# lenovo_topic_analysis_reviews_src

November 23, 2021

[ ]:
```
import re
import pandas as pd
import nltk
import gensim
import matplotlib.pyplot as plt
%matplotlib inline
from wordcloud import WordCloud
```

[ ]:
```
!pip install wordcloud
```

[ ]:
```
replace_vals = [(re.compile(r'@\w+'), ''),
                          (re.compile(r'http\S+'), '')]
```

[ ]:
```
# common functions
```

[ ]:
```
def read_data_set(data_filename):
    '''function to read dataset and print some information about the dataset'''
    # read csv file into dataframe
    data_df = pd.read_csv(
        data_filename, delimiter=",", encoding="utf-8")

    # print the info of twitter data frame
    print(data_df.head())
    print(data_df.shape)
    print(data_df.columns)
    print(data_df.isnull().sum())

    return data_df
```

[ ]:
```
def pre_token_cleanup(text, replace_vals):
    '''function to pre-process the tweets'''
    text = text.lower()  # convert to lower case

    # text = replace_with(text, [('&amp;', 'and'), ('&gt;', '>'), ('&lt;',␣
↪'<')])
    for replace in replace_vals:
        text = re.sub(replace[0], replace[1], text)
```

```
    text = text.strip()  # remove leading and trailing whitespace

    return text
```

[ ]:
```
def clean_reviews(review_df):
    '''function to clean the reviews'''
    review_df["review"] = review_df["review"].apply(pre_token_cleanup,␣
 ↪args=(replace_vals,))
    print(review_df.head())


    return review_df
```

[ ]:
```
def view_wordcloud_common_words(review_df):
    '''function to view the common words'''
    # get the common words
    all_words = ','.join(list(review_df['review'].values))
    # print(all_words)

    # view the word cloud
    w_cloud = WordCloud(background_color='white', max_words=5000, width=600,␣
 ↪height=300, contour_width=3, contour_color='steelblue')
    w_cloud.generate(all_words)

    w_cloud.to_file('wordcloud.png')
```

[ ]:
```
def get_values_for_topic_analysis(review_df):
    review_values = review_df["review"].values

    return review_values
```

[ ]:
```
def get_word_tokens(review_values):
    '''function to get the word tokens'''
    word_tokens = []
    for review in review_values:
        word_tokens.append(nltk.word_tokenize(review))

    return word_tokens
```

[ ]:
```
# read the data from csv file into dataframe
review_df = read_data_set('../data/k8_review.csv')
```

[ ]:
```
# Normalize casings and clean up the tweets
review_df = clean_reviews(review_df)

# extract the review text values into a list for easier manipulation.
review_values = get_values_for_topic_analysis(review_df)
```

```
print(review_values[:10])
```

[ ]:
```
view_wordcloud_common_words(review_df)
```

[ ]:
```
# tokenize the reviews using NLTK
review_word_tokens = get_word_tokens(review_values)
print(review_word_tokens[:10])
```

[ ]:
```
# using NLTK to get the parts of speech of the sentences
review_sentences_postags = [nltk.pos_tag(sentence) for sentence in␣
 ↪review_word_tokens]
print(review_sentences_postags[:2])
```

[ ]:
```
def get_postags_with_nouns(review_sentences_postags):
    '''function to get the pos tags with nouns'''
    noun_tags = ['NN', 'NNS', 'NNP', 'NNPS']
    postags_with_nouns = []
    for sentence in review_sentences_postags:
        postags_with_nouns.append([word for word, tag in sentence if tag in␣
 ↪noun_tags])

    return postags_with_nouns
```

[ ]:
```
# get the pos tags with nouns
postags_with_nouns = get_postags_with_nouns(review_sentences_postags)
print(postags_with_nouns[:10])
```

[ ]:
```
def get_postags_with_nouns_lemmed(postags_with_nouns):
    '''function to get the pos tags with nouns lemmatized'''
    lemmatizer = nltk.stem.WordNetLemmatizer()
    postags_with_nouns_lemmed = []
    for sentence in postags_with_nouns:
        postags_with_nouns_lemmed.append([lemmatizer.lemmatize(word) for word␣
 ↪in sentence])

    return postags_with_nouns_lemmed
```

[ ]:
```
# lemmatize the nouns
postags_with_nouns_lemmed = get_postags_with_nouns_lemmed(postags_with_nouns)
print(postags_with_nouns_lemmed[:10])
```

[ ]:
```
def remove_stop_words_and_puncs(postags_with_nouns_lemmed):
    '''function to remove stop words and punctuations'''
    stop_words = set(nltk.corpus.stopwords.words('english'))
    postags_with_nouns_lemmed_no_stop_words = []
    for sentence in postags_with_nouns_lemmed:
```

```
        postags_with_nouns_lemmed_no_stop_words.append([word for word in␣
↪sentence if word not in stop_words])

    return postags_with_nouns_lemmed_no_stop_words
```

[ ]:
```
# Remove stopwords and punctuation (if there are any).
postags_with_nouns_lemmed_no_stop_words =␣
↪remove_stop_words_and_puncs(postags_with_nouns_lemmed)
print(postags_with_nouns_lemmed_no_stop_words[:10])
```

[ ]:
```
def get_top_terms_for_topics_using_lda(postags_with_nouns_lemmed_no_stop_words,␣
↪num_topics, alpha, passes, workers):
    '''function to get the top terms for topics using LDA'''
    # Create a dictionary representation of the documents.
    dictionary = gensim.corpora.
↪Dictionary(postags_with_nouns_lemmed_no_stop_words)
    # Create a corpus from the bag of words.
    corpus = [dictionary.doc2bow(sentence) for sentence in␣
↪postags_with_nouns_lemmed_no_stop_words]
    # Build the LDA model.
    lda_model = gensim.models.LdaMulticore(corpus, num_topics=num_topics,␣
↪id2word=dictionary, passes=passes, alpha=alpha, random_state=426,␣
↪workers=workers)

    return lda_model, dictionary
```

[ ]:
```
num_topics=12
lda_model, dictionary =␣
↪get_top_terms_for_topics_using_lda(postags_with_nouns_lemmed_no_stop_words,␣
↪num_topics=num_topics, passes=20,alpha='symmetric', workers=3)

print(lda_model.print_topics(num_topics=num_topics, num_words=10))
```

[ ]:
```
def get_coherence_score_using_lda(lda_model, dictionary,␣
↪postags_with_nouns_lemmed_no_stop_words):
    '''function to get the coherence score using LDA'''

    # Compute Coherence Score
    coherence_model_lda = gensim.models.CoherenceModel(model=lda_model,␣
↪texts=postags_with_nouns_lemmed_no_stop_words, dictionary=dictionary,␣
↪coherence='c_v')
    coherence_lda = coherence_model_lda.get_coherence()

    return coherence_lda
```

```
[ ]: coherence_lda = get_coherence_score_using_lda(lda_model, dictionary,␣
     ↪postags_with_nouns_lemmed_no_stop_words)
     print('Coherence score: ',coherence_lda)
```

```
[ ]: def␣
     ↪get_coherence_score_for_multiple_topics(postags_with_nouns_lemmed_no_stop_words,␣
     ↪num_topics_list):
         '''function to get the coherence score for multiple topics'''
         coherence_scores = []
         for num_topics in num_topics_list:

             lda_model, dictionary =␣
     ↪get_top_terms_for_topics_using_lda(postags_with_nouns_lemmed_no_stop_words,␣
     ↪num_topics=num_topics, passes=30,alpha='symmetric', workers=5)
             coherence_model_lda = get_coherence_score_using_lda(lda_model,␣
     ↪dictionary, postags_with_nouns_lemmed_no_stop_words, num_topics)
             coherence_scores.append(coherence_model_lda)

         return coherence_scores
```

```
[ ]: num_topics_list = [5,6,7,8,9,10]
     coherence_scores =␣
     ↪get_coherence_score_for_multiple_topics(postags_with_nouns_lemmed_no_stop_words,␣
     ↪num_topics_list)
     print(coherence_scores)
```

```
[ ]: # get the model for better coherence score
     num_topics_for_better_coherence = num_topics_list[coherence_scores.
     ↪index(max(coherence_scores))]
     print('Number of topics for better coherence score:␣
     ↪',num_topics_for_better_coherence)
     lda_model_v1, dictionary_v1 =␣
     ↪get_top_terms_for_topics_using_lda(postags_with_nouns_lemmed_no_stop_words,␣
     ↪num_topics=num_topics_for_better_coherence, passes=30,alpha='symmetric',␣
     ↪workers=3)

     better_coherence_model_lda = get_coherence_score_using_lda(lda_model_v1,␣
     ↪dictionary_v1, postags_with_nouns_lemmed_no_stop_words,␣
     ↪num_topics_for_better_coherence)
     print('Better coherence model: ',better_coherence_model_lda)
```

```
[ ]: def print_topics_report(final_lda_model):
         topic_words = {}
         for idx, topic in final_lda_model.print_topics(-1):
             temp = []
             for item in topic.split('+'):
                 item_alpha = [letter for letter in item if letter.isalpha()]
```

```
            temp.append(''.join(item_alpha))
        topic_words[('Topic_'+str(idx+1))] = temp

    topics_df = pd.DataFrame(topic_words)
    topics_df.index = ['Word_'+str(i+1) for i in range(topics_df.shape[0])]
    print(topics_df)
```

[ ]:
```
print_topics_report(lda_model_v1)
```

[ ]:
```

```