latent_semantic_analysis

December 2, 2020

0.1 Imports

[1]:

```
# Std imports
import operator
from itertools import combinations
# read data
import pandas as pd
# preprocessing
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
# SVD
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
from gensim.models import KeyedVectors
# Visualization
import matplotlib.pyplot as plt
```

0.2 Read Data

[2]:

```
# filename = "sleep.txt"
# filename = "concussion.txt"
filename = "mental_health.txt"

fp = open(filename, 'r')

data = fp.readlines()

fp.close()
```

[3]:

```
# Converting paragraphs to sentences
sentences = []

for d in data:
    sentence_list = nltk.sent_tokenize(d)
    sentences.extend(sentence_list)
```

```python
len(sentences)
```

[3]:

```
142
```

[4]:

```python
dict = {'text': sentences}

df = pd.DataFrame(dict)
print(df.head(), df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142 entries, 0 to 141
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    142 non-null    object
dtypes: object(1)
memory usage: 1.2+ KB
                               text
0          Social anxiety disorder.
1    Obsessive compulsive disorder.
2        Major depressive disorder.
3  Borderline personality disorder.
4   Post-traumatic stress disorder. None
```

0.3 Preprocess Text

0.3.1 1. Remove punctuation (,/.!’?”)

0.3.2 2. Convert to lower case

0.3.3 3. Text tokenization

0.3.4 4. Remove stop words

0.3.5 5. Text leminization

0.3.6 6. Bag of words

[5]:

```python
# remove punctuations
df['processed_text'] = df['text'].str.replace('[^\w\s]','')

print(df['processed_text'])
```

```
0                                 Social anxiety disorder
1                           Obsessive compulsive disorder
2                               Major depressive disorder
```

```
3                         Borderline personality disorder
4                           Posttraumatic stress disorder
                                        …
137    Finally transdiagnostic research and practice …
138    Coordinated efforts are required to ensure tha…
139    In particular our trials need to be designed t…
140    Overall we need to better reflect the personal…
141    Transdiagnostic approaches potentially provide…
Name: processed_text, Length: 142, dtype: object
```

[6]:

```
# Remove stopwords
stop_words = set(nltk.corpus.stopwords.words('english'))

df['processed_text'] = df['processed_text'].apply(lambda x: ' '.join([word for␣
 ↪word in x.lower().split() if word not in stop_words]))
df.head()
```

[6]:

```
                                text                   processed_text
0          Social anxiety disorder.          social anxiety disorder
1    Obsessive compulsive disorder.    obsessive compulsive disorder
2        Major depressive disorder.        major depressive disorder
3  Borderline personality disorder.  borderline personality disorder
4   Post-traumatic stress disorder.    posttraumatic stress disorder
```

[7]:

```
# from SKLearn docs
# class for tokenizing and lemmatizing
class LemmaTokenizer():
    def __init__(self):
        self.wnl = nltk.stem.WordNetLemmatizer()

    def __call__(self, doc):
        return [self.wnl.lemmatize(t) for t in nltk.word_tokenize(doc)]
```

[8]:

```
# TfIDFVectorizer will calculate the TfIDF weights that can be fed into the LSA␣
 ↪Model.
vect = TfidfVectorizer(analyzer='word',
                       tokenizer=LemmaTokenizer(),
                       lowercase='True',
                       max_df = 0.5,
                       smooth_idf=True)

x = vect.fit_transform(df['processed_text'])
x.shape
```

[8]:

```
(142, 974)
```

0.4 Topic Modelling - LSA

0.4.1 Evaluation (Topic Coherence - TC-W2V)

[9]:

```
## Load pretrained Word2Vec model from file
# It is a pretrained Word2Vec model. It was trained on Google news dataset.␣
↪Since the input dataset is an article on # health issues and news usuallly␣
↪talk about them, the news model would be suitable for the dataset.
# To download the below model, please visit: https://drive.google.com/file/d/
↪0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
w2v_model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.
↪bin', binary=True)
```

[10]:

```
# http://derekgreene.com/slides/topic-modelling-with-scikitlearn.pdf
def get_mean_coherence(w2v_model, term_rankings):
    overall_coherence = 0.0

    for topic_index in range(len(term_rankings)):
        # check each pair of terms
        pair_scores = []

        for pair in combinations( term_rankings[topic_index], 2 ):
            try:
                pair_scores.append( w2v_model.similarity(pair[0], pair[1]) )
            except KeyError:
                continue

        # get the mean for all pairs in this topic
        topic_score = sum(pair_scores) / len(pair_scores)
        overall_coherence += topic_score

    # mean coherence across all topics
    mean_coherence = overall_coherence / len(term_rankings)

    return mean_coherence
```

0.4.2 Hyperparameters tuning

[11]:

```
# Returns the n_top_words words for each topic as a list of list
def get_term_rankings(vect, svd, n_top_words):
    term_rankings = []

    # Appending n_top_words of each topic to term_rankings
    terms = vect.get_feature_names()
    for topic_idx, topic in enumerate(svd.components_):
        top_terms = [terms[i] for i in topic.argsort()[:-n_top_words - 1:-1]]
        term_rankings.append(top_terms)

    return term_rankings
```

[12]:

```
# Function to print the top n_top_words for each topic
def print_top_words(model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(model.components_):
        message = "Topic #%d: " % topic_idx
        message += " ".join([feature_names[i]
                             for i in topic.argsort()[:-n_top_words - 1:-1]])
        print(message)
```

[13]:

```
# Grid Search to find best parameters for LSA model
topic_count = [2, 3, 4, 5, 6, 7, 8, 9, 10]
hyperparameters_score_dict = {}

for ntopic in topic_count:
    print("Topics: ", ntopic)
    svd = TruncatedSVD(n_components=ntopic, algorithm="randomized")
    svd.fit(x)
    feature_names = vect.get_feature_names()
    term_rankings = get_term_rankings(vect, svd, 10)
    mu_coherence = get_mean_coherence(w2v_model, term_rankings)
    hyperparameters_score_dict[ntopic] = round(mu_coherence, 4)
    print(mu_coherence)
```

```
Topics:  2
0.17825785818018225
Topics:  3
0.1401333979167113
Topics:  4
0.13965057526883257
Topics:  5
0.15815332830624862
Topics:  6
0.15181779327017633
Topics:  7
0.15073007720373696
Topics:  8
0.1358580873493338
Topics:  9
0.1459110670632681
Topics:  10
0.14123182120766226
```

[14]:

```python
# Retrieve best parameters
best_n_topic = max(hyperparameters_score_dict.items(), key=operator.
 ↪itemgetter(1))[0]
best_n_topic
```

[14]:

```
2
```

[15]:

```python
# SVD modeling - SVD with best parameters
svd_model = TruncatedSVD(n_components=best_n_topic, algorithm='randomized')

svd_model.fit(x)
```

[15]:

```
TruncatedSVD()
```

[16]:

```python
# Matrix decomposition
U = svd_model.transform(x)
S = svd_model.singular_values_
V = svd_model.components_

print('U,Sigma,V', U,S,V)
```

```
U,Sigma,V [[ 2.29510691e-01  6.45903788e-01]
 [ 8.63343685e-02  2.21356982e-01]
 [ 1.27120430e-01  3.23376559e-01]
 [ 1.22736128e-01  3.05693164e-01]
 [ 1.21188022e-01  2.85700307e-01]
 [ 2.25835629e-01  6.16782837e-01]
 [ 2.83365165e-01 -1.35568122e-01]
 [ 1.21107728e-01  4.89394022e-02]
 [ 1.10234292e-01  3.63416843e-02]
 [ 4.89508045e-02  9.38250437e-02]
 [ 1.32556822e-01  3.85568181e-01]
 [ 2.95555694e-01  5.22283023e-02]
 [ 2.59035442e-01  4.46283632e-02]
 [ 1.50903999e-01 -3.13723912e-02]
 [ 1.76662725e-01 -2.85797419e-02]
 [ 2.50968860e-01 -1.22269826e-02]
 [ 2.74044071e-01 -1.38801641e-01]
 [ 1.43859000e-01 -5.98953781e-02]
 [ 1.27891171e-01 -2.64041041e-02]
 [ 2.77783187e-01 -1.77581698e-01]
 [ 2.06160619e-01  1.78073160e-01]
 [ 2.64580371e-01  2.55106922e-01]
 [ 3.33582513e-01 -1.50627694e-01]
 [ 1.36526830e-01 -8.12118829e-02]
 [ 1.28430110e-01 -1.16322951e-01]
 [-3.80559021e-05 -3.74014075e-04]
```

```
[ 2.31385177e-01  5.09197084e-02]
[ 2.16049896e-01 -7.33684003e-02]
[ 1.52479100e-01 -9.33210087e-02]
[ 2.44270918e-02 -9.62695010e-03]
[ 5.53723830e-02  7.79461009e-02]
[ 4.26699207e-02  2.99366566e-03]
[ 2.62046133e-02 -9.22709242e-03]
[ 2.70226192e-01 -7.52437885e-02]
[ 6.26283356e-02 -2.30706248e-02]
[ 2.60313911e-01 -8.28319714e-02]
[ 1.72542469e-01 -8.77613373e-02]
[ 2.13814825e-01 -1.41993005e-01]
[ 3.58213642e-01 -1.08255566e-01]
[ 1.98966867e-01 -2.93899530e-02]
[ 2.60814426e-02  9.48027231e-03]
[ 1.45349576e-01 -7.70222897e-02]
[ 3.38884675e-01 -1.59391697e-01]
[ 1.56068010e-01 -6.13932581e-02]
[ 2.20680185e-01 -1.06723079e-01]
[ 1.25076623e-01  2.37859345e-01]
[ 1.51491425e-01 -5.44356767e-02]
[ 5.78480034e-02  1.25110462e-01]
[ 1.74098810e-01  2.77834859e-01]
[ 2.12555804e-01 -1.54301753e-01]
[ 7.04323674e-02  2.09047017e-02]
[ 2.93190600e-01 -7.12690887e-02]
[ 1.54208859e-01  4.03270580e-02]
[ 1.17968898e-01  1.76342608e-01]
[ 1.92761382e-01  1.68511942e-01]
[ 2.79846319e-01  6.28776657e-02]
[ 1.04321625e-01  1.02036635e-02]
[ 1.25905037e-01  1.68152552e-02]
[ 1.29787968e-01  2.66444815e-01]
[ 2.11927324e-01  1.75451984e-01]
[ 2.38744573e-01 -1.60903802e-01]
[ 2.84558629e-01 -1.66470945e-01]
[ 2.09847742e-01 -7.18962666e-03]
[ 2.76989552e-01 -1.30839771e-02]
[ 1.95408044e-01 -4.06705100e-02]
[ 1.80665739e-01  1.60225359e-01]
[ 3.41335938e-01 -6.67292602e-02]
[ 1.53461841e-01 -5.37798579e-02]
[ 1.97613712e-01 -6.44087080e-02]
[ 1.51732278e-01  3.41044141e-02]
[ 1.63569019e-01 -1.08503544e-01]
[ 7.06855246e-02 -7.37154638e-02]
[ 3.17734727e-01  1.04169261e-01]
[ 1.09582549e-01 -2.78943626e-02]
[ 2.21392429e-01 -1.95784902e-01]
[ 1.93287195e-01 -8.58815285e-02]
[ 1.71725345e-01 -7.72920246e-02]
[ 9.42214843e-02 -5.00048482e-02]
[ 1.58979904e-01 -9.41170192e-02]
[ 2.88655248e-01 -1.42459405e-01]
[ 2.33002746e-01 -7.91134706e-02]
[ 2.35609333e-01  4.78796109e-01]
[ 7.03647751e-02  1.67999288e-02]
[ 1.63651408e-01  2.23558832e-02]
[ 2.74917810e-01  9.94407628e-02]
[ 1.29424211e-01 -8.12391790e-02]
[ 2.16582282e-01 -1.38220583e-02]
[ 2.03967717e-01 -4.73626570e-02]
[ 3.02097030e-01 -1.66483112e-01]
[ 2.13540568e-01 -7.13459852e-02]
[ 1.69794098e-01  2.35957292e-02]
[ 1.01832054e-01  1.36478167e-01]
[ 2.08083814e-01 -6.87976924e-02]
[ 2.27800051e-01  1.89958474e-01]
[ 1.45471452e-01  1.25216016e-01]
[ 2.22496794e-01 -5.20892637e-02]
[ 1.19840399e-01 -3.82848670e-02]
[ 3.71723370e-02  4.26067373e-03]
[ 1.64782141e-02  8.11552178e-04]
[ 5.82412394e-02 -1.40736774e-02]
[ 1.09559816e-01  8.23205012e-02]
[ 4.58528820e-02 -1.79730521e-02]
[ 1.05262198e-01  6.16674079e-02]
[ 1.55154683e-01  3.14668294e-01]
[ 1.54716159e-01  4.45760453e-03]
[ 2.98413466e-01 -1.70701950e-01]
[ 2.66426389e-01 -5.45993196e-02]
[ 1.93169034e-01 -7.46790147e-02]
[ 2.36179177e-01 -3.53979198e-02]
[ 4.76724988e-02  1.95549703e-02]
[ 2.03004607e-01  2.67174580e-01]
[ 7.74259130e-02  1.87756275e-03]
[ 3.05056041e-01 -1.07948946e-01]
[ 3.77442242e-01  1.48154689e-01]
[ 1.74025404e-01  4.34766398e-02]
[ 1.21788699e-01 -2.67131243e-02]
[ 2.14641563e-01 -8.16307547e-02]
[ 3.27221454e-01 -1.83736793e-01]
[ 1.53757684e-01 -1.69818085e-02]
[ 1.26022227e-01  6.74180886e-02]
[ 3.25283406e-02 -3.33151306e-02]
[ 2.89770400e-01 -1.51262338e-01]
```

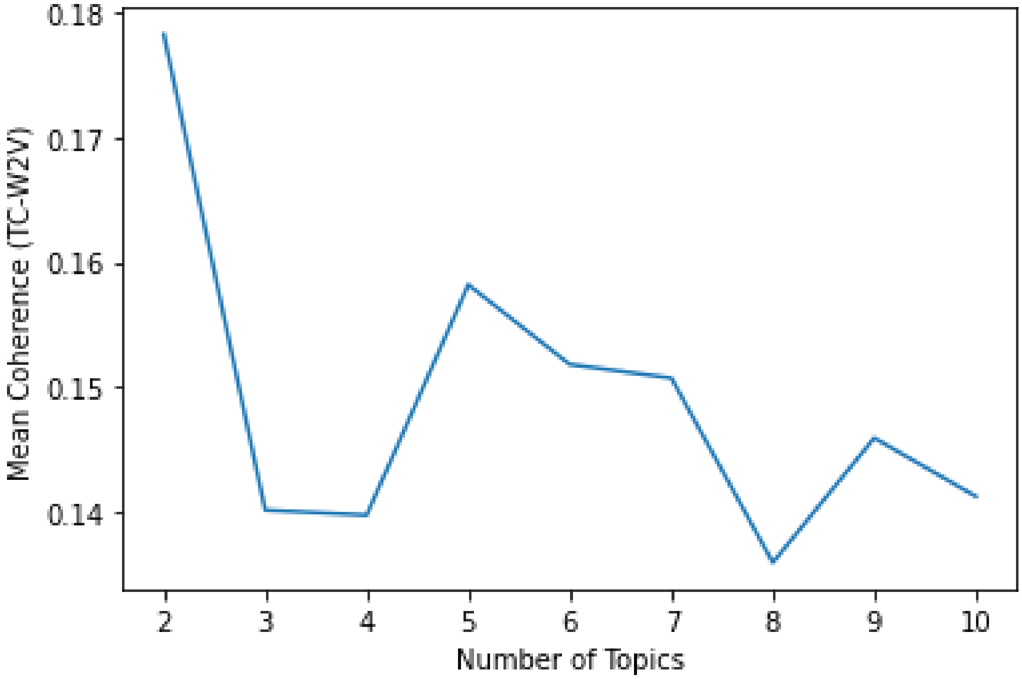
```
 [ 1.15417691e-01  2.78666105e-02]
 [ 1.87863343e-01 -3.01911897e-02]
 [ 1.13610894e-01  2.39404302e-01]
 [ 1.12826253e-01  7.11053088e-04]
 [ 1.73266037e-01  1.99904132e-01]
 [ 1.14731913e-01 -2.14234350e-02]
 [ 1.58978608e-01 -5.31398119e-02]
 [ 1.31112562e-01 -1.00686363e-01]
 [-6.24396096e-05 -1.70321159e-03]
 [ 2.86035335e-01 -1.26313401e-01]
 [ 2.68027695e-01 -1.37258362e-01]
 [ 1.77903587e-01 -7.38606229e-02]
 [ 1.48546278e-01  1.44631836e-01]
 [ 1.44362994e-01 -4.55547389e-02]
 [ 1.54155933e-02 -2.30003248e-02]
 [ 1.38521809e-01 -6.68348376e-02]
 [ 9.95746762e-02 -4.35255295e-02]
 [ 2.08715088e-01 -3.80893260e-02]
 [ 4.03103610e-01 -1.07704636e-01]
 [ 1.40513908e-01 -8.75837058e-02]] [2.313453   1.75572944] [[ 0.01258158
0.00277439  0.00151041 …  0.01661916  0.00585659
   0.00573   ]
 [ 0.01801006 -0.00139865  0.00041208 … -0.00577072  0.02984014
   0.00531527]]
```

[17]:
```
print("\nTopics in Best LSA model:")
n_top_words = 10
terms = vect.get_feature_names()
print_top_words(svd_model, terms, n_top_words)
```

```
Topics in Best LSA model:
Topic #0: health mental disorder problem treatment approach transdiagnostic
system research anxiety
Topic #1: disorder anxiety social generalised personality depressive major mood
example depression
```

0.5 Visualizations

[18]:
```
fig, ax = plt.subplots()
ax.plot(topic_count, list(hyperparameters_score_dict.values()))
plt.xlabel('Number of Topics')
plt.ylabel('Mean Coherence (TC-W2V)')
fig.show()
```



[25]:
```
# Save figure
fig.savefig("lsa_ntopics_tuning.svg")
```

latent_dirichlet_allocation

December 2, 2020

0.1 Imports

[3]:

```python
# Std imports
import operator
from itertools import combinations
# read data
import pandas as pd
# preprocessing
import nltk
from sklearn.feature_extraction.text import CountVectorizer
# LDA model
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.preprocessing import Normalizer
from gensim.models import KeyedVectors
# Visualization
import matplotlib.pyplot as plt
```

0.2 Read Data

[4]:

```python
# filename = "sleep.txt"
filename = "concussion.txt"
# filename = "mental_health.txt"

fp = open(filename, 'r')

data = fp.readlines()

fp.close()
```

[5]:

```python
# Converting paragraphs to sentences
sentences = []

for d in data:
    sentence_list = nltk.sent_tokenize(d)
    sentences.extend(sentence_list)
```

```
len(sentences)
```

[5]:
```
326
```

[6]:
```
dict = {'text': sentences}

df = pd.DataFrame(dict)
print(df)
```

```
                                                  text
0    It’s a late summer day in 1998 during pre-seas…
1    I’m 17 years old, a freshman fullback on the u…
2    After a lackluster start, I’m working to prove…
3    I’ve let nerves shake my confidence and I can …
4    We’re doing a drill and I’m standing with my b…
..                                                 …
321  But we are finally easing into each other, sus…
322  We share our secrets, and that impulse to want…
323  We know about the feverish devotion to this th…
324  We know how it turned a young girl without a l…
325  And that what she could never feel in church -…

[326 rows x 1 columns]
```

0.3 Preprocess Text

0.3.1 1. Remove punctuation (,/.!’?”)

0.3.2 2. Convert to lower case

0.3.3 3. Text tokenization

0.3.4 4. Remove stop words

0.3.5 5. Text leminization

0.3.6 6. Bag of words

[7]:
```
# remove punctuations
df['processed_text'] = df['text'].str.replace('[^\w\s]','')

print(df['processed_text'])
```

```
0      Its a late summer day in 1998 during preseason…
1      Im 17 years old a freshman fullback on the upa…
2      After a lackluster start Im working to prove m…
```

```
3      Ive let nerves shake my confidence and I can s…
4      Were doing a drill and Im standing with my bac…
                                            …
321    But we are finally easing into each other susp…
322    We share our secrets and that impulse to want …
323    We know about the feverish devotion to this th…
324    We know how it turned a young girl without a l…
325    And that what she could never feel in church  …
Name: processed_text, Length: 326, dtype: object
```

[8]:
```
# Remove stopwords
stop_words = set(nltk.corpus.stopwords.words('english'))

df['processed_text'] = df['processed_text'].apply(lambda x: ' '.join([word for␣
 ↪word in x.lower().split() if word not in stop_words]))
df.head()
```

[8]:
```
                                                text  \
0  It’s a late summer day in 1998 during pre-seas…
1  I’m 17 years old, a freshman fullback on the u…
2  After a lackluster start, I’m working to prove…
3  I’ve let nerves shake my confidence and I can …
4  We’re doing a drill and I’m standing with my b…

                                      processed_text
0  late summer day 1998 preseason training womens…
1  im 17 years old freshman fullback upandcoming …
2                   lackluster start im working prove
3  ive let nerves shake confidence see coach fell…
4                  drill im standing back lanky senior
```

[9]:
```
# from SKLearn docs
# class for tokenizing and lemmatizing
class LemmaTokenizer():
    def __init__(self):
        self.wnl = nltk.stem.WordNetLemmatizer()

    def __call__(self, doc):
        return [self.wnl.lemmatize(t) for t in nltk.word_tokenize(doc)]
```

[10]:
```
# this CountVectorizer will put our data in a bag of words w/ word counts
# provided the parameters below, it will also convert words to lower case,␣
 ↪tokenize text,
# lemmatize text, stem text, and remove stop words!
vect = CountVectorizer(analyzer='word',
                       tokenizer=LemmaTokenizer(),
                       lowercase='True')
```

```
x = vect.fit_transform(df['processed_text'])
print('After preprocessing, we have this many words: ' + str(len(vect.
 ↪get_feature_names())))
```

```
After preprocessing, we have this many words: 1660
```

0.4 Topic Modelling - LDA

0.4.1 Evaluation (Topic Coherence - TC-W2V)

[11]:

```
## Load pretrained Word2Vec model from file
# It is a pretrained Word2Vec model. It was trained on Google news dataset.␣
 ↪Since the input dataset is an article on # health issues and news usuallly␣
 ↪talk about them, the news model would be suitable for the dataset.
# To download the below model, please visit: https://drive.google.com/file/d/
 ↪0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
w2v_model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.
 ↪bin', binary=True)
```

[12]:

```
# http://derekgreene.com/slides/topic-modelling-with-scikitlearn.pdf
def get_mean_coherence(w2v_model, term_rankings):
    overall_coherence = 0.0

    for topic_index in range(len(term_rankings)):
        # check each pair of terms
        pair_scores = []

        for pair in combinations( term_rankings[topic_index], 2 ):
            try:
                pair_scores.append( w2v_model.similarity(pair[0], pair[1]) )
            except KeyError:
                continue

        # get the mean for all pairs in this topic
        topic_score = sum(pair_scores) / len(pair_scores)
        overall_coherence += topic_score

    # mean coherence across all topics
    mean_coherence = overall_coherence / len(term_rankings)

    return mean_coherence
```

0.4.2 Hyperparameters tuning

[14]:

```
# Returns the n_top_words words for each topic as a list of list
def get_term_rankings(vect, lda, n_top_words):
    term_rankings = []

    # Appending n_top_words of each topic to term_rankings
    terms = vect.get_feature_names()
    for topic_idx, topic in enumerate(lda.components_):
        top_terms = [terms[i] for i in topic.argsort()[:-n_top_words - 1:-1]]
        term_rankings.append(top_terms)

    return term_rankings
```

[15]:

```
# Function to print the top n_top_words for each topic
def print_top_words(model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(model.components_):
        message = "Topic #%d: " % topic_idx
        message += " ".join([feature_names[i]
                             for i in topic.argsort()[:-n_top_words - 1:-1]])
        print(message)
```

[17]:

```
# Grid Search to find best parameters for LDA model
alpha_values = [0.05, 0.1, 0.5]
beta_values = [0.05, 0.1, 0.5]
topic_count = [2, 3, 4, 5, 6, 7, 8, 9]
hyperparameters_score_dict = {}

for ntopic in topic_count:
    for a in alpha_values:
        for b in beta_values:
            print("Topics: ", ntopic, "Alpha: ", a, "Beta: ", b)
            lda = LatentDirichletAllocation(n_components=ntopic,␣
 ↪doc_topic_prior=a, topic_word_prior=b)
            lda.fit(x)
            feature_names = vect.get_feature_names()
            term_rankings = get_term_rankings(vect, lda, 10)
            mu_coherence = get_mean_coherence(w2v_model, term_rankings)
            hyperparameters_score_dict[str(ntopic)+","+str(a)+","+str(b)] =␣
 ↪round(mu_coherence, 4)
            print(mu_coherence)
```

```
Topics:  2 Alpha:  0.05 Beta:  0.05
0.1462694206016345
Topics:  2 Alpha:  0.05 Beta:  0.1
0.16708992365747688
Topics:  2 Alpha:  0.05 Beta:  0.5
```

```
0.1641011028136644
Topics:  2 Alpha:  0.1 Beta:  0.05
0.17111735503292747
Topics:  2 Alpha:  0.1 Beta:  0.1
0.16001946596014832
Topics:  2 Alpha:  0.1 Beta:  0.5
0.18773259822693134
Topics:  2 Alpha:  0.5 Beta:  0.05
0.18524607544143995
Topics:  2 Alpha:  0.5 Beta:  0.1
0.15832168165555532
Topics:  2 Alpha:  0.5 Beta:  0.5
0.1504070373588345
Topics:  3 Alpha:  0.05 Beta:  0.05
0.16294109364971518
Topics:  3 Alpha:  0.05 Beta:  0.1
0.16552134149328426
Topics:  3 Alpha:  0.05 Beta:  0.5
0.15193596217367386
Topics:  3 Alpha:  0.1 Beta:  0.05
0.1659587673980881
Topics:  3 Alpha:  0.1 Beta:  0.1
0.17218972604583813
Topics:  3 Alpha:  0.1 Beta:  0.5
0.13513551793854547
Topics:  3 Alpha:  0.5 Beta:  0.05
0.17294065593051966
Topics:  3 Alpha:  0.5 Beta:  0.1
0.15754295385451297
Topics:  3 Alpha:  0.5 Beta:  0.5
0.15793851160797878
Topics:  4 Alpha:  0.05 Beta:  0.05
0.1538407094606858
Topics:  4 Alpha:  0.05 Beta:  0.1
0.16251178031994237
Topics:  4 Alpha:  0.05 Beta:  0.5
0.16070944217758046
Topics:  4 Alpha:  0.1 Beta:  0.05
0.15256715956040556
Topics:  4 Alpha:  0.1 Beta:  0.1
0.13581847043759707
Topics:  4 Alpha:  0.1 Beta:  0.5
0.15012102948191264
Topics:  4 Alpha:  0.5 Beta:  0.05
0.15064807785125192
Topics:  4 Alpha:  0.5 Beta:  0.1
0.1453213319783875
Topics:  4 Alpha:  0.5 Beta:  0.5
```

```
0.1430017458055065
Topics:  5 Alpha:  0.05 Beta:  0.05
0.16229864372328545
Topics:  5 Alpha:  0.05 Beta:  0.1
0.15776190235176019
Topics:  5 Alpha:  0.05 Beta:  0.5
0.15108437028713523
Topics:  5 Alpha:  0.1 Beta:  0.05
0.15863579253459142
Topics:  5 Alpha:  0.1 Beta:  0.1
0.14236100727278325
Topics:  5 Alpha:  0.1 Beta:  0.5
0.1749464074211816
Topics:  5 Alpha:  0.5 Beta:  0.05
0.15550564417408572
Topics:  5 Alpha:  0.5 Beta:  0.1
0.18068532978701923
Topics:  5 Alpha:  0.5 Beta:  0.5
0.14089308870438902
Topics:  6 Alpha:  0.05 Beta:  0.05
0.16276551653820745
Topics:  6 Alpha:  0.05 Beta:  0.1
0.1682505056279263
Topics:  6 Alpha:  0.05 Beta:  0.5
0.16266164179819284
Topics:  6 Alpha:  0.1 Beta:  0.05
0.15534874855958924
Topics:  6 Alpha:  0.1 Beta:  0.1
0.16068483499536615
Topics:  6 Alpha:  0.1 Beta:  0.5
0.15002920301587977
Topics:  6 Alpha:  0.5 Beta:  0.05
0.12917468945605734
Topics:  6 Alpha:  0.5 Beta:  0.1
0.14970904387516418
Topics:  6 Alpha:  0.5 Beta:  0.5
0.13878546511426706
Topics:  7 Alpha:  0.05 Beta:  0.05
0.14421451038547925
Topics:  7 Alpha:  0.05 Beta:  0.1
0.14145461543692303
Topics:  7 Alpha:  0.05 Beta:  0.5
0.14456046389486077
Topics:  7 Alpha:  0.1 Beta:  0.05
0.15706315620447553
Topics:  7 Alpha:  0.1 Beta:  0.1
0.1499913015487327
Topics:  7 Alpha:  0.1 Beta:  0.5
```

```
0.13062711899050528
Topics:  7 Alpha:  0.5 Beta:  0.05
0.14888237087020934
Topics:  7 Alpha:  0.5 Beta:  0.1
0.1421647433524153
Topics:  7 Alpha:  0.5 Beta:  0.5
0.1300635011482333
Topics:  8 Alpha:  0.05 Beta:  0.05
0.16242547788699285
Topics:  8 Alpha:  0.05 Beta:  0.1
0.1266189483097858
Topics:  8 Alpha:  0.05 Beta:  0.5
0.1361045151517222
Topics:  8 Alpha:  0.1 Beta:  0.05
0.14177073851719293
Topics:  8 Alpha:  0.1 Beta:  0.1
0.14594325011259776
Topics:  8 Alpha:  0.1 Beta:  0.5
0.12796664955062118
Topics:  8 Alpha:  0.5 Beta:  0.05
0.13673215921153314
Topics:  8 Alpha:  0.5 Beta:  0.1
0.1418941968062427
Topics:  8 Alpha:  0.5 Beta:  0.5
0.15012419010957867
Topics:  9 Alpha:  0.05 Beta:  0.05
0.1464546021723683
Topics:  9 Alpha:  0.05 Beta:  0.1
0.13235003735600745
Topics:  9 Alpha:  0.05 Beta:  0.5
0.14908988197538198
Topics:  9 Alpha:  0.1 Beta:  0.05
0.1514416251262581
Topics:  9 Alpha:  0.1 Beta:  0.1
0.13268972109819266
Topics:  9 Alpha:  0.1 Beta:  0.5
0.14780841158231559
Topics:  9 Alpha:  0.5 Beta:  0.05
0.15381051076239804
Topics:  9 Alpha:  0.5 Beta:  0.1
0.15218484464969576
Topics:  9 Alpha:  0.5 Beta:  0.5
0.1390041409799504
```

[18]:

```
# Retrieve best parameters
best_n_topic, best_alpha, best_beta = max(hyperparameters_score_dict.items(),␣
 ↪key=operator.itemgetter(1))[0].split(',')
```

```
best_n_topic = int(best_n_topic)
best_alpha = float(best_alpha)
best_beta = float(best_beta)

best_n_topic, best_alpha, best_beta
```

[18]:
```
(2, 0.1, 0.5)
```

[19]:
```
# LDA modeling - LDA with best parameters
lda = LatentDirichletAllocation(n_components=best_n_topic,␣
 ↪doc_topic_prior=best_alpha, topic_word_prior=best_beta)
lda.fit(x)
```

[19]:
```
LatentDirichletAllocation(doc_topic_prior=0.1, n_components=2,
                          topic_word_prior=0.5)
```

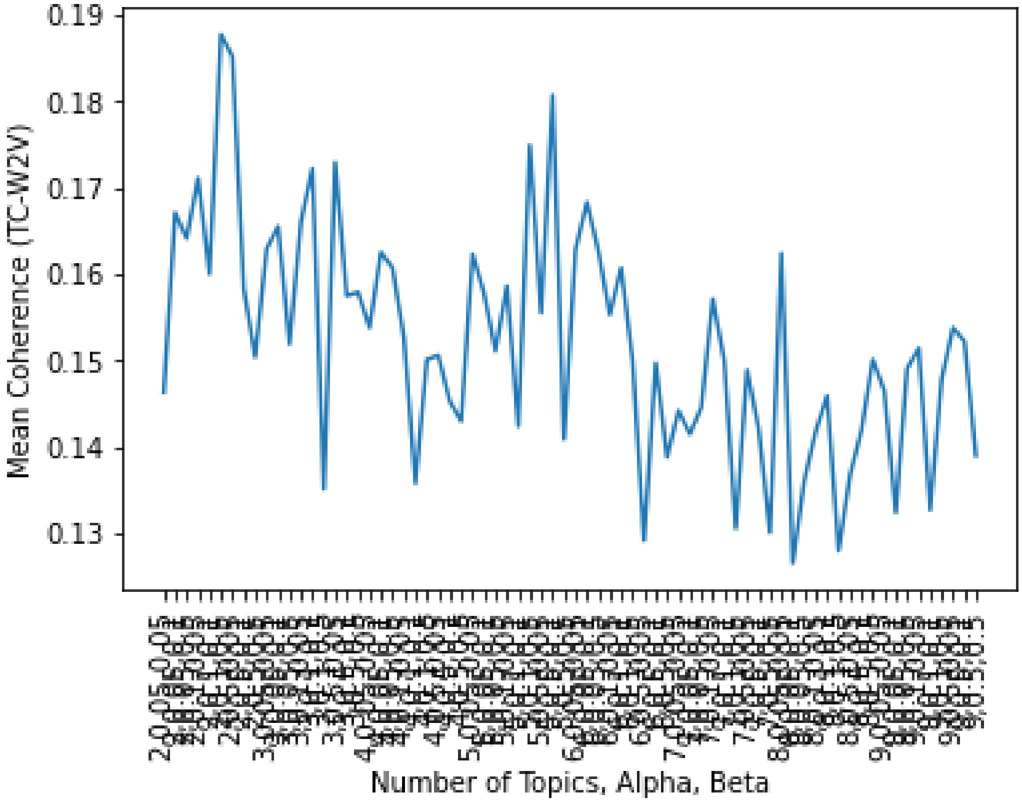
[20]:
```
print("\nTopics in Best LDA model:")
n_top_words = 10
feature_names = vect.get_feature_names()
print_top_words(lda, feature_names, n_top_words)
```

```

Topics in Best LDA model:
Topic #0: brain concussion injury year like team player id say played
Topic #1: concussion brain player athlete play time symptom ball field could
```

0.5 Visualizations

[21]:
```
fig, ax = plt.subplots()
ax.plot(list(hyperparameters_score_dict.keys()),␣
 ↪list(hyperparameters_score_dict.values()))
plt.xticks(rotation=90)
# fig.autofmt_xdate()
plt.xlabel('Number of Topics, Alpha, Beta')
plt.ylabel('Mean Coherence (TC-W2V)')
fig.show()
```



[53]:
```
fig.savefig("lda_ntopics_alpha_beta_tuning.svg")
```