Lab 3 - Textual Data Analytics

Complete the code with TODO tag.

1. Feature Engineering

In this exercise we will understand the functioning of TF/IDF ranking. Implement the feature engineering and its application, based on the code framework provided below.

First we use textual data from Twitter.

In [1]:
```
import numpy as np
import pandas as pd
data = pd.read_csv('elonmusk_tweets.csv')
print(len(data))
data.head()
```

```
2819
```

Out[1]:

	id	created_at	text
0	849636868052275200	2017-04-05 14:56:29	b'And so the robots spared humanity ... https:...
1	848988730585096192	2017-04-03 20:01:01	b"@ForIn2020 @waltmossberg @mims @defcon_5 Exa...
2	848943072423497728	2017-04-03 16:59:35	b'@waltmossberg @mims @defcon_5 Et tu, Walt?'
3	848935705057280001	2017-04-03 16:30:19	b'Stormy weather in Shortville ...'
4	848416049573658624	2017-04-02 06:05:23	b"@DaveLeeBBC @verge Coal is dying due to nat ...

1.1. Text Normalization

Now we need to normalize text by stemming, tokenizing, and removing stopwords.

In [2]:
```
from __future__ import print_function, division
from nltk.stem import PorterStemmer, WordNetLemmatizer
import nltk
nltk.download('punkt')
import string
from nltk.corpus import stopwords
import math
from collections import Counter
nltk.download('stopwords')
import pprint
pp = pprint.PrettyPrinter(indent=4)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/adamnguyen/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/adamnguyen/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [3]:
```
def normalize(document):
    # TODO: remove punctuation
    text = "".join([ch for ch in document if ch not in string.punctuation])

    # TODO: tokenize text
    tokens = nltk.word_tokenize(text)

    # TODO: Stemming
    stemmer = PorterStemmer()
    ret = " ".join([stemmer.stem(word.lower()) for word in tokens])
    return ret

original_documents = [x.strip() for x in data['text']]
documents = [normalize(d).split() for d in original_documents]
documents[0]
```

Out[3]:
```
['band', 'so', 'the', 'robot', 'spare', 'human', 'httpstcov7jujqwfcv']
```

As you can see that the normalization is still not perfect. Please feel free to improve upon (OPTIONAL), e.g. https://marcobonzanini.com/2015/03/09/mining-twitter-data-with-python-part-2/

1.2. Implement TF-IDF

Now you need to implement TF-IDF, including creating the vocabulary, computing term frequency, and normalizing by tf-idf weights.

In [4]:
```
# Flatten all the documents
flat_list = [word for doc in documents for word in doc]

# TODO: remove stop words from the vocabulary
words = [word for word in flat_list if word not in stopwords.words('english')]

# TODO: we take the 500 most common words only
counts = Counter(words)
vocabulary = counts.most_common(500)
print([x for x in vocabulary if x[0] == 'tesla'])
vocabulary = [x[0] for x in vocabulary]
assert len(vocabulary) == 500

# vocabulary.sort()
vocabulary[:5]
```

```
[('tesla', 287)]
```

Out[4]:
```
['brt', 'tesla', 'spacex', 'model', 'thi']
```

In [5]:
```
def tf(vocabulary, documents):
    matrix = [0] * len(documents)
    for i, document in enumerate(documents):
        counts = Counter(document)
        matrix[i] = [0] * len(vocabulary)
        for j, term in enumerate(vocabulary):
            matrix[i][j] = counts[term]
    return matrix

tf = tf(vocabulary, documents)
np.array(vocabulary)[np.where(np.array(tf[1]) > 0)], np.array(tf[1])[np.where(np.array(tf[1]) > 0)]
```

Out[5]:
```
(array(['tesla', 'exactli'], dtype='<U17'), array([1, 1]))
```

In [6]:
```
def idf(vocabulary, documents):
    """TODO: compute IDF, storing values in a dictionary"""
    idf = {}
    num_documents = len(documents)
    for i, term in enumerate(vocabulary):
        idf[term] = math.log(num_documents / sum(term in document for document in documents), 2)
    return idf

idf = idf(vocabulary, documents)
[idf[key] for key in vocabulary[:5]]
```

Out[6]:
```
[2.539126825495932,
 3.3163095197385393,
 3.7262581423445837,
 3.8171115727956972,
 3.8027562798186274]
```

In [7]:
```
def vectorize(document, vocabulary, idf):
    vector = [0]*len(vocabulary)
    counts = Counter(document)
    for i,term in enumerate(vocabulary):
        vector[i] = idf[term] * counts[term]
    return vector

document_vectors = [vectorize(s, vocabulary, idf) for s in documents]
np.array(vocabulary)[np.where(np.array(document_vectors[1]) > 0)], np.array(document_vectors[1])[np.whe
re(np.array(document_vectors[1]) > 0)]
```

Out[7]:
```
(array(['tesla', 'exactli'], dtype='<U17'), array([3.31630952, 6.65361284]))
```

1.3. Compare the results with the reference implementation of scikit-learn library.

Now we use the scikit-learn library. As you can see that, the way we do text normalization affects the result. Feel free to further improve upon (OPTIONAL), e.g. https://stackoverflow.com/questions/36182502/add-stemming-support-to-countvectorizer-sklearn

In [8]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

tfidf = TfidfVectorizer(analyzer='word', ngram_range=(1,1), min_df = 1, stop_words = 'english', max_fea
tures=500)

features = tfidf.fit(original_documents)
corpus_tf_idf = tfidf.transform(original_documents)

sum_words = corpus_tf_idf.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in tfidf.vocabulary_.items()]
print(sorted(words_freq, key = lambda x: x[1], reverse=True)[:5])
print('testla', corpus_tf_idf[1, features.vocabulary_['tesla']])
```

```
[('http', 163.54366542841234), ('https', 151.85039944652075), ('rt', 112.61998731390989), ('tesla', 9
5.96401470715628), ('xe2', 88.20944486346477)]
testla 0.3495243100660956
```

1.4. Apply TF-IDF for information retrieval

We can use the vector representation of documents to implement an information retrieval system. We test with the query Q = "tesla nasa"

In [9]:
```
def cosine_similarity(v1,v2):
    """TODO: compute cosine similarity"""
    sumxx, sumxy, sumyy = 0, 0, 0
    for i in range(len(v1)):
        x = v1[i]; y = v2[i]
        sumxx += x*x
        sumyy += y*y
        sumxy += x*y
    if sumxy == 0:
            result = 0
    else:
            result = sumxy/math.sqrt(sumxx*sumyy)
    return result

def search_vec(query, k, vocabulary, stemmer, document_vectors, original_documents):
    q = query.split()
    q = [stemmer.stem(w) for w in q]
    query_vector = vectorize(q, vocabulary, idf)

    # TODO: rank the documents by cosine similarity
    scores = [[cosine_similarity(query_vector, document_vectors[d]), d] for d in range(len(document_vec
tors))]
    scores.sort(key=lambda x: -x[0])

    print('Top-{0} documents'.format(k))
    for i in range(k):
        print(i, original_documents[scores[i][1]])

query = "tesla nasa"
stemmer = PorterStemmer()
search_vec(query, 5, vocabulary, stemmer, document_vectors, original_documents)
```

```
Top-5 documents
0 b'@ashwin7002 @NASA @faa @AFPAA We have not ruled that out.'
1 b'RT @NASA: Updated @SpaceX #Dragon #ISS rendezvous times: NASA TV coverage begins Sunday at 3:30am
ET: http://t.co/qrm0Dz4jPE. Grapple at  ...'
2 b"Deeply appreciate @NASA's faith in @SpaceX. We will do whatever it takes to make NASA and the Ame
rican people proud."
3 b'Would also like to congratulate @Boeing, fellow winner of the @NASA commercial crew program'
4 b"@astrostephenson We're aiming for late 2015, but NASA needs to have overlapping capability to be
safe. Would do the same"
```

We can also use the scikit-learn library to do the retrieval.

In [10]:
```
new_features = tfidf.transform([query])

cosine_similarities = linear_kernel(new_features, corpus_tf_idf).flatten()
related_docs_indices = cosine_similarities.argsort()[::-1]

topk = 5
print('Top-{0} documents'.format(topk))
for i in range(topk):
    print(i, original_documents[related_docs_indices[i]])
```

```
Top-5 documents
0 b'@ashwin7002 @NASA @faa @AFPAA We have not ruled that out.'
1 b"SpaceX could not do this without NASA. Can't express enough appreciation. https://t.co/uQpI60zAV
7"
2 b'@NASA launched a rocket into the northern lights http://t.co/tR2cSeMV'
3 b'Whatever happens today, we could not have done it without @NASA, but errors are ours alone and me
most of all.'
4 b'RT @NASA: Updated @SpaceX #Dragon #ISS rendezvous times: NASA TV coverage begins Sunday at 3:30am
ET: http://t.co/qrm0Dz4jPE. Grapple at  ...'
```