

```

1  # coding: utf-8
2  ### Original Notebook Created by CIEP / Global DDM COE
3  #### Nidhi Sawhney, Stojan Maleschlijski & Ian Henry
4
5  # In[16]:
6
7  import numpy as np # you probably don't need this line
8  from glob import glob
9  import os
10 import sys
11 import io
12 import pyhdb
13
14 from sklearn.metrics import precision_recall_fscore_support
15 from sklearn.metrics import f1_score
16
17
18 # In[27]:
19
20 from keras.preprocessing import sequence
21 from keras.utils import np_utils
22
23 from keras.models import Sequential
24 from keras.layers import Embedding
25 from keras.layers.core import Flatten,Dense,Dropout
26 from keras.optimizers import SGD,Adam
27 from keras.layers.convolutional import *
28
29 from keras.layers import SpatialDropout1D
30 from keras.layers import Merge,Input
31 from keras.models import Model
32
33 import base64
34 hanauid='My User'
35 hanapw='My Pass'
36
37
38 # In[29]:
39
40 from imblearn.over_sampling import SMOTE
41 from sklearn.metrics import confusion_matrix
42 import sklearn
43 print(sklearn.__version__)
44
45
46 # In[4]:
47
48 import re
49 from numpy.random import normal
50 import gensim.models.keyedvectors as Word2vec
51
52
53 # In[5]:
54
55 get_ipython().magic('matplotlib inline')
56
57
58 # In[6]:
59
60 def createBinaryModel(input_length, vocab_size) :
61     model = Sequential([
62         Embedding(vocab_size, 32, input_length=input_length,
63                 dropout=0.2),
64         SpatialDropout1D(0.2),
65         Dropout(0.25),
66         Convolution1D(64, 5, padding='same', activation='relu'),
67         Dropout(0.25),
68         MaxPooling1D(),
69         Flatten(),

```

```

70     Dense(100, activation='relu'),
71     Dropout(0.7),
72     Dense(1, activation='sigmoid')])
73     model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
74     model.summary()
75     return model
76
77
78 # In[7]:
79
80 def createMultiClassModel(num_labels,input_length , vocab_size) :
81     model = Sequential([
82         Embedding(vocab_size, 32, input_length=input_length,
83                 dropout=0.2),
84         SpatialDropout1D(0.2),
85         Dropout(0.25),
86         Convolution1D(64, 5, padding='same', activation='relu'),
87         Dropout(0.25),
88         MaxPooling1D(),
89         Flatten(),
90         Dense(100, activation='relu'),
91         Dropout(0.7),
92         Dense(num_labels, activation='softmax')])
93     model.compile(loss='categorical_crossentropy', optimizer=Adam(),
94                 metrics=['accuracy'])
95     model.summary()
96     return model
97
98 # In[8]:
99
100 def createMultiClassModelWithPTV(num_labels, input_length, vocab_size, emb) :
101     model = Sequential([
102         Embedding(vocab_size, len(emb[0]), input_length=input_length,
103                 weights=[emb], trainable=False),
104         SpatialDropout1D(0.2),
105         Dropout(0.25),
106         Convolution1D(64, 5, padding='same', activation='relu'),
107         Dropout(0.25),
108         MaxPooling1D(),
109         Flatten(),
110         Dense(100, activation='relu'),
111         Dropout(0.7),
112         Dense(num_labels, activation='softmax')])
113     model.compile(loss='categorical_crossentropy', optimizer=Adam(),
114                 metrics=['accuracy'])
115     model.summary()
116     return model
117
118 # In[26]:
119
120 cnxn = pyhdb.connect(
121     host="mo-3fdall1e5.mo.sap.corp",
122     port=30041,
123     user=hanauuid,
124     password=hanapw
125 )
126
127
128 # In[28]:
129
130 def getTrainingData() :
131
132     cursor = cnxn.cursor()
133     cursor.execute("CALL FAKENEWS.GET_REVIEW_DATA()")
134     data = cursor.fetchall()
135     data_array=np.asarray(data)
136     token_id = ((data_array[:,3]).astype(int))

```

```

137     data_ids = ((data_array[:,[0,3]]).astype(int))
138     file_names = (data_array[:,4])
139     return data_ids
140
141
142 # In[11]:
143
144 def formatTrainingData(raw_training_data, vocab_size, seq_len) :
145     document_ids = list(set(raw_training_data[:,0]))
146     xval = []
147     l=(raw_training_data[:,0])
148     for i in document_ids :
149         #sublist = list(data_ids[(get_indexes(i,l)),1])
150         sublist = (raw_training_data[np.where(l == i),1]).tolist()[0]
151         xval.append(sublist)
152
153     print(len(xval))
154     #Make rare words equal
155     val = [np.array([i if i<vocab_size-1 else vocab_size-1 for i in s]) for s in xval]
156     #Make matrix of same size by padding zeros or truncating
157     val = sequence.pad_sequences(val, maxlen=seq_len, value=0)
158
159     print(val.shape)
160     return val
161
162
163
164
165 # In[12]:
166
167 def getLabels() :
168
169     cursor = cnxn.cursor()
170
171     cursor.execute("SELECT FAKE FROM FAKENEWS.TITLE_140 ORDER BY ID")
172     labels = cursor.fetchall()
173     labels_array=np.asarray(labels)
174     labels = ((labels_array[:,0]).astype(int))
175
176     sub_labels = labels_array[np.where(labels_array > 0)]
177     return sub_labels
178
179
180 # In[13]:
181
182 def getBinaryLabels() :
183
184     cursor = cnxn.cursor()
185
186     cursor.execute("SELECT FAKE FROM FAKENEWS.TITLE_140 ORDER BY ID")
187     labels = cursor.fetchall()
188     labels_array=np.asarray(labels)
189     labels = ((labels_array[:,0]).astype(int))
190
191     return labels
192
193
194 # In[21]:
195
196 def getWords() :
197
198     cursor = cnxn.cursor()
199     cursor.execute("CALL FAKENEWS.GET_WORDS()")
200     data = cursor.fetchall()
201     words_array=np.asarray(data)
202     print(len(words_array))
203     return words_array
204
205

```

```

206 # In[15]:
207
208
209 def create_emb(words,wikimodel, vocab_size, n_fact):
210     #n_fact = model.shape[1]
211     #n_fact = 300
212     emb = np.zeros((vocab_size, n_fact))
213
214     for i in range(1,len(emb)):
215         word = words_array[i,1] #wikimodel.wv.index2word[i]
216         if word and re.match(r"^[a-zA-Z\-\_]*$", word):
217             #src_idx = wordidx[word]
218             #src_idx = wikimodel.vocab[word].index
219             #emb[i] = vecs[src_idx]
220             try:
221                 emb[i] = wikimodel.wv[word.lower()]
222
223             except:
224                 emb[i] = normal(scale=0.6, size=(n_fact,))
225         else:
226             # If we can't find the word in glove, randomly initialize
227             emb[i] = normal(scale=0.6, size=(n_fact,))
228
229     # This is our "rare word" id - we want to randomly initialize
230     emb[-1] = normal(scale=0.6, size=(n_fact,))
231     emb/=3
232     return emb
233
234
235 # In[16]:
236
237 def createCustomEmbedding(wordVector) :
238
239     wikimodel = Word2vec.KeyedVectors.load_word2vec_format(wordVector, binary=False)
240     emb=create_emb(getWords(), wikimodel,500,300)
241     return emb
242
243
244 # In[17]:
245
246 test = getBinaryLabels()
247 sum(test)
248
249
250 # In[ ]:
251
252
253
254

```