cleaned_topic_modeling_and_similarities_with_W2Vec_tfidf_LDA-checkpoint

August 21, 2021

1 Topic Modeling and Similarities

- Latent Dirichlet Allocation (LDA) with Python

[1]:
```
import numpy as np
import sys, re
import nltk
import gensim

import glob, sys

data_dir = './datasets/bbc_sports/'

from nlp_utils import prepare_text
```

1.0.1 Load, Preprocess and Tokenize the document

[2]:
```
def read_data_files(data_dir):
    all_data_files = glob.glob(data_dir+'*')
    raw_doc = []
    for file in all_data_files:
        # Use try and except method as some files may not be readable
        try:
            f = open(file, 'r', encoding='utf-8')
            raw_doc.append(f.read())
        except:
            print (f"skipping the unreadable file:  {file}")
            pass
    return raw_doc

raw_doc = read_data_files(data_dir)
print (f"Total # of documents: {len(raw_doc)}")
```

```
skipping the unreadable file:  ./datasets/bbc_sports/199.txt
Total # of documents: 510
```

[3]:
```
def prepare_data(raw_doc):
    tokenized_doc = [prepare_text(doc, TOKENIZE=True, STEM=True) for doc in␣
 ↪raw_doc]
    dictionary    = gensim.corpora.Dictionary(tokenized_doc)
    numerical_corpus = [dictionary.doc2bow(text) for text in tokenized_doc]
    return (dictionary, tokenized_doc, numerical_corpus)

def model_lda(numerical_corpus, dictionary):
    return gensim.models.LdaModel(corpus=numerical_corpus, num_topics=10,␣
 ↪id2word=dictionary)

def model_tfidf(numerical_corpus, dictionary):
    tf_idf = gensim.models.TfidfModel(numerical_corpus, id2word=dictionary)
    similarity_object = gensim.similarities.Similarity(data_dir,
                                                       tf_idf[numerical_corpus],
                                                       ␣
 ↪num_features=len(dictionary))
    return tf_idf, similarity_object

def model_w2v(tokenized_doc):
    w2v_model = gensim.models.Word2Vec(
                sentences=tokenized_doc,
                size=300, # The size of the dense vector to represent each␣
 ↪token or word
                window=10, # The maximum distance between the target word and␣
 ↪its neighboring word.
                min_count=5, # Minimium frequency count of words. The model␣
 ↪would ignore words with counts< min_count
                workers=10) # How many threads to use behind the scenes

    w2v_model.train(tokenized_doc, total_examples=len(tokenized_doc), epochs=15)
    return w2v_model

def save_model(model, model_name):

    import joblib
    joblib.dump(model, model_name)
    #model can be loaded as
    #joblib.load(model_file_joblib)
    print (f"model: {model} is saved to {model_name}")

    #import pickle
    #model_file_pickle = './datasets/tfidf_model.p'
    #pickle.dump(model, open(model_name, 'wb'))
```

[4]:
```
def sort_list(A, key=0):
    return sorted(A, reverse=True, key=lambda x: x[key] )

def print_top_k(topics, all_topics, k=2):
    topics_sorted = sort_list(topics, key=1)
    for i, topics in enumerate(topics_sorted[:k]):
        idx = topics[0]
        print ("\n",i, all_topics[idx])

def test_lda(raw_doc):
    (dictionary, tokenized_doc, numerical_corpus) = prepare_data(raw_doc)
    model = model_lda(numerical_corpus, dictionary)

    model_name = './models/lda_model'
    save_model(model, model_name)

    all_topics = model.print_topics()

    print( "\n first 10 most representative topics")
    for i in range(10):
        print(f"\nTopic #{i} : {model.print_topic(i, 5 )}")

    doc_new = "My wife plans to go out tonight."
    doc_new_prepared = prepare_text(doc_new, TOKENIZE=True)
    print ( doc_new_prepared )
    doc_bow = dictionary.doc2bow(doc_new_prepared)
    print (doc_bow)

    topics= model.get_document_topics( doc_bow )
    top_k_topics = print_top_k(topics, all_topics, k=2)

test_lda(raw_doc)
```

```
model: LdaModel(num_terms=10321, num_topics=10, decay=0.5, chunksize=2000) is
saved to ./models/lda_model

 first 10 most representative topics

Topic #0 : 0.008*"play" + 0.007*"said" + 0.005*"cup" + 0.005*"last" +
0.005*"world"

Topic #1 : 0.011*"said" + 0.007*"player" + 0.006*"england" + 0.006*"game" +
0.006*"win"

Topic #2 : 0.008*"said" + 0.008*"win" + 0.006*"play" + 0.006*"game" +
0.005*"two"
```

```
Topic #3 : 0.012*"said" + 0.006*"win" + 0.006*"game" + 0.006*"play" +
0.005*"player"

Topic #4 : 0.010*"said" + 0.007*"play" + 0.007*"game" + 0.006*"win" + 0.005*"go"

Topic #5 : 0.006*"first" + 0.005*"play" + 0.005*"champion" + 0.005*"said" +
0.005*"win"

Topic #6 : 0.007*"said" + 0.006*"game" + 0.006*"back" + 0.006*"first" +
0.005*"play"

Topic #7 : 0.009*"said" + 0.009*"game" + 0.006*"player" + 0.005*"play" +
0.005*"year"

Topic #8 : 0.007*"said" + 0.006*"england" + 0.005*"v" + 0.005*"last" +
0.005*"world"

Topic #9 : 0.009*"said" + 0.008*"play" + 0.007*"win" + 0.007*"game" +
0.006*"would"
['wife', 'plans', 'go', 'tonight']
[(57, 1), (2074, 1), (5826, 1)]

 0 (5, '0.006*"first" + 0.005*"play" + 0.005*"champion" + 0.005*"said" +
0.005*"win" + 0.005*"year" + 0.004*"ireland" + 0.004*"world" + 0.004*"game" +
0.004*"second"')

 1 (8, '0.007*"said" + 0.006*"england" + 0.005*"v" + 0.005*"last" +
0.005*"world" + 0.004*"first" + 0.004*"win" + 0.004*"game" + 0.004*"ireland" +
0.004*"year"')
```

[5]:
```python
def test_tfidf(raw_doc):
    (dictionary, tokenized_doc, numerical_corpus) = prepare_data(raw_doc)
    tf_idf, similarity_object = model_tfidf(numerical_corpus, dictionary)

    model_name = './models/tfidf_model'
    save_model(tf_idf, model_name)

    #query_text
    q_text = raw_doc[8]
    q_text_processed = prepare_text(q_text, TOKENIZE=True, STEM=True)
    print ( "\nfirst 10 tokens:\n",q_text_processed[:10])
    q_text_bow = dictionary.doc2bow(q_text_processed)
    print ( "\nfirst 10 bow:\n",q_text_bow[:10])
    q_text_tfidf = tf_idf[q_text_bow]
    print ( "\nfirst 10 tfidf:\n",q_text_tfidf[:10] )
    similarity_scores=list(similarity_object[q_text_tfidf])
    print ( "\nfirst 10 similarity scores:\n", similarity_scores[:10])
```

```
    max_score = max(similarity_scores)
    max_score_index = similarity_scores.index(max_score)

    print (f"\nmax score {max_score} and max score index {max_score_index}")

    sorted_score = sorted(similarity_scores, reverse=True)

    indices = []
    for i in range(3):
        score = sorted_score[i]
        indx  = similarity_scores.index(score)
        print ( f"\nscore: {score} index:{indx}")

        indices.append(indx)

    print ("\nPritining the docs with very similar similarity score\n")
    print ("\n", raw_doc[indices[0]][:100] )
    print ("\n\n", raw_doc[indices[1]][:100] )
    print ("\n\n", raw_doc[indices[1]][:100] )

test_tfidf(raw_doc)
```

```
model: TfidfModel(num_docs=510, num_nnz=67925) is saved to ./models/tfidf_model

first 10 tokens:
 ['robben', 'sidelin', 'broken', 'footchelsea', 'winger', 'arjen', 'robben',
'broken', 'two', 'metatars']

first 10 bow:
 [(23, 3), (45, 1), (46, 1), (50, 1), (57, 1), (71, 1), (94, 1), (106, 1), (110,
1), (111, 1)]

first 10 tfidf:
 [(23, 0.060779464016390256), (45, 0.07075027144828656), (46,
0.014605368204018868), (50, 0.028797575313044728), (57, 0.015421487398221299),
(71, 0.0404114823434502), (94, 0.015008913499203016), (106,
0.004434193825346702), (110, 0.020478063934517673), (111, 0.026530003463721384)]

first 10 similarity scores:
 [0.024998276, 0.020571105, 0.009893991, 0.016575314, 0.014618116, 0.013003329,
0.020246074, 0.011348683, 1.0000001, 0.022904249]

max score 1.0000001192092896 and max score index 8

score: 1.0000001192092896 index:8
```

```
score: 0.2723110318183899 index:220

score: 0.1732272207736969 index:113

Pritining the docs with very similar similarity score


 Robben sidelined with broken foot

Chelsea winger Arjen Robben has broken two metatarsal bones in hi


 Robben plays down European return

Injured Chelsea winger Arjen Robben has insisted that he only has


 Robben plays down European return

Injured Chelsea winger Arjen Robben has insisted that he only has
```

1.1 Word2Vec Model

[6]:
```
def test_word2vec(raw_doc):
    (dictionary, tokenized_doc, numerical_corpus) = prepare_data(raw_doc)
    model = model_w2v(tokenized_doc)

    model_name = './models/w2v_model'
    save_model(model, model_name)

    words = list(model.wv.vocab)
    print (f"\nThere are {len(words)} words. First 10 words:\n{words[:10]}")

    vectors = np.array([model.wv[word] for word in words])
    print (f"\nvectors.shape:{vectors.shape}")

    my_word = 'defend'
    sim_word = model.wv.most_similar(positive=my_word)
    print (f"\ntop 3 similar words to {my_word} are {sim_word[:3]}")

    v=model.wv[my_word]
    idx=dictionary.doc2idx([my_word])[0]
    print (f"\ndictionary index: {idx}")
    print (f"\nsanity check for word {my_word} index: {idx}^th item in the␣
 ↪dictionary: {dictionary[idx]}")

    word_pairs = [["celtic", "everton"],
```

```
                ["good", "bad"],
                ["good", "celtic"],
                ["good", "good"],
                ["kid", "men"] ]

    for (w1, w2) in word_pairs:
        simi_score = model.wv.similarity(w1=w1, w2=w2)
        print (f"\nSimilarity score of ## Word2Vec Model## '{w1}' and '{w2}' :␣
↪{simi_score}")

test_word2vec(raw_doc)
```

```
model: Word2Vec(vocab=2817, size=300, alpha=0.025) is saved to
./models/w2v_model

There are 2817 words. First 10 words:
['robinson', 'blast', 'coach', 'andi', 'insist', 'livid', 'side', 'deni', 'two',
'tri']

vectors.shape:(2817, 300)

top 3 similar words to defend are [('rosenborg', 0.8802071809768677),
('burnley', 0.8630086779594421), ('milan', 0.8540078401565552)]

dictionary index: 31

sanity check for word defend index: 31^th item in the dictionary: defend

Similarity score of ## Word2Vec Model## 'celtic' and 'everton' :
0.8854888677597046

Similarity score of ## Word2Vec Model## 'good' and 'bad' : 0.711058497428894

Similarity score of ## Word2Vec Model## 'good' and 'celtic' :
0.11172105371952057

Similarity score of ## Word2Vec Model## 'good' and 'good' : 1.0

Similarity score of ## Word2Vec Model## 'kid' and 'men' : 0.16241569817066193
```

[ ]:

[ ]: