Twitter_NER

May 21, 2024

1 Imports

[2]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
from keras_preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.models import Model
from tensorflow.keras.layers import Input
# LSTM components
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout,␣
 ↪Bidirectional
# CRF layer
from tensorflow_addons.layers import CRF
# Sigmoid focal cross entropy loss. works well with highly unbalanced input data
from tensorflow_addons.losses import SigmoidFocalCrossEntropy
from tensorflow_addons.optimizers import AdamW
from gensim.models import KeyedVectors
from tensorflow.keras.preprocessing.text import Tokenizer
```

2 Reading Data

[3]:
```
with open('wnut_16_train.txt.conll','r',encoding='utf-8') as f:
    raw_train = f.read()

with open('wnut_16_test.txt.conll','r',encoding='utf-8') as f:
    raw_test = f.read()
```

3 EDA

[4]:
```
# Tweets present in train and test
len(raw_train.split('\n\n')), len(raw_test.split('\n\n'))
```

[4]:
```
(4850, 1394)
```

[5]:
```
# get labels from raw_train
train_labels = []
train_words = []

for tweet in raw_train.split('\n\n'):
    for line in tweet.split('\n'):
        if line:
            train_labels.append(line.split('\t')[1])
            train_words.append(line.split('\t')[0])

print('# of unique words : ', len(set(train_words)))
print('\nFrequency of words: \n',pd.DataFrame(train_words)[0].value_counts().
 ↪head())
print('------------------------------'*2)
print('\n# of unique labels : ', len(set(train_labels)))
print('\nFrequency of labels : \n',pd.DataFrame(train_labels)[0].value_counts().
 ↪head())
```

```
# of unique words :  21530

Frequency of words:
 .      2166
:       2093
,       1803
the     1311
to      1141
Name: 0, dtype: int64
------------------------------------------------------------

# of unique labels :  21

Frequency of labels :
 O             74245
B-geo-loc      1011
I-other         713
B-company       699
B-other         692
Name: 0, dtype: int64
```

[6]:
```
# get labels from raw_test
test_labels = []
test_words = []

for tweet in raw_test.split('\n\n'):
    for line in tweet.split('\n'):
        if line:
            test_labels.append(line.split('\t')[1])
            test_words.append(line.split('\t')[0])

# get unique labels
print('# of unique words : ', len(set(test_words)))
print('\nFrequency of words : \n',pd.DataFrame(test_words)[0].value_counts().
 ↪head())
print('-----------------------------'*2)
print('\n# of unique labels : ', len(set(test_labels)))
print('\nFrequency of labels : \n',pd.DataFrame(test_labels)[0].value_counts().
 ↪head())
```

```
# of unique words :  7203

Frequency of words :
 .      917
,      519
the    510
I      468
to     466
Name: 0, dtype: int64
------------------------------------------------------------

# of unique labels :  21

Frequency of labels :
 O            25715
B-person       263
I-other        163
B-geo-loc      147
I-person       118
Name: 0, dtype: int64
```

4 Build Sentences

[24]:
```
def build_sentences(raw):
    sentences = []
    for tweet in raw.split('\n\n'):
        sentence = []
        for line in tweet.split('\n'):
            if line:
                sentence.append(line.split('\t'))
        sentences.append(sentence)
    return sentences
```

[25]:
```
train_sentences = build_sentences(raw_train)
test_sentences = build_sentences(raw_test)
```

[26]:
```
len(train_sentences), len(test_sentences)
```

[26]: (4850, 1394)

[27]:
```
print('Avg length of train sentences :',np.round(np.mean([len(s) for s in␣
 ↪train_sentences])))
print('Max length of train sentences :',np.round(np.max([len(s) for s in␣
 ↪train_sentences])))
print('---')
print('Avg length of test sentences :',np.round(np.mean([len(s) for s in␣
 ↪test_sentences])))
print('Max length of test sentences :',np.round(np.max([len(s) for s in␣
 ↪test_sentences])))
```

```
Avg length of train sentences : 17.0
Max length of train sentences : 37
---
Avg length of test sentences : 19.0
Max length of test sentences : 39
```

[28]:
```
n_words = len(set(train_words))
n_tags = len(set(train_labels))
words = list(set(train_words))
tags = list(set(train_labels))
n_words, n_tags
```

[28]: (21530, 21)

[29]:
```
# Vocabulary Key:word -> Value:token_index
# The first 2 entries are reserved for PAD and UNK
word2idx = {w: i + 2 for i, w in enumerate(words)}
word2idx["PAD"] = 0 # Padding
```

```
word2idx["UNK"] = 1 # Unknown words

# Vocabulary Key:token_index -> Value:word
idx2word = {i: w for w, i in word2idx.items()}

# Vocabulary Key:Label/Tag -> Value:tag_index
# The first entry is reserved for PAD
tag2idx = {t: i+1 for i, t in enumerate(tags)}
tag2idx["PAD"] = 0

# Vocabulary Key:tag_index -> Value:Label/Tag
idx2tag = {i: w for w, i in tag2idx.items()}
```

[30]:
```
print("The word Obama is identified by the index: {}".format(word2idx["Obama"]))
print("The labels B-person is identified by the index: {}".
 ↪format(tag2idx["B-person"]))
```

```
The word Obama is identified by the index: 15132
The labels B-person is identified by the index: 1
```

---

5 Preparing train test data

[31]:
```
MAX_LEN = 40
```

[32]:
```
X_train = [[word2idx.get(x[0], word2idx["UNK"]) for x in s] for s in␣
 ↪train_sentences]
X_train = pad_sequences(maxlen=MAX_LEN, sequences=X_train, padding="post",␣
 ↪value = word2idx["PAD"])
```

[33]:
```
X_test = [[word2idx.get(x[0], word2idx["UNK"]) for x in s] for s in␣
 ↪test_sentences]
X_test = pad_sequences(maxlen=MAX_LEN, sequences=X_test, padding="post", value␣
 ↪= word2idx["PAD"])
```

[34]:
```
y_train = [[tag2idx.get(x[1]) for x in s] for s in train_sentences]
y_train = pad_sequences(maxlen=MAX_LEN, sequences=y_train, padding="post",␣
 ↪value = tag2idx["PAD"])
y_train = np.array([to_categorical(s, num_classes=len(tag2idx)) for s in␣
 ↪y_train])
```

[35]:
```
y_test = [[tag2idx.get(x[1]) for x in s] for s in test_sentences]
y_test = pad_sequences(maxlen=MAX_LEN, sequences=y_test, padding="post", value␣
 ↪= tag2idx["PAD"])
```

```
y_test = np.array([to_categorical(s, num_classes=len(tag2idx)) for s in y_test])
```

[36]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[36]:
```
((4850, 40), (1394, 40), (4850, 40, 22), (1394, 40, 22))
```

6 Using Pretrained Embeddings

[37]:
```
pretrained_w2v = KeyedVectors.
 ↪load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
```

[39]:
```
pretrained_w2v['Obama'].shape
```

[39]:
```
(300,)
```

[44]:
```
print(f"Number of words we have embeddings for : {len([w for w in␣
 ↪set(train_words) if w in pretrained_w2v.key_to_index.keys()])} /␣
 ↪{len(set(train_words))}")
```

```
Number of words we have embeddings for : 11621 / 21530
```

[49]:
```
[w for w in set(train_words) if w not in pretrained_w2v.key_to_index.keys()][:
 ↪10]
```

[49]:
```
['http://tinyurl.com/26zeju5',
 'http://t.co/6yOqP1HYHV',
 'http://t.co/Zrf3iXpjec',
 'http://bit.ly/9GkyjU',
 '(@PaigeLouiseRyan',
 '#5-8',
 '@WesternToday',
 '01:03',
 '17:00',
 '@xXLauraJXx']
```

[53]:
```
embeddings_matrix = np.zeros((len(word2idx), 300))
```

[58]:
```
for k,v in word2idx.items():
    if k in pretrained_w2v.key_to_index.keys():
        embeddings_matrix[v] = pretrained_w2v[k]
```

7 Training

[72]:
```
# tf.random.set_seed(42)
tf.random.set_seed(84)
```

[84]:
```
def build_model(max_len = MAX_LEN, input_dim = len(word2idx), embedding_dim =␣
↪100):

  # Model definition
  input = Input(shape=(max_len,))

  # Get embeddings
  embeddings = Embedding(input_dim=input_dim,
                         output_dim=embedding_dim,
                         input_length=max_len,
                         mask_zero=True,
                         trainable=True,
                         weights=[embeddings_matrix])(input)

  # variational biLSTM
  output_sequences = Bidirectional(LSTM(units=50,␣
↪return_sequences=True))(embeddings)

  # Stacking
  output_sequences = Bidirectional(LSTM(units=50,␣
↪return_sequences=True))(output_sequences)

  # Adding more non-linearity
  dense_out = TimeDistributed(Dense(25, activation="relu"))(output_sequences)

  # CRF layer
  crf = CRF(len(tag2idx), name='crf')
  predicted_sequence, potentials, sequence_length, crf_kernel = crf(dense_out)

  model = Model(input, potentials)
  model.compile(
      optimizer=AdamW(weight_decay=0.001),
      loss= SigmoidFocalCrossEntropy(alpha=0.125)) # Sigmoid focal cross␣
↪entropy loss

  return model
```

[85]:
```
model = build_model(embedding_dim=300)

# Checkpointing
save_model = tf.keras.callbacks.ModelCheckpoint(filepath='twitter_ner_crf.h5',
```

```
  monitor='val_loss',
  save_weights_only=True,
  save_best_only=True,
  verbose=1
)

# Early stopping
es = tf.keras.callbacks.EarlyStopping(monitor='val_loss', verbose=1,␣
 ↪patience=10)

callbacks = [save_model, es]

model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 40)]              0

 embedding_1 (Embedding)     (None, 40, 300)           6459600

 bidirectional_2 (Bidirectio  (None, 40, 100)          140400
 nal)

 bidirectional_3 (Bidirectio  (None, 40, 100)          60400
 nal)

 time_distributed_1 (TimeDis  (None, 40, 25)           2525
 tributed)

 crf (CRF)                   [(None, 40),              1100
                              (None, 40, 22),
                              (None,),
                              (22, 22)]

=================================================================
Total params: 6,664,025
Trainable params: 6,664,025
Non-trainable params: 0
_________________________________________________________________
```

[86]:
```
model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 10,␣
 ↪shuffle = True, callbacks=[callbacks])
```

```
Epoch 1/10
WARNING:tensorflow:Gradients do not exist for variables ['chain_kernel:0'] when
```

```
minimizing the loss. If you're using `model.compile()`, did you forget to
provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['chain_kernel:0'] when
minimizing the loss. If you're using `model.compile()`, did you forget to
provide a `loss` argument?
152/152 [==============================] - ETA: 0s - loss: 0.1999
Epoch 1: val_loss improved from inf to 0.09670, saving model to
twitter_ner_crf.h5
152/152 [==============================] - 66s 289ms/step - loss: 0.1999 -
val_loss: 0.0967
Epoch 2/10
152/152 [==============================] - ETA: 0s - loss: 0.0762
Epoch 2: val_loss improved from 0.09670 to 0.05196, saving model to
twitter_ner_crf.h5
152/152 [==============================] - 34s 222ms/step - loss: 0.0762 -
val_loss: 0.0520
Epoch 3/10
152/152 [==============================] - ETA: 0s - loss: 0.0392
Epoch 3: val_loss improved from 0.05196 to 0.04179, saving model to
twitter_ner_crf.h5
152/152 [==============================] - 34s 220ms/step - loss: 0.0392 -
val_loss: 0.0418
Epoch 4/10
152/152 [==============================] - ETA: 0s - loss: 0.0234
Epoch 4: val_loss did not improve from 0.04179
152/152 [==============================] - 34s 225ms/step - loss: 0.0234 -
val_loss: 0.0550
Epoch 5/10
152/152 [==============================] - ETA: 0s - loss: 0.0171
Epoch 5: val_loss did not improve from 0.04179
152/152 [==============================] - 33s 218ms/step - loss: 0.0171 -
val_loss: 0.0668
Epoch 6/10
152/152 [==============================] - ETA: 0s - loss: 0.0140
Epoch 6: val_loss did not improve from 0.04179
152/152 [==============================] - 33s 218ms/step - loss: 0.0140 -
val_loss: 0.0792
Epoch 7/10
152/152 [==============================] - ETA: 0s - loss: 0.0120
Epoch 7: val_loss did not improve from 0.04179
152/152 [==============================] - 34s 222ms/step - loss: 0.0120 -
val_loss: 0.1244
Epoch 8/10
152/152 [==============================] - ETA: 0s - loss: 0.0104
Epoch 8: val_loss did not improve from 0.04179
152/152 [==============================] - 35s 233ms/step - loss: 0.0104 -
val_loss: 0.1103
Epoch 9/10
```

```
152/152 [==============================] - ETA: 0s - loss: 0.0091
Epoch 9: val_loss did not improve from 0.04179
152/152 [==============================] - 33s 214ms/step - loss: 0.0091 -
val_loss: 0.1615
Epoch 10/10
152/152 [==============================] - ETA: 0s - loss: 0.0097
Epoch 10: val_loss did not improve from 0.04179
152/152 [==============================] - 31s 207ms/step - loss: 0.0097 -
val_loss: 0.1447
```

[86]: <keras.callbacks.History at 0x1bbe901cfd0>

8 Predictions

[87]:
```
# %load_ext tensorboard
# %tensorboard --logdir logs
```

[88]:
```
X_test_predictions = model.predict(X_test)
```

```
44/44 [==============================] - 10s 35ms/step
```

[89]:
```
X_test_predictions.shape
```

[89]: (1394, 40, 22)

[90]:
```
X_test_predictions = np.argmax(X_test_predictions, axis = -1)
```

[91]:
```
X_test_predictions.shape
```

[91]: (1394, 40)

[92]:
```
X_test_predictions = [[idx2tag.get(x) for x in s] for s in X_test_predictions]
```

[93]:
```
performance_dict = dict(zip(tags, [[0,0],[0,0],[0,0],[0,0],[0,0],
                                   [0,0],[0,0],[0,0],[0,0],[0,0],
                                   [0,0],[0,0],[0,0],[0,0],[0,0],
                                   [0,0],[0,0],[0,0],[0,0],[0,0],
                                   [0,0],[0,0],[0,0],[0,0],[0,0],[0,0]]))
for i in range(len(test_sentences)):
    for j in range(len(test_sentences[i])):
        performance_dict[test_sentences[i][j][1]][0] += 1 # actual tag count
        if (test_sentences[i][j][1] == X_test_predictions[i][j]):
            performance_dict[test_sentences[i][j][1]][1] += 1
performance_dict = pd.DataFrame(performance_dict).T.reset_index()
performance_dict.columns =␣
 ↪['tag','actual_tag_count','correctly_predicted_tag_count']
```

```
performance_dict['recall'] =␣
 ↪round(100*performance_dict['correctly_predicted_tag_count']/
 ↪performance_dict['actual_tag_count'],1)
print('Mean Recall : ',performance_dict.loc[:,'recall'].mean().round(1), '%')
print('Mean Recall without Others Tag: ',performance_dict.
 ↪loc[performance_dict['tag']!='O','recall'].mean().round(1), '%')
performance_dict.sort_values(by = 'recall', ascending = False)
```

```
Mean Recall :  14.9 %
Mean Recall without Others Tag:  11.4 %
```

[93]:

	tag	actual_tag_count	correctly_predicted_tag_count	recall
17	O	25715	21797	84.8
7	B-geo-loc	147	79	53.7
8	I-other	163	73	44.8
14	B-other	117	49	41.9
19	B-company	93	30	32.3
16	I-facility	60	15	25.0
0	B-person	263	32	12.2
2	I-person	118	13	11.0
12	B-facility	56	3	5.4
10	I-product	43	1	2.3
1	B-tvshow	22	0	0.0
11	B-movie	21	0	0.0
9	B-musicartist	32	0	0.0
13	I-movie	28	0	0.0
6	I-sportsteam	16	0	0.0
15	B-product	60	0	0.0
5	I-tvshow	23	0	0.0
4	I-geo-loc	22	0	0.0
18	B-sportsteam	34	0	0.0
3	I-musicartist	39	0	0.0
20	I-company	15	0	0.0

9 Logging

- Not enough training data present for all the tags. Hence, the model is not able to predict all the tags.

[ ]:
```
performance_dict = dict(zip(tags, [[0,0],[0,0],[0,0],[0,0],[0,0],
                                   [0,0],[0,0],[0,0],[0,0],[0,0],
                                   [0,0],[0,0],[0,0],[0,0],[0,0],
                                   [0,0],[0,0],[0,0],[0,0],[0,0],
                                   [0,0],[0,0],[0,0],[0,0],[0,0],[0,0]]))
for i in range(len(test_sentences)):
    for j in range(len(test_sentences[i])):
```

```
        performance_dict[test_sentences[i][j][1]][0] += 1 # actual tag count
        if (test_sentences[i][j][1] == X_test_predictions[i][j]):
            performance_dict[test_sentences[i][j][1]][1] += 1
performance_dict = pd.DataFrame(performance_dict).T.reset_index()
performance_dict.columns =␣
 ↪['tag','actual_tag_count','correctly_predicted_tag_count']
performance_dict['recall'] =␣
 ↪round(100*performance_dict['correctly_predicted_tag_count']/
 ↪performance_dict['actual_tag_count'],1)
print('Mean Recall : ',performance_dict.loc[:,'recall'].mean().round(1), '%')
print('Mean Recall without Others Tag: ',performance_dict.
 ↪loc[performance_dict['tag']!='O','recall'].mean().round(1), '%')
performance_dict.sort_values(by = 'recall', ascending = False)
```

```
Mean Recall :  14.9 %
Mean Recall without Others Tag:  11.4 %
```

	tag	actual_tag_count	correctly_predicted_tag_count	recall
17	O	25715	21797	84.8
7	B-geo-loc	147	79	53.7
8	I-other	163	73	44.8
14	B-other	117	49	41.9
19	B-company	93	30	32.3
16	I-facility	60	15	25.0
0	B-person	263	32	12.2
2	I-person	118	13	11.0
12	B-facility	56	3	5.4
10	I-product	43	1	2.3
1	B-tvshow	22	0	0.0
11	B-movie	21	0	0.0
9	B-musicartist	32	0	0.0
13	I-movie	28	0	0.0
6	I-sportsteam	16	0	0.0
15	B-product	60	0	0.0
5	I-tvshow	23	0	0.0
4	I-geo-loc	22	0	0.0
18	B-sportsteam	34	0	0.0
3	I-musicartist	39	0	0.0
20	I-company	15	0	0.0