

```

1 # modules.text_features.py
2 # https://github.com/QuantCS109/TrumpTweets/blob/master/modules/text\_features.py
3 # This is a .py file used in the notebooks
4
5 from sklearn.feature_extraction.text import CountVectorizer
6 from sklearn.feature_extraction.text import TfidfTransformer
7 from sklearn.decomposition import TruncatedSVD
8
9 import numpy as np
10 from scipy import sparse
11 import os
12
13 import pandas as pd
14
15
16 class TextFeaturesGenerator:
17
18     def __init__(self, text_series=None, score_series=None):
19         """
20         :param text_series: A pandas series with the text
21         """
22         self.text_series = text_series
23         self.count_vectorizer = None
24         self.tfidf_vectorizer = None
25
26         self.bow_mat = None
27         self.tfidf_mat = None
28
29         self.bow_mat_scored = None
30         self.tfidf_mat_scored = None
31
32         self.svd_bow_mat = None
33         self.svd_tfidf_mat = None
34         self.svd_bow_mat_scored = None
35         self.svd_tfidf_mat_scored = None
36
37         self.score_series = score_series
38
39     def get_bow_matrix(self):
40         """
41         Returns:
42             bow_matrix: A CSR (Compressed Sparse Row Matrix) of bag-of-words
43                        representation
44                        of the matrix
45         """
46         if self.bow_mat is None:
47             self.count_vectorizer = CountVectorizer()
48             self.count_vectorizer = self.count_vectorizer.fit(self.text_series)
49             self.bow_mat = self.count_vectorizer.transform(self.text_series)
50         if self.score_series is not None:
51             self.bow_mat_scored = self.count_vectorizer.transform(self.score_series)
52         return self.bow_mat, self.bow_mat_scored
53
54     def get_tfidf_matrix(self):
55         """
56         Returns:
57             bow_matrix: A CSR (Compressed Sparse Row Matrix) of tf-idf representation
58                        of the matrix
59         """
60         if self.tfidf_mat is None:
61             if self.bow_mat is None:
62                 _ = self.get_bow_matrix()
63             self.tfidf_vectorizer = TfidfTransformer(use_idf=True).fit(self.bow_mat)
64             self.tfidf_mat = self.tfidf_vectorizer.transform(self.bow_mat)
65         if self.score_series is not None:
66             self.tfidf_mat_scored = self.tfidf_vectorizer.transform(self.bow_mat_scored)

```

```

67         return self.tfidf_mat, self.tfidf_mat_scored
68     return self.tfidf_mat
69
70     def get_svd_bow_mat(self, n_components=2,
71                         algorithm='randomized',
72                         n_iter=5,
73                         random_state=None,
74                         tol=0.0):
75         if self.bow_mat is None:
76             _ = self.get_bow_matrix()
77         svd_transformer = TruncatedSVD(n_components, algorithm, n_iter,
78                                       random_state, tol).fit(self.bow_mat)
79         self.svd_bow_mat = svd_transformer.transform(self.bow_mat)
80         if self.score_series is not None:
81             self.svd_bow_mat_scored = svd_transformer.transform(self.bow_mat_scored)
82         return self.svd_bow_mat, self.svd_bow_mat_scored
83     return self.svd_bow_mat
84
85     def get_svd_tfidf_mat(self, n_components=2,
86                           algorithm='randomized',
87                           n_iter=5,
88                           random_state=None,
89                           tol=0.0):
90
91         if self.tfidf_mat is None:
92             _ = self.get_tfidf_matrix()
93         svd_transformer = TruncatedSVD(n_components, algorithm, n_iter,
94                                       random_state, tol).fit(self.tfidf_mat)
95         self.svd_tfidf_mat = svd_transformer.transform(self.tfidf_mat)
96         if self.score_series is not None:
97             self.svd_tfidf_mat_scored = svd_transformer.transform(self.tfidf_mat_scored)
98         return self.svd_tfidf_mat, self.svd_tfidf_mat_scored
99     return self.svd_tfidf_mat
100
101
102     def save_matrices(self, folder='../data/intermediate_data/', suffix=""):
103         """
104         Arguments:
105         :param folder: Folder / directory in which to save the matrices
106                       Will save in current working folder if not specified
107         """
108         if self.bow_mat is None:
109             _ = self.get_bow_matrix()
110         if self.tfidf_mat is None:
111             _ = self.get_tfidf_matrix()
112         if folder:
113             if not os.path.exists(folder):
114                 os.makedirs(folder)
115         bow_file = "tfidf_mat"+suffix+".npz"
116         tfidf_file = "bow_mat" + suffix + ".npz"
117         svd_bow_file = "svd tfidf_mat"+suffix+".npz"
118         svd_tfidf_file = "svd_bow_mat" + suffix + ".npz"
119
120         bow_location = os.path.join(folder, bow_file) if folder else bow_file
121         tfidf_location = os.path.join(folder, tfidf_file) if folder else tfidf_file
122         svd_bow_location = os.path.join(folder, svd_bow_file) if folder else svd_bow_file
123         svd_tfidf_location = os.path.join(folder, svd_tfidf_file) if folder else svd_tfidf_file
124
125         sparse.save_npz(bow_location, self.bow_mat)
126         sparse.save_npz(tfidf_location, self.tfidf_mat)
127         np.save(svd_bow_location, self.svd_bow_mat) if \
128             self.svd_bow_mat is not None else None
129         np.save(svd_tfidf_location, self.svd_tfidf_mat) if \
130             self.svd_tfidf_mat is not None else None
131
132

```

```

133
134 from nltk.sentiment.vader import SentimentIntensityAnalyzer
135
136 class SentimentFeaturesGenerator:
137     def __init__(self, tweet_df, aggregate=False):
138         self.text = tweet_df.tweets
139         self.tweets_df = tweet_df
140         self.sid = SentimentIntensityAnalyzer()
141         self.sentiment_df = pd.DataFrame()
142         self.sentiment_series = None
143         self.sentiment_df_aggregate = pd.DataFrame()
144         self.aggregate = aggregate
145
146     def get_sentiments(self):
147         self.sentiment_series = self.text.map(self.sid.polarity_scores)
148         self.sentiment_df['negative_proportion'] = self.sentiment_series.map(lambda x: x
149                                     .get('neg'))
150         self.sentiment_df['positive_proportion'] = self.sentiment_series.map(lambda x: x
151                                     .get('pos'))
152         self.sentiment_df['neutral_proportion'] = self.sentiment_series.map(lambda x: x
153                                     .get('neu'))
154         self.sentiment_df['combined_score'] = self.sentiment_series.map(lambda x: x.get(
155             'compound'))
156         self.sentiment_df['date'] = self.tweets_df.after4_date
157         self.sentiment_df.index = self.sentiment_series.index
158         self.sentiment_df.index = pd.to_datetime(self.sentiment_df.index)
159
160     def aggregate_sentiments(self):
161         self.sentiment_df_aggregate = self.sentiment_df.groupby('date').agg(['min', 'max'
162                                     , 'mean'])
163         self.sentiment_df_aggregate.columns = ["_".join([x[0], x[1]]) for x in\
164                                     self.sentiment_df_aggregate.columns]
165         self.sentiment_df_aggregate.index = pd.to_datetime(self.sentiment_df_aggregate.
166             index)
167
168     def run(self):
169         self.get_sentiments()
170         if self.aggregate:
171             self.aggregate_sentiments()
172
173
174

```