```python
# -*- coding: utf-8 -*-

import spacy
import math
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from spacytextblob.spacytextblob import SpacyTextBlob
from flair.models import TextClassifier
from flair.data import Sentence
from sentence_transformers import SentenceTransformer
from tqdm import tqdm, tqdm_pandas
from sklearn import metrics
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

#!python -m spacy download en_core_web_sm

DEBUG = False

def expandColumn(df, columnName, showProgress=False, progress=500):
    nlp = spacy.load('en_core_web_sm')
    totalRecords = len(df)
    for i, row in tqdm(df.iterrows(), desc='Expanding column: ' + columnName):
        if i % progress == 0 and showProgress:
            print(str(i) + " " + str("{:.1%}".format(i/totalRecords)) + " records processed for " + str(columnName))
        if (row[columnName] and len(str(row[columnName])) < 1000000):
            doc = nlp(str(row[columnName]))
            adjectives = []
            nouns = []
            verbs = []
```

```
34              lemmas = []
35
36              for token in doc:
37                  if not token.is_stop:
38                      lemmas.append(token.lemma_)
39                      if token.pos_ == "ADJ":
40                          adjectives.append(token.lemma_)
41                      if token.pos_ == "NOUN" or token.
   pos_ == "PROPN":
42                          nouns.append(token.lemma_)
43                      if token.pos_ == "VERB":
44                          verbs.append(token.lemma_)
45
46              df.at[i, columnName + "_lemma"] = " ".join(
   lemmas)
47              df.at[i, columnName + "_nouns"] = " ".join(
   nouns)
48              df.at[i, columnName + "_adjectives"] = " ".
   join(adjectives)
49              df.at[i, columnName + "_verbs"] = " ".join(
   verbs)
50              df.at[i, columnName + "_nav"] = " ".join(
   nouns + adjectives + verbs)
51
52 def calcTextBlobSentiment(df, columnName, showProgress=
   False, progress=500):
53      nlp = spacy.load('en_core_web_sm')
54      nlp.add_pipe('spacytextblob')
55
56      totalRecords = len(df)
57      for i, row in tqdm(df.iterrows(), desc='Calculating
    TextBlob Sentiment'):
58          if i % progress == 0 and showProgress:
59              print(str(i) + " " + str("{:.1%}".format(i
    / totalRecords)) + " records processed for " + str(
   columnName))
60          if (row[columnName] and len(str(row[columnName
   ])) < 1000000):
```

```
61              doc = nlp(str(row[columnName]))
62
63              df.at[i, columnName + "_tb_pol"] = doc._.
   polarity
64              df.at[i, columnName + "_tb_subj"] = doc._.
   subjectivity
65              df.at[i, columnName + "_tb_tokens"] = len(
   doc) #tokens including punctuation etc
66              df.at[i, columnName + "_tb_length"] = len(
   str(doc)) #length of text including spaces
67
68 def isNaN(num):
69     return num!= num
70
71 def binSpacyPolarity(polarity, numBins):
72   if isNaN(polarity):
73       return None
74
75   if polarity == -1:
76     return 1
77   else:
78     return math.ceil(((polarity + 1) / 2) * numBins)
79
80 def binPolarity(df, columnName, numBins=5):
81     tqdm.pandas()
82     tDf = df.copy()
83     tDf[columnName + '_norm'] = tDf.progress_apply(
84         lambda x: binSpacyPolarity(x[columnName],
   numBins=numBins), axis=1)
85     return tDf
86
87 def binPositiveNegative(val):
88     if isNaN(val):
89         return None
90
91     if val > 0:
92         return 1
93     else:
```

```
 94         return 0
 95
 96 def binPolarityPosNeg(df, columnName):
 97     tqdm.pandas()
 98     tDf = df.copy()
 99     tDf[columnName + '_posneg'] = tDf.progress_apply(
100         lambda x: binPositiveNegative(x[columnName]),
    axis=1)
101     return tDf
102
103 def splitSpacySentences(df, columnName, showProgress=
    False, progress=500):
104   nlp = spacy.load('en_core_web_sm')
105   nlp.add_pipe('spacytextblob')
106
107   split1=[]
108   split2=[]
109   split3=[]
110   split4=[]
111   split5=[]
112
113   totalRecords = len(df)
114   for i, row in tqdm(df.iterrows(), desc="Splitting
    sentences by polarity"):
115       #progress notification
116       if i % progress == 0 and showProgress:
117           print(str(i) + " " + str("{:.1%}".format(i/
    totalRecords)) + " records processed for " + str(
    columnName))
118
119       #is our sentence ok to process
120       if (row[columnName] and len(str(row[columnName
    ])) < 1000000):
121           doc = nlp(str(row[columnName]))
122           assert doc.has_annotation("SENT_START")
123
124       #process sentences in document
125       for sent in doc.sents:
```

```
126           sentDoc = nlp(str(sent.text))
127           #print(sent.text + ' (pol:' + str(sentDoc._.
    polarity) + ', subj:' + str(sentDoc._.subjectivity
    ) + ')')
128           polBin = binSpacyPolarity(sentDoc._.polarity
    , 5)
129           if polBin == 1:
130             split1.append(sent.text)
131           elif polBin == 2:
132             split2.append(sent.text)
133           elif polBin == 3:
134             split3.append(sent.text)
135           elif polBin == 4:
136             split4.append(sent.text)
137           elif polBin == 5:
138             split5.append(sent.text)
139           else:
140             print("Error: spacy sentence split found
    sentiment out of range")
141
142       df.at[i, columnName + "_tb_star1"] = " ".join(
    split1)
143       df.at[i, columnName + "_tb_star2"] = " ".join(
    split2)
144       df.at[i, columnName + "_tb_star3"] = " ".join(
    split3)
145       df.at[i, columnName + "_tb_star4"] = " ".join(
    split4)
146       df.at[i, columnName + "_tb_star5"] = " ".join(
    split5)
147
148 def calcFlairSentiment(doc, classifier):
149   if len(doc) == 0:
150     return
151
152   sentence = Sentence(doc)
153
154   classifier.predict(sentence)
```

```
155
156   value = sentence.labels[0].to_dict()['value']
157   if value == 'POSITIVE':
158       return sentence.to_dict()['labels'][0]['
    confidence']
159   else:
160       return -(sentence.to_dict()['labels'][0]['
    confidence'])
161
162
163 def flairSentimentEncode(df, columnName):
164     tqdm.pandas()
165     classifierName = 'en-sentiment'
166     print("Loading FLAIR text classifier: " +
    classifierName)
167     classifier = TextClassifier.load(classifierName)
168     print("FLAIR text classifier has been loaded")
169     print("Generating FLAIR sentiments")
170     df[columnName + '_flairSent'] = df.progress_apply(
    lambda x: calcFlairSentiment(x[columnName], classifier
    ), axis=1)
171     print("FLAIR sentiments completed")
172
173
174 def bertEncode(df, columnName):
175     tqdm.pandas()
176     bertType = 'bert-base-nli-max-tokens'
177     print("Loading BERT sentence transformer: " +
    bertType)
178     model_bert = SentenceTransformer(bertType)
179     print("BERT sentence transformer has been loaded")
180     print("Generating BERT encodings")
181     df[columnName + '_bert'] = df.progress_apply(
    lambda x: model_bert.encode(x[columnName]), axis=1)
182     print("BERT encodings completed")
183
184
185 def getBertEncodeFrame(df, bertColumn, uniqueColumn,
```

```
185 otherColumns=[], colPrefix='c'):
186     addCol = [uniqueColumn]
187     addCol = addCol + otherColumns
188 
189     numpy_data = np.array(df[bertColumn].to_list())
190     numpy_index = df[uniqueColumn].to_list()
191 
192     dfExp = pd.DataFrame(data=numpy_data, index=
    numpy_index)
193     dfExp.reset_index(inplace=True)
194     dfExp.rename(columns={'index': uniqueColumn},
    inplace=True)
195     for colname in dfExp.columns:
196         if colname != uniqueColumn:
197             dfExp.rename(columns={colname: colPrefix
     + str(colname)}, inplace=True)
198 
199     if len(otherColumns) > 0:
200 
201         dfOth = df[df.columns.intersection(addCol)]
202         dfRet = pd.merge(dfExp, dfOth, how='inner', on
    =uniqueColumn)
203         return dfRet
204     else:
205         return dfExp
206 
207 
208 def plotConfusionMatrix(conf_matrix, axis_labels,
    titleSuffix, cmap='mako',plotsize=5):
209     ax = sns.heatmap(conf_matrix,annot=True, fmt='d',
    cmap=cmap,xticklabels=axis_labels, yticklabels=
    axis_labels)
210 
211     if plotsize == 5:
212         sns.set(rc={'figure.figsize': (20, 8)})
213     elif plotsize == 4:
214         sns.set(rc={'figure.figsize': (15, 8)})
215     elif plotsize == 3:
```

```
216         sns.set(rc={'figure.figsize': (10, 8)})
217     elif plotsize == 2:
218         sns.set(rc={'figure.figsize': (8, 8)})
219     elif plotsize == 1:
220         sns.set(rc={'figure.figsize': (4, 8)})
221     else:  # Should be size 1
222         # should only be one but catch it and default
    to size 1
223         sns.set(rc={'figure.figsize': (4, 4)})
224 
225     plt.title(f'Confusion Matrix: {titleSuffix}',
    fontsize = 20) # title with fontsize 20
226     plt.xlabel('Predicted', fontsize = 15) # x-axis
    label with fontsize 15
227     plt.ylabel('Actual', fontsize = 15) # y-axis label
     with fontsize 15
228     plt.show()
229 
230 def showTestReport(df, colNameActual, colNamePredict,
    axisLabels, chartTitle):
231   results = metrics.classification_report(pd.
    to_numeric(df[colNameActual]).to_list(),
232                                           df[
    colNamePredict].to_list(),
233 
    zero_division=0)
234   print(results)
235 
236   cm = confusion_matrix(np.array(pd.to_numeric(df[
    colNameActual])).reshape(-1, 1),
237                         np.array(pd.to_numeric(df[
    colNamePredict])).reshape(-1, 1)
238                        )
239   plotConfusionMatrix(cm, axisLabels, chartTitle)
240 
241 
242 def createBertModel(df, bertColumn, uniqueColumn,
    targetColumn):
```

```
243     dfBert = getBertEncodeFrame(df=df,
244                                 bertColumn=bertColumn,
245                                 uniqueColumn=
    uniqueColumn,
246                                 otherColumns=[
    targetColumn]
247                                )
248     # Get X Value from dataframe
249     Y = np.array(dfBert[targetColumn])
250     dfBert.drop([uniqueColumn, targetColumn], axis=1,
    inplace=True)
251     X = dfBert.to_numpy()
252 
253     # split data into train and test sets
254     seed = 7
255     test_size = 0.33
256     X_train, X_test, y_train, y_test =
    train_test_split(X, Y, test_size=test_size,
    random_state=seed)
257     # fit model no training data
258     model = XGBClassifier()
259     model.fit(X_train, y_train)
260     # make predictions for test data
261     y_pred = model.predict(X_test)
262 
263     # make a dataframe for the results
264     tDf = pd.DataFrame(data=y_test, columns=["y_test"
    ])
265     tDf['y_pred'] = y_pred.tolist()
266 
267     return model, tDf
```