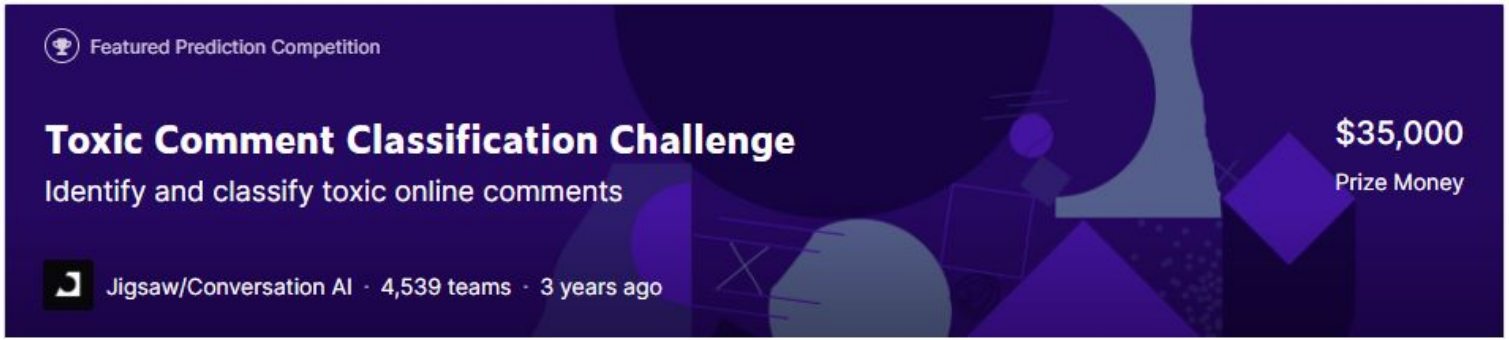
Toxic Comment Classification Challenge

이상윤


Featured Prediction Competition
Toxic Comment Classification Challenge
Identify and classify toxic online comments
$35,000
Prize Money
Jigsaw/Conversation AI · 4,539 teams · 3 years ago

- Kaggle Data

Data Description

You are provided with a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity are:

- `toxic`
- `severe_toxic`
- `obscene`
- `threat`
- `insult`
- `identity_hate`

You must create a model which predicts a probability of each type of toxicity for each comment.

토큰화

```
from keras.preprocessing.text import Tokenizer
from keras import preprocessing
from keras.models import Model
from keras.layers import Dense, Input, Dropout, Activation
from keras.layers import Bidirectional, LSTM, Embedding, GlobalMaxPool1D
from keras import initializers, regularizers, constraints, optimizer_v1, layers

embed_size = 50
max_features = 20000
max_len = 200

tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(train_text))
```

```
list_tokenized_train = tokenizer.texts_to_sequences(train_text)
list_tokenized_test = tokenizer.texts_to_sequences(test_text)

X_train = preprocessing.sequence.pad_sequences(list_tokenized_train, maxlen=max_len)
X_test = preprocessing.sequence.pad_sequences(list_tokenized_test, maxlen=max_len)
```

GloVe 단어 임베딩 전처리

```
[14]  import os

      glove_dir = '/content/'

      embeddings_index = {}
      f = open(os.path.join(glove_dir, 'glove.6B.50d.txt'), encoding="utf8")
      for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
      f.close()

      print('%s개의 단어 벡터..' %len(embeddings_index))
```

```
400000개의 단어 벡터..
```

임베딩 행렬

```
[15] all_embs = np.stack(embeddings_index.values())
     emb_mean, emb_std = all_embs.mean(), all_embs.std()
     emb_mean, emb_std
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2822: FutureWarning: arrays to stack must be passed as
  if self.run_code(code, result):
(0.020940498, 0.6441043)
```

```
[16] tokenizer.word_index.items()
```

```
dict_items([('the', 1), ('to', 2), ('of', 3), ('and', 4), ('a', 5), ('you', 6), ('i', 7), ('is', 8), ('that', 9), ('in', 10),
```

```
[17] word_index = tokenizer.word_index
     nb_words = min(max_features, len(word_index))
     embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))

     for word, i in word_index.items():
       if i >= max_features:
         continue
       embedding_vector = embeddings_index.get(word)
       if embedding_vector is not None :
         embedding_matrix[i] = embedding_vector
```

np.std : 표준편차

np.mean : 평균

모델 정의

Embedding

Bidirectional LSTM

max_pooling1d

dense & dropout

```
[20] from keras.models import Sequential

     model = Sequential([
         layers.Embedding(input_dim=max_features, output_dim=embed_size,
                          input_length=max_len, weights=[embedding_matrix]),
         layers.Bidirectional(layers.LSTM(60, return_sequences=True,
                                          dropout=0.1, recurrent_dropout=0.1)),
         layers.GlobalMaxPool1D(),
         layers.Dropout(0.1),
         layers.Dense(50, activation='relu'),
         layers.Dropout(0.1),
         layers.Dense(6, activation='sigmoid')])

     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

     model.summary()
```

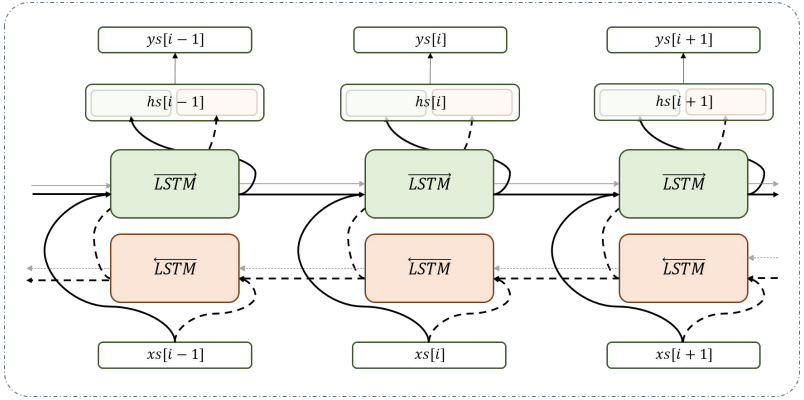
```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 200, 50)           1000000
_________________________________________________________________
bidirectional (Bidirectional (None, 200, 120)          53280
_________________________________________________________________
global_max_pooling1d (Global (None, 120)               0
_________________________________________________________________
dropout (Dropout)            (None, 120)               0
_________________________________________________________________
dense (Dense)                (None, 50)                6050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 306
=================================================================
Total params: 1,059,636
Trainable params: 1,059,636
Non-trainable params: 0
_________________________________________________________________
```

Bidirectional LSTM

양방향 LSTM

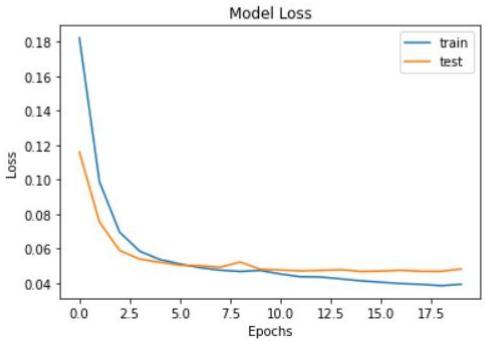
정방향과 역방향 (양방향) 모두 추론에 활용

ex) 나는 _____를 뒤집어 쓰고 펑펑 울었다.



결과

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```




```
batch_size = 2048
epochs = 20

history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,validation_split=0.2,verbose=1)
```

```
Epoch 1/100
63/63 [==============================] - 142s 2s/step - loss: 0.3433 - accuracy: 0.2039 - val_loss: 0.1323 - val_accuracy: 0.9938
Epoch 2/100
63/63 [==============================] - 112s 2s/step - loss: 0.1331 - accuracy: 0.6944 - val_loss: 0.0900 - val_accuracy: 0.9937
Epoch 3/100
63/63 [==============================] - 111s 2s/step - loss: 0.0876 - accuracy: 0.7272 - val_loss: 0.0631 - val_accuracy: 0.9938
Epoch 4/100
63/63 [==============================] - 109s 2s/step - loss: 0.0644 - accuracy: 0.8159 - val_loss: 0.0548 - val_accuracy: 0.9915
Epoch 5/100
63/63 [==============================] - 110s 2s/step - loss: 0.0559 - accuracy: 0.8266 - val_loss: 0.0521 - val_accuracy: 0.9917
Epoch 6/100
63/63 [==============================] - 113s 2s/step - loss: 0.0522 - accuracy: 0.8472 - val_loss: 0.0512 - val_accuracy: 0.9929
Epoch 7/100
63/63 [==============================] - 113s 2s/step - loss: 0.0515 - accuracy: 0.8890 - val_loss: 0.0500 - val_accuracy: 0.9909
Epoch 8/100
63/63 [==============================] - 112s 2s/step - loss: 0.0491 - accuracy: 0.8673 - val_loss: 0.0491 - val_accuracy: 0.9905
Epoch 9/100
63/63 [==============================] - 111s 2s/step - loss: 0.0478 - accuracy: 0.8851 - val_loss: 0.0487 - val_accuracy: 0.9876
Epoch 10/100
63/63 [==============================] - 111s 2s/step - loss: 0.0460 - accuracy: 0.8773 - val_loss: 0.0480 - val_accuracy: 0.9927
Epoch 11/100
63/63 [==============================] - 110s 2s/step - loss: 0.0454 - accuracy: 0.9028 - val_loss: 0.0475 - val_accuracy: 0.9919
Epoch 12/100
63/63 [==============================] - 112s 2s/step - loss: 0.0447 - accuracy: 0.9063 - val_loss: 0.0482 - val_accuracy: 0.9932
Epoch 13/100
63/63 [==============================] - 110s 2s/step - loss: 0.0435 - accuracy: 0.9142 - val_loss: 0.0472 - val_accuracy: 0.9894
Epoch 14/100
63/63 [==============================] - 112s 2s/step - loss: 0.0432 - accuracy: 0.8986 - val_loss: 0.0477 - val_accuracy: 0.9917
Epoch 15/100
63/63 [==============================] - 112s 2s/step - loss: 0.0425 - accuracy: 0.9180 - val_loss: 0.0469 - val_accuracy: 0.9931
Epoch 16/100
63/63 [==============================] - 111s 2s/step - loss: 0.0415 - accuracy: 0.9160 - val_loss: 0.0478 - val_accuracy: 0.9907
Epoch 17/100
63/63 [==============================] - 111s 2s/step - loss: 0.0404 - accuracy: 0.9035 - val_loss: 0.0468 - val_accuracy: 0.9909
Epoch 18/100
63/63 [==============================] - 112s 2s/step - loss: 0.0388 - accuracy: 0.8956 - val_loss: 0.0468 - val_accuracy: 0.9891
Epoch 19/100
63/63 [==============================] - 110s 2s/step - loss: 0.0391 - accuracy: 0.9067 - val_loss: 0.0472 - val_accuracy: 0.9896
Epoch 20/100
63/63 [==============================] - 111s 2s/step - loss: 0.0373 - accuracy: 0.9096 - val_loss: 0.0467 - val_accuracy: 0.9917
```