(r"#(\w+)", i)
        hashtags.append(ht)

    return hashtags
```

```
# extracting hashtags from non racist/sexist tweets
HT_regular = hashtag_extract(combi['tweet'][combi['label'] == 0])
# extracting hashtags from racist/sexist tweets
HT_negative = hashtag_extract(combi['tweet'][combi['label'] == 1])
# unnesting list
HT_regular = sum(HT_regular,[])
HT_negative = sum(HT_negative,[])
```

19. Positive hashtags

```
a = nltk.FreqDist(HT_regular)
d = pd.DataFrame({'Hashtag': list(a.keys()),'Count': list(a.values())})
# selecting top 20 most frequent hashtags
d = d.nlargest(columns="Count", n = 20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
plt.show()
```

20. Negative tweet

```
a = nltk.FreqDist(HT_negative)
d = pd.DataFrame({'Hashtag': list(a.keys()),'Count': list(a.values())})
# selecting top 20 most frequent hashtags
d = d.nlargest(columns="Count", n = 20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
plt.show()
```

Word embeddings

21. Import libraries

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import gensim
```

22. BoW implementation (look that max_features is set up to 1000)

```
bow_vectorizer = CountVectorizer(max_df = 0.90, min_df = 2, max_features = 1000, stop_words='english')
bow = bow_vectorizer.fit_transform(combi['tidy_tweet'])
bow.shape
```

23. TF-IDF implementation (look that max_features is set up to 1000)

```
tfidf_vectorizer = TfidfVectorizer(max_df = 0.90, min_df = 2, max_features = 1000, stop_words='english')
tfidf = tfidf_vectorizer.fit_transform(combi['tidy_tweet'])
tfidf.shape
```

Question : what is the meaning of max_features for both models ?

24. Word2vec implementation

```
tokenized_tweet = combi['tidy_tweet'].apply(lambda x: x.split()) # tokenizing
model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            vector_size=200, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2,
            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 2, # no.of cores
            seed = 34)

model_w2v.train(tokenized_tweet, total_examples= len(combi['tidy_tweet']), epochs=20)
```

Question : what are the meaning of the following hyperparameters : vector_size, wondow, negative)

25. Let's find the similarity with a random word :

```
model_w2v.wv.most_similar(positive="dinner")
```

26. And now with another word

```
model_w2v.wv.most_similar(positive="trump")
```

Question : do the similar word makes sense ? how to intepret the probability given by the similarity measurement ?

27. Printing a vector from w2v

```
model_w2v.wv['food']
```

Modelling doc2vec

28. Preparing Vectors for Tweets

```
def word_vector(tokens, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in tokens:
        try:
            vec += model_w2v.wv[word].reshape((1, size))
            count += 1.
        except KeyError: # handling the case where the token is not in vocabulary
            continue
    if count != 0:
        vec /= count
    return vec
```

29. Preparing feature set

```
wordvec_arrays = np.zeros((len(tokenized_tweet), 200))
for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector(tokenized_tweet[i], 200)
    wordvec_df = pd.DataFrame(wordvec_arrays)

wordvec_df.shape
```

30. Import libraries

```
#Let's load the required libraries.
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")
from gensim.models.doc2vec import TaggedDocument as TD
```

31. To implement doc2vec, we have to labelise or tag each tokenised tweet with unique IDs.

```
def add_label(twt):
    output = []
    for i, s in zip(twt.index, twt):
        output.append(TD(s, ["tweet_" + str(i)]))
    return output

labeled_tweets = add_label(tokenized_tweet) # label all the tweets
```

```
#Let's have a look at the result.
labeled_tweets[:6]
```

32. Running doc2vec model

```
#Now let's train a doc2vec model.
model_d2v = gensim.models.Doc2Vec(
    dm=1, # dm = 1 for 'distributed memory' model
    dm_mean=1, # dm = 1 for using mean of the context word vectors
    #size=200, # no. of desired features
    window=5, # width of the context window
    negative=7, # if > 0 then negative sampling will be used
    min_count=5, # Ignores all words with total frequency lower than 2.
    workers=3, # no. of cores
    alpha=0.1, # learning rate
    seed = 23)

model_d2v.build_vocab([i for i in tqdm(labeled_tweets)])
model_d2v.train(labeled_tweets, total_examples= len(combi['tidy_tweet']), epochs=15)
```

33. Preparing feature set

```
#Preparing doc2vec Feature Set
docvec_arrays = np.zeros((len(tokenized_tweet), 100))
for i in range(len(combi)):
    docvec_arrays[i,:] = model_d2v.docvecs[i].reshape((1,100))

docvec_df = pd.DataFrame(docvec_arrays)
docvec_df.shape
```

Classification of tweets

Logistic regression

34. Import libraries

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
```

35. Modelling using BoW

```
# Extracting train and test BoW features
train_bow = bow[:31962,:]
test_bow = bow[31962:,:]
# splitting data into training and validation set
xtrain_bow, xvalid_bow, ytrain, yvalid = train_test_split(train_bow, train['label'],
lreg = LogisticRegression()
# training the model
lreg.fit(xtrain_bow, ytrain)
```

36. Prediction on train and test set

```
prediction = lreg.predict_proba(xvalid_bow) # predicting on the validation set
prediction_int = prediction[:,1] >= 0.3 # if prediction is greater than or equal to 0.3 than 1 else 0
#prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int) # calculating f1 score for the validation set
```

```
#Now let's make predictions for the test dataset and create a submission file.
test_pred = lreg.predict_proba(test_bow)
test_pred_int = test_pred[:,1] >= 0.3
#test_pred_int = test_pred_int.astype(np.int)
test['label'] = test_pred_int
submission = test[['id','label']]
submission.to_csv('sub_lreg_bow.csv', index=False) # writing data to a CSV file
```

37. MOdelling using TF-IDF

```
train_tfidf = tfidf[:31962,:]
test_tfidf = tfidf[31962:,:]
xtrain_tfidf = train_tfidf[ytrain.index]
xvalid_tfidf = train_tfidf[yvalid.index]
lreg.fit(xtrain_tfidf, ytrain)
prediction = lreg.predict_proba(xvalid_tfidf)
prediction_int = prediction[:,1] >= 0.3
#prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int) # calculating f1 score for the validation set
```

38. Modelling using word2vec

```
train_w2v = wordvec_df.iloc[:31962,:]
test_w2v = wordvec_df.iloc[31962:,:]
xtrain_w2v = train_w2v.iloc[ytrain.index,:]
xvalid_w2v = train_w2v.iloc[yvalid.index,:]
lreg.fit(xtrain_w2v, ytrain)
prediction = lreg.predict_proba(xvalid_w2v)
prediction_int = prediction[:,1] >= 0.3
#prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int)
```

39. Modelling using doc2vec

```
train_d2v = docvec_df.iloc[:31962,:]
test_d2v = docvec_df.iloc[31962:,:]
xtrain_d2v = train_d2v.iloc[ytrain.index,:]
xvalid_d2v = train_d2v.iloc[yvalid.index,:]
lreg.fit(xtrain_d2v, ytrain)
prediction = lreg.predict_proba(xvalid_d2v)
prediction_int = prediction[:,1] >= 0.3
#prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int)
```

SVM

40. Import libraries

```
from sklearn import svm
```

41. Modelling with BoW

```
svc = svm.SVC(kernel='linear', C=1, probability=True).fit(xtrain_bow, ytrain)
prediction = svc.predict_proba(xvalid_bow)
prediction_int = prediction[:,1] >= 0.3
#prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int)
```

```
test_pred = svc.predict_proba(test_bow)
test_pred_int = test_pred[:,1] >= 0.3
#test_pred_int = test_pred_int.astype(np.int)
test['label'] = test_pred_int
submission = test[['id','label']]
submission.to_csv('sub_svm_bow.csv', index=False)
```

42. Modelling with TF-IDF

```
svc = svm.SVC(kernel='linear',
C=1, probability=True).fit(xtrain_tfidf, ytrain)
prediction = svc.predict_proba(xvalid_tfidf)
prediction_int = prediction[:,1] >= 0.3
#prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int)
```

43. Modelling with word2vec

```
svc = svm.SVC(kernel='linear', C=1, probability=True).fit(xtrain_w2v, ytrain)
prediction = svc.predict_proba(xvalid_w2v)
prediction_int = prediction[:,1] >= 0.3
#prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int)
```

44. MOdelling with doc2vec

```
svc = svm.SVC(kernel='linear', C=1, probability=True).fit(xtrain_d2v, ytrain)
prediction = svc.predict_proba(xvalid_d2v)
prediction_int = prediction[:,1] >= 0.3
#prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int)
```

Random forest

45. Import libraries

```
from sklearn.ensemble import RandomForestClassifier
```

46. Modelling with BoW

```
rf = RandomForestClassifier(n_estimators=400, random_state=11).fit(xtrain_bow, ytrain)
prediction = rf.predict(xvalid_bow)
# validation score
f1_score(yvalid, prediction)
```

```
#Let's make predictions for the test dataset and create another submission file.
test_pred = rf.predict(test_bow)
test['label'] = test_pred
submission = test[['id','label']]
submission.to_csv('sub_rf_bow.csv', index=False)
```

47. Modelling with TF-IDF

```
rf = RandomForestClassifier(n_estimators=400, random_state=11).fit(xtrain_tfidf, ytrain)
prediction = rf.predict(xvalid_tfidf)
f1_score(yvalid, prediction)
```

48. Modelling with word2vec

```
rf = RandomForestClassifier(n_estimators=400, random_state=11).fit(xtrain_w2v, ytrain)
prediction = rf.predict(xvalid_w2v)
f1_score(yvalid, prediction)
```

49. Modelling with doc2vec

```
rf = RandomForestClassifier(n_estimators=400, random_state=11).fit(xtrain_d2v, ytrain)
prediction = rf.predict(xvalid_d2v)
f1_score(yvalid, prediction)
```

XGBoost

50. Import libraries

```
from xgboost import XGBClassifier
```

51. Modelling with BoW

```
xgb_model = XGBClassifier(max_depth=6, n_estimators=1000).fit(xtrain_bow, ytrain)
prediction = xgb_model.predict(xvalid_bow)
f1_score(yvalid, prediction)
```

```
test_pred = xgb_model.predict(test_bow)
test['label'] = test_pred
submission = test[['id','label']]
submission.to_csv('sub_xgb_bow.csv', index=False)
```

52. Modelling with TF-IDF

```
xgb = XGBClassifier(max_depth=6, n_estimators=1000).fit(xtrain_tfidf, ytrain)
prediction = xgb.predict(xvalid_tfidf)
f1_score(yvalid, prediction)
```

53. Modelling with word2vec

```
xgb = XGBClassifier(max_depth=6, n_estimators=1000, nthread= 3).fit(xtrain_w2v, ytrain)
prediction = xgb.predict(xvalid_w2v)
f1_score(yvalid, prediction)
```

54. Modelling doc2vec

```
xgb = XGBClassifier(max_depth=6, n_estimators=1000, nthread= 3).fit(xtrain_d2v, ytrain)
prediction = xgb.predict(xvalid_d2v)
f1_score(yvalid, prediction)
```

Questions : On each of the previous models built :

1. **Which one ws the best and why ?**
2. **For each model, suggest and implement an improvement**