06_sentiment_analysis_pretrained_embeddings

September 29, 2021

1 Sentiment analysis with pretrained word vectors

In Chapter 15, Word Embeddings, we discussed how to learn domain-specific word embeddings. Word2vec, and related learning algorithms, produce high-quality word vectors, but require large datasets. Hence, it is common that research groups share word vectors trained on large datasets, similar to the weights for pretrained deep learning models that we encountered in the section on transfer learning in the previous chapter.

We are now going to illustrate how to use pretrained Global Vectors for Word Representation (GloVe) provided by the Stanford NLP group with the IMDB review dataset.

[1]:
```
%matplotlib inline

from pathlib import Path

import numpy as np
import pandas as pd

from sklearn.metrics import roc_auc_score

import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GRU, Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
import tensorflow.keras.backend as K

import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
gpu_devices = tf.config.experimental.list_physical_devices('GPU')
if gpu_devices:
    print('Using GPU')
    tf.config.experimental.set_memory_growth(gpu_devices[0], True)
else:
    print('Using CPU')
```

```
Using CPU
```

[3]:
```
sns.set_style('whitegrid')
np.random.seed(42)
```

[4]:
```
results_path = Path('results', 'sentiment_imdb')
if not results_path.exists():
    results_path.mkdir(parents=True)
```

1.1 Load Reviews

We are going to load the IMDB dataset from the source for manual preprocessing.

Data source: Stanford IMDB Reviews Dataset

Dowload extract, and place the content in a newly created `data` folder so that your directory structure looks as follows:

```
19_recurrent_neural_nets
 |-data
     |-aclimdb
          |-train
              |-neg
              |-pos
              ...
          |-test
          |-imdb.vocab
```

[5]:
```
path = Path('data', 'aclImdb')
```

[6]:
```
files = path.glob('**/*.txt')
len(list(files))
```

[6]:
```
50003
```

[7]:
```
files = path.glob('*/**/*.txt')
outcomes = set()
data = []
for f in files:
    if f.stem.startswith(('urls_', 'imdbEr')):
        continue
    _, _, data_set, outcome = f.parent.as_posix().split('/')
    if outcome == 'unsup':
        continue
    data.append([data_set, int(outcome == 'pos'),
                 f.read_text(encoding='latin1')])
```

[8]:
```
data = pd.DataFrame(data, columns=['dataset', 'label', 'review'])
```

[9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   dataset  50000 non-null  object
 1   label    50000 non-null  int64 
 2   review   50000 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.1+ MB
```

[10]:
```
train_data = data.loc[data.dataset=='train', ['label', 'review']]
test_data = data.loc[data.dataset=='test', ['label', 'review']]
```

[11]:
```
train_data.label.value_counts()
```

[11]:
```
0    12500
1    12500
Name: label, dtype: int64
```

[12]:
```
test_data.label.value_counts()
```

[12]:
```
0    12500
1    12500
Name: label, dtype: int64
```

1.2 Prepare Data

1.2.1 Tokenizer

Keras provides a tokenizer that we use to convert the text documents to integer-encoded sequences, as shown here:

[13]:
```
num_words = 10000
t = Tokenizer(num_words=num_words,
              lower=True,
              oov_token=2)
t.fit_on_texts(train_data.review)
```

[14]:
```
vocab_size = len(t.word_index) + 1
vocab_size
```

[14]:
```
88586
```

[15]:
```
train_data_encoded = t.texts_to_sequences(train_data.review)
test_data_encoded = t.texts_to_sequences(test_data.review)
```

[16]:
```
max_length = 100
```

1.2.2 Pad Sequences

We also use the pad_sequences function to convert the list of lists (of unequal length) to stacked sets of padded and truncated arrays for both the train and test datasets:

[17]:
```
X_train_padded = pad_sequences(train_data_encoded,
                               maxlen=max_length,
                               padding='post',
                              truncating='post')
y_train = train_data['label']
X_train_padded.shape
```

[17]:
```
(25000, 100)
```

[18]:
```
X_test_padded = pad_sequences(test_data_encoded,
                               maxlen=max_length,
                               padding='post',
                              truncating='post')
y_test = test_data['label']
X_test_padded.shape
```

[18]:
```
(25000, 100)
```

1.3 Load Embeddings

Assuming we have downloaded and unzipped the GloVe data to the location indicated in the code, we now create a dictionary that maps GloVe tokens to 100-dimensional real-valued vectors, as follows:

[19]:
```
# load the whole embedding into memory
glove_path = Path('..', 'data', 'glove', 'glove.6B.100d.txt')
embeddings_index = dict()

for line in glove_path.open(encoding='latin1'):
    values = line.split()
    word = values[0]
    try:
        coefs = np.asarray(values[1:], dtype='float32')
    except:
        continue
    embeddings_index[word] = coefs
```

[20]:
```
print('Loaded {:,d} word vectors.'.format(len(embeddings_index)))
```

```
Loaded 399,883 word vectors.
```

There are around 340,000 word vectors that we use to create an embedding matrix that matches the vocabulary so that the RNN model can access embeddings by the token index:

[21]:
```
embedding_matrix = np.zeros((vocab_size, 100))
for word, i in t.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

[22]:
```
embedding_matrix.shape
```

[22]:
```
(88586, 100)
```

1.4 Define Model Architecture

The difference between this and the RNN setup in the previous example is that we are going to pass the embedding matrix to the embedding layer and set it to non-trainable, so that the weights remain fixed during training:

[23]:
```
embedding_size = 100
```

[24]:
```
rnn = Sequential([
    Embedding(input_dim=vocab_size,
              output_dim= embedding_size,
              input_length=max_length,
              weights=[embedding_matrix],
              trainable=False),
    GRU(units=32,  dropout=0.2, recurrent_dropout=0.2),
    Dense(1, activation='sigmoid')
])
rnn.summary()
```

```
WARNING:tensorflow:Layer gru will not use cuDNN kernel since it doesn't meet the
cuDNN kernel criteria. It will use generic GPU kernel as fallback when running
on GPU
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 100)          8858600
_________________________________________________________________
gru (GRU)                    (None, 32)                12864
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 8,871,497
Trainable params: 12,897
Non-trainable params: 8,858,600
_________________________________________________________________
```

[25]:
```
rnn.compile(loss='binary_crossentropy',
            optimizer='RMSProp',
            metrics=['accuracy',
                     tf.keras.metrics.AUC(name='AUC')])
```

[26]:
```
rnn_path = (results_path / 'lstm.pretrained.h5').as_posix()

checkpointer = ModelCheckpoint(filepath=rnn_path,
                               verbose=1,
                               monitor='val_AUC',
                               mode='max',
                               save_best_only=True)
```

[27]:
```
early_stopping = EarlyStopping(monitor='val_AUC',
                               patience=5,
                               mode='max',
                               restore_best_weights=True)
```

[28]:
```
training = rnn.fit(X_train_padded,
                   y_train,
                   batch_size=32,
                   epochs=100,
                   validation_data=(X_test_padded,
                                    y_test),
                   callbacks=[early_stopping,
                              checkpointer],
                   verbose=1)
```

```
Epoch 1/100
782/782 [==============================] - ETA: 0s - loss: 0.6505 - accuracy:
0.6087 - AUC: 0.6565
Epoch 00001: val_AUC improved from -inf to 0.80524, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 111s 141ms/step - loss: 0.6505 -
accuracy: 0.6087 - AUC: 0.6565 - val_loss: 0.7026 - val_accuracy: 0.6586 -
val_AUC: 0.8052
Epoch 2/100
782/782 [==============================] - ETA: 0s - loss: 0.5053 - accuracy:
0.7570 - AUC: 0.8316
Epoch 00002: val_AUC improved from 0.80524 to 0.86949, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 111s 143ms/step - loss: 0.5053 -
accuracy: 0.7570 - AUC: 0.8316 - val_loss: 0.4681 - val_accuracy: 0.7776 -
val_AUC: 0.8695
Epoch 3/100
782/782 [==============================] - ETA: 0s - loss: 0.4538 - accuracy:
0.7872 - AUC: 0.8679
```

```
Epoch 00003: val_AUC improved from 0.86949 to 0.88737, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 117s 149ms/step - loss: 0.4538 -
accuracy: 0.7872 - AUC: 0.8679 - val_loss: 0.4352 - val_accuracy: 0.7931 -
val_AUC: 0.8874
Epoch 4/100
782/782 [==============================] - ETA: 0s - loss: 0.4303 - accuracy:
0.8000 - AUC: 0.8823
Epoch 00004: val_AUC improved from 0.88737 to 0.89225, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 109s 139ms/step - loss: 0.4303 -
accuracy: 0.8000 - AUC: 0.8823 - val_loss: 0.4404 - val_accuracy: 0.7947 -
val_AUC: 0.8922
Epoch 5/100
782/782 [==============================] - ETA: 0s - loss: 0.4138 - accuracy:
0.8062 - AUC: 0.8915
Epoch 00005: val_AUC improved from 0.89225 to 0.89876, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 109s 139ms/step - loss: 0.4138 -
accuracy: 0.8062 - AUC: 0.8915 - val_loss: 0.4284 - val_accuracy: 0.7994 -
val_AUC: 0.8988
Epoch 6/100
782/782 [==============================] - ETA: 0s - loss: 0.4064 - accuracy:
0.8123 - AUC: 0.8959
Epoch 00006: val_AUC improved from 0.89876 to 0.90227, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 109s 139ms/step - loss: 0.4064 -
accuracy: 0.8123 - AUC: 0.8959 - val_loss: 0.4015 - val_accuracy: 0.8176 -
val_AUC: 0.9023
Epoch 7/100
782/782 [==============================] - ETA: 0s - loss: 0.3959 - accuracy:
0.8201 - AUC: 0.9016
Epoch 00007: val_AUC improved from 0.90227 to 0.90520, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 108s 138ms/step - loss: 0.3959 -
accuracy: 0.8201 - AUC: 0.9016 - val_loss: 0.4228 - val_accuracy: 0.8072 -
val_AUC: 0.9052
Epoch 8/100
782/782 [==============================] - ETA: 0s - loss: 0.3884 - accuracy:
0.8230 - AUC: 0.9054
Epoch 00008: val_AUC improved from 0.90520 to 0.90554, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 118s 151ms/step - loss: 0.3884 -
accuracy: 0.8230 - AUC: 0.9054 - val_loss: 0.4152 - val_accuracy: 0.8064 -
val_AUC: 0.9055
Epoch 9/100
782/782 [==============================] - ETA: 0s - loss: 0.3814 - accuracy:
0.8264 - AUC: 0.9090
```

```
Epoch 00009: val_AUC improved from 0.90554 to 0.90921, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 107s 137ms/step - loss: 0.3814 -
accuracy: 0.8264 - AUC: 0.9090 - val_loss: 0.3939 - val_accuracy: 0.8167 -
val_AUC: 0.9092
Epoch 10/100
782/782 [==============================] - ETA: 0s - loss: 0.3766 - accuracy:
0.8275 - AUC: 0.9112
Epoch 00010: val_AUC improved from 0.90921 to 0.90976, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 108s 138ms/step - loss: 0.3766 -
accuracy: 0.8275 - AUC: 0.9112 - val_loss: 0.3896 - val_accuracy: 0.8194 -
val_AUC: 0.9098
Epoch 11/100
782/782 [==============================] - ETA: 0s - loss: 0.3698 - accuracy:
0.8348 - AUC: 0.9147
Epoch 00011: val_AUC improved from 0.90976 to 0.91103, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 142s 182ms/step - loss: 0.3698 -
accuracy: 0.8348 - AUC: 0.9147 - val_loss: 0.3808 - val_accuracy: 0.8267 -
val_AUC: 0.9110
Epoch 12/100
782/782 [==============================] - ETA: 0s - loss: 0.3644 - accuracy:
0.8345 - AUC: 0.9174
Epoch 00012: val_AUC improved from 0.91103 to 0.91111, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 106s 136ms/step - loss: 0.3644 -
accuracy: 0.8345 - AUC: 0.9174 - val_loss: 0.3912 - val_accuracy: 0.8201 -
val_AUC: 0.9111
Epoch 13/100
782/782 [==============================] - ETA: 0s - loss: 0.3600 - accuracy:
0.8369 - AUC: 0.9195
Epoch 00013: val_AUC improved from 0.91111 to 0.91143, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 111s 142ms/step - loss: 0.3600 -
accuracy: 0.8369 - AUC: 0.9195 - val_loss: 0.3763 - val_accuracy: 0.8279 -
val_AUC: 0.9114
Epoch 14/100
782/782 [==============================] - ETA: 0s - loss: 0.3553 - accuracy:
0.8398 - AUC: 0.9217
Epoch 00014: val_AUC improved from 0.91143 to 0.91288, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 108s 138ms/step - loss: 0.3553 -
accuracy: 0.8398 - AUC: 0.9217 - val_loss: 0.3769 - val_accuracy: 0.8277 -
val_AUC: 0.9129
Epoch 15/100
782/782 [==============================] - ETA: 0s - loss: 0.3505 - accuracy:
0.8433 - AUC: 0.9239
```

```
Epoch 00015: val_AUC improved from 0.91288 to 0.91290, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 110s 141ms/step - loss: 0.3505 -
accuracy: 0.8433 - AUC: 0.9239 - val_loss: 0.3991 - val_accuracy: 0.8195 -
val_AUC: 0.9129
Epoch 16/100
782/782 [==============================] - ETA: 0s - loss: 0.3488 - accuracy:
0.8429 - AUC: 0.9247
Epoch 00016: val_AUC did not improve from 0.91290
782/782 [==============================] - 110s 140ms/step - loss: 0.3488 -
accuracy: 0.8429 - AUC: 0.9247 - val_loss: 0.4081 - val_accuracy: 0.8132 -
val_AUC: 0.9118
Epoch 17/100
782/782 [==============================] - ETA: 0s - loss: 0.3450 - accuracy:
0.8452 - AUC: 0.9264
Epoch 00017: val_AUC improved from 0.91290 to 0.91368, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 110s 140ms/step - loss: 0.3450 -
accuracy: 0.8452 - AUC: 0.9264 - val_loss: 0.3795 - val_accuracy: 0.8306 -
val_AUC: 0.9137
Epoch 18/100
782/782 [==============================] - ETA: 0s - loss: 0.3430 - accuracy:
0.8460 - AUC: 0.9272
Epoch 00018: val_AUC did not improve from 0.91368
782/782 [==============================] - 108s 138ms/step - loss: 0.3430 -
accuracy: 0.8460 - AUC: 0.9272 - val_loss: 0.3891 - val_accuracy: 0.8221 -
val_AUC: 0.9127
Epoch 19/100
782/782 [==============================] - ETA: 0s - loss: 0.3376 - accuracy:
0.8477 - AUC: 0.9296
Epoch 00019: val_AUC did not improve from 0.91368
782/782 [==============================] - 106s 135ms/step - loss: 0.3376 -
accuracy: 0.8477 - AUC: 0.9296 - val_loss: 0.3822 - val_accuracy: 0.8267 -
val_AUC: 0.9134
Epoch 20/100
782/782 [==============================] - ETA: 0s - loss: 0.3368 - accuracy:
0.8516 - AUC: 0.9300
Epoch 00020: val_AUC improved from 0.91368 to 0.91385, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 105s 135ms/step - loss: 0.3368 -
accuracy: 0.8516 - AUC: 0.9300 - val_loss: 0.3994 - val_accuracy: 0.8138 -
val_AUC: 0.9139
Epoch 21/100
782/782 [==============================] - ETA: 0s - loss: 0.3340 - accuracy:
0.8532 - AUC: 0.9312
Epoch 00021: val_AUC did not improve from 0.91385
782/782 [==============================] - 105s 135ms/step - loss: 0.3340 -
accuracy: 0.8532 - AUC: 0.9312 - val_loss: 0.3741 - val_accuracy: 0.8297 -
```

```
val_AUC: 0.9131
Epoch 22/100
782/782 [==============================] - ETA: 0s - loss: 0.3310 - accuracy:
0.8509 - AUC: 0.9324
Epoch 00022: val_AUC improved from 0.91385 to 0.91487, saving model to
results/sentiment_imdb/lstm.pretrained.h5
782/782 [==============================] - 107s 137ms/step - loss: 0.3310 -
accuracy: 0.8509 - AUC: 0.9324 - val_loss: 0.3795 - val_accuracy: 0.8306 -
val_AUC: 0.9149
Epoch 23/100
782/782 [==============================] - ETA: 0s - loss: 0.3276 - accuracy:
0.8534 - AUC: 0.9339
Epoch 00023: val_AUC did not improve from 0.91487
782/782 [==============================] - 107s 137ms/step - loss: 0.3276 -
accuracy: 0.8534 - AUC: 0.9339 - val_loss: 0.3904 - val_accuracy: 0.8244 -
val_AUC: 0.9135
Epoch 24/100
782/782 [==============================] - ETA: 0s - loss: 0.3266 - accuracy:
0.8554 - AUC: 0.9343
Epoch 00024: val_AUC did not improve from 0.91487
782/782 [==============================] - 108s 138ms/step - loss: 0.3266 -
accuracy: 0.8554 - AUC: 0.9343 - val_loss: 0.3725 - val_accuracy: 0.8280 -
val_AUC: 0.9141
Epoch 25/100
782/782 [==============================] - ETA: 0s - loss: 0.3221 - accuracy:
0.8580 - AUC: 0.9361
Epoch 00025: val_AUC did not improve from 0.91487
782/782 [==============================] - 123s 158ms/step - loss: 0.3221 -
accuracy: 0.8580 - AUC: 0.9361 - val_loss: 0.3825 - val_accuracy: 0.8276 -
val_AUC: 0.9142
Epoch 26/100
782/782 [==============================] - ETA: 0s - loss: 0.3206 - accuracy:
0.8590 - AUC: 0.9367
Epoch 00026: val_AUC did not improve from 0.91487
782/782 [==============================] - 127s 162ms/step - loss: 0.3206 -
accuracy: 0.8590 - AUC: 0.9367 - val_loss: 0.3769 - val_accuracy: 0.8284 -
val_AUC: 0.9144
Epoch 27/100
782/782 [==============================] - ETA: 0s - loss: 0.3195 - accuracy:
0.8608 - AUC: 0.9372
Epoch 00027: val_AUC did not improve from 0.91487
782/782 [==============================] - 125s 160ms/step - loss: 0.3195 -
accuracy: 0.8608 - AUC: 0.9372 - val_loss: 0.3782 - val_accuracy: 0.8279 -
val_AUC: 0.9142
```

[29]:
```
y_score = rnn.predict(X_test_padded)
roc_auc_score(y_score=y_score.squeeze(), y_true=y_test)
```

[29]: 0.914964528

[30]:
```
df = pd.DataFrame(training.history)
best_auc = df.val_AUC.max()
best_acc = df.val_accuracy.max()

fig, axes = plt.subplots(ncols=2, figsize=(14,4))
df.index = df.index.to_series().add(1)
df[['AUC', 'val_AUC']].plot(ax=axes[0],
                            title=f'AUC | Best: {best_auc:.4f}',
                            legend=False,
                            xlim=(1, 33),
                            ylim=(.7, .95))

axes[0].axvline(df.val_AUC.idxmax(), ls='--', lw=1, c='k')
df[['accuracy', 'val_accuracy']].plot(ax=axes[1],
                                      title=f'Accuracy | Best:␣
 ↪{best_acc:.2%}',
                                      legend=False,
                                      xlim=(1, 33),
                                  ylim=(.7, .9))
axes[1].axvline(df.val_accuracy.idxmax(), ls='--', lw=1, c='k')
axes[0].set_xlabel('Epoch')
axes[0].set_ylabel('AUC')
axes[1].set_xlabel('Epoch')
axes[1].set_ylabel('Accuracy')
fig.suptitle('Sentiment Analysis - Pretrained Vectors', fontsize=14)
fig.legend(['Train', 'Validation'], loc='center right')

sns.despine()
fig.tight_layout()
fig.subplots_adjust(top=.9)
fig.savefig(results_path / 'imdb_pretrained', dpi=300);
```