2021-1 STA618 고급통계적딥러닝

FINAL REPORT

USE OF TRANSFER LEARNING IN IMBALANCED TEXT CLASSIFICATION

2020021218 통계학과 이소연

erinlee42@korea.ac.kr

1. Introduction

1.1. 레포트 목적 및 개요

클래스 불균형(Class Imbalance)은 특정 데이터에서 각 범주가 갖고 있는 데이터의 양에 큰 차이가 있는 경우를 말한다. 실생활 속 이러한 불균형 데이터는 다양하게 존재한다. 필자는 지난 학기, '고급통계적머신러닝'에서 불균형 데이터에 다양한 재표집(Resampling) 방법을 적용하고, 각 방법마다 최고성능을 도출하는 머신러닝 모형을 탐구해본 적이 있다. 본 프로젝트는 앞선 프로젝트의 일환으로, 필자가 지속적으로 관심 있는 불균형 텍스트 자료의 분류에 있어 딥러닝의 세 가지 이전학습(Transfer Learning)방식을 적용하고, 그 효과를 분석해볼 것이다.

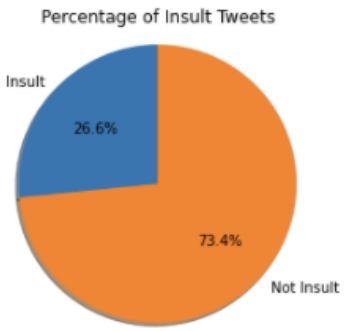
1.2. 데이터 소개

본 프로젝트는 Wisconsin-Madison 대학에서 공개한 트위터 기반의 bullyingV3 데이터셋[1]을 사용하였다. 다음은 'Comment'열에 대해 전처리를 진행한 결과이다. 불용어(Stopwords), html 태그, 교정부호와 특수문자, 그리고 영어 및 숫자가 아닌 단어를 삭제한 다음, 모든 단어를 소문자로 변환하였다. 'Insult'열은 해당 텍스트가 비방 트윗이면 1, 아니라면 0을 표시한다. 이를 가장 잘 분류하는 모형을 찾는 것이 본 프로젝트의 목표이다.

```
[32] data.drop("Comment",axis=1,inplace=True)
     data['CleanText'] = txt
     data.reset_index(inplace=False)
     data[["Insult","CleanText"]].head()
```

	Insult	CleanText
0	1	fuck dad
1	0	really understand seems mixing apples oranges
2	0	canadians wrong n nunless supportive idea noth...

비방 트윗의 비율은 전체 데이터의 약 26.6%으로, 클래스 불균형 문제가 심하게 나타나는 것을 확인할 수 있다.



[1] **Data for the study of bullying** (https://research.cs.wisc.edu/bullying/data.html)

1.3. 진행방향

비방 트윗을 분류하는 모형의 학습에 세 가지 이전학습 방법을 적용하여, 학습 데이터의 클래스 불균형 문제를 완화해보고자 한다. 첫 번째로, GLOVE의 word embedding을 이용한 이전학습을 통해 Embedding 층을 사전 정의하는 방법이다. 두 번째로, 클래스 비율이 1:1인 가짜 문장을 생성하여 학습시킨 신경망 레이어를 실제 분류 모형에 추가하는 방법(서수인[2])이 있다. 마지막으로, Wikitext-103 데이터셋과 AWD-LSTM 모델로 이미 pre-trained 되어있는 베이스 모델인 ULMFiT[3]을 사용하여 fine-tuning을 하는 방법이다. 다음은 ULMFiT 기반의 이전학습 절차를 도식화한 것이다.

[그림 1] ULMFiT의 학습 절차

2. **Model Fitting**

모든 모형 구축에 있어 EarlyStopping(patience=2)을 통해 최적의 에폭수를 결정하였으며, monitor 는 'val_auc'로 설정하였다. 이는 데이터 불균형이 심한 경우, 보편적으로 PR_AUC 를 모델 평가 지표로 사용하기 때문이다. 주어진 데이터가 불균형자료일 때, 정확도만으로 모델을 평가하는 것은 바람직하지 않다. 본 프로젝트의 데이터의 경우, 모든 경우에 대하여 무조건 'Not Insult'로 분류한다면 약 73.4%의 정확도가 나올 수 있기 때문이다.

2.1. Without Transfer Learning

주어진 데이터에서 Tokenizer 클래스를 import한 후 총 10000개의 단어로 토큰화하도록 설정하였다. 각 표본의 계열 길이는 100개로 설정하였으며, 그 결과 학습데이터가 (3947, 100)의 행렬로 토큰화되었다.

---

[2] 가짜문장의 전이학습을 이용한 소규모 불균형 데이터의 욕설문장 분류방법 (https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE07207572)

[3] Universal Language Model Fine-tuning for Text Classification (https://arxiv.org/abs/1801.06146)

```
[27] max_words=10000
     maxlen=100
     token=Tokenizer(num_words=max_words)
     token.fit_on_texts(data['CleanText'].values)
     sequence=token.texts_to_sequences(data['CleanText'].values)
     word_index=token.word_index
     print(len(word_index))
     labels=data['Insult'].values
     data=pad_sequences(sequence, maxlen=maxlen)
     print(data.shape)
     print(labels.shape)

     14400
     (3947, 100)
     (3947,)
```

다음으로 아래와 같이 이전학습 없이 양방향 LSTM을 사용하여 RNN 모형을 구축하였다.

```
[39] model_gl=Sequential()
     model_gl.add(Embedding(max_words,embedding_dim,input_length=maxlen))
     model_gl.add(Bidirectional(LSTM(64),merge_mode='concat'))
     model_gl.add(Flatten())
     model_gl.add(Dropout(0.3))
     model_gl.add(Dense(1,activation='sigmoid'))
     model_gl.summary()
     callback_list=[EarlyStopping(monitor='val_auc', patience=2)]
     model_gl.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['AUC'])
     model_gl.fit(x_train,y_train, epochs=5, batch_size=128, validation_data=(x_test, y_test), callbacks=callback_list)

     train_tf = model_gl.predict_classes(x_train)
     precision, recall, thresholds = precision_recall_curve(y_train, train_tf)
     auc_precision_recall = auc(recall, precision)
     print("Train:₩n","F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC ₩n",round(f1_score(y_train,train_tf, average="micro"),5),'       /' ,
           round(f1_score(y_train,train_tf, average="macro"),5),'   /' ,
           round(auc_precision_recall,5), '    /' ,round(roc_auc_score(y_train,train_tf),5))

     test_tf = model_gl.predict_classes(x_test)
     precision, recall, thresholds = precision_recall_curve(y_test, test_tf)
     auc_precision_recall = auc(recall, precision)
     print("Test:₩n","F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC ₩n",round(f1_score(y_test,test_tf, average="micro"),5),'       /' ,
           round(f1_score(y_test,test_tf, average="macro"),5),'   /' ,
           round(auc_precision_recall,5), '    /' ,round(roc_auc_score(y_test,test_tf),5))
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
bidirectional_2 (Bidirection (None, 128)               84480
_________________________________________________________________
flatten_2 (Flatten)          (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,084,609
Trainable params: 1,084,609
Non-trainable params: 0
_________________________________________________________________
```

그 결과, 아래와 같이 학습 데이터에 대해선 약 68%, 테스트 데이터에 대해선 약 63%의 AUC 값을 가졌다. 모든 클래스를 동등하게 취급하는 Macro F1-score 가 각 55%, 48%인 것을 보면, 비방 트윗을 분류하기엔 성능이 다소 떨어짐을 확인할 수 있다. Train, test 데이터에 대해 전체적인 지표의 차이가 크게 나진 않아 모델 적합에 있어서 과대적합 문제는 일어나지 않았다고 판단하였다.

```
Train:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.77046         / 0.55054     / 0.68171     / 0.56694
Test:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.74768         / 0.48284     / 0.63071     / 0.52939
```

2.2. With Transfer Learning - GLOVE

GLOVE 모형에서 이미 만들어진 100 차원 word embedding 벡터를 불러와, 다음과 같이 앞서 만들어진 word_index 를 GLOVE 의 단어 인덱스로 전환하였다.

```
glove_dir='/content/drive/MyDrive/딥러닝/딥러닝실습데이터/glove.6B'
embedding_index={}
f=open(os.path.join(glove_dir,'glove.6B.100d.txt'),encoding='UTF8')
for line in f:
    values=line.split()
    word=values[0]
    seq=np.asarray(values[1:],dtype='float32')
    embedding_index[word]=seq
f.close
```

```
embedding_dim=100
embedding_matrix=np.zeros((max_words,embedding_dim))
for word,i in word_index.items():
    if i<max_words:
        embedding_vector=embedding_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i]=embedding_vector
```

앞선 모델에서 Embedding 층을 GLOVE word embedding 으로 대체하였으므로, 모수를 재추정하지 못하도록 기존 코드에서 모델 compile 전 아래 코드를 추가하였다.

```
model_gl.layers[0].set_weights([embedding_matrix])
model_gl.layers[0].trainable=False
```

그 결과, 아래와 같이 이전학습이 없는 모델의 결과에 비해 모든 지표가 확연히 상향된 것을 확인할 수 있다. 학습 데이터와 테스트 데이터에 대해서 모두 약 79%의 AUC 값을 가졌으며, Macro F1-score 또한 각 71%, 67%으로, 약 15%p ~20%p 정도 성능이 향상되었다.

```
Train:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.80196         / 0.71411     / 0.79149     / 0.69494
Test:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.76793         / 0.66893     / 0.78933     / 0.65536
```

약간의 과대적합을 해소하기 위해 모형의 은닉층엔 Dropout(0.1)을, 출력층엔 l2 규제화를 추가한 결과, 아래와 같이 Train, test 데이터에 대해 전체적인 지표들의 차이가 줄어들었다.

```
Train:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.79833         / 0.69277     / 0.79806     / 0.67107
Test:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.76962         / 0.68247     / 0.78917     / 0.63749
```

2.3. With Transfer Learning – Faker

다음으로, 클래스 비율이 1:1 인 임의의 학습 문장을 생성하고, 이를 학습한 레이어를 모델에 추가하여, 실제 훈련 문장을 학습하는 이전 학습 방식을 진행해보았다. 가짜 문장은 트위터 기반의 파이썬 오픈 소스 패키지인 Faker[4]를 통해 총 만 문장을 생성하였으며, 구글에서 '욕설'로 판정하는 google-bad-words 리스트에서 임의의 욕설단어를 추출하여 그 중 절반의 문장에 첨가하였다. 문장 생성에 사용한 구체적인 코드는 아래와 같다.

```
good = list()
bad = list()

for _ in range(5000):
  txt = fake.text()
  good.append(txt)

for _ in range(5000,10000):
  temp = fake.text()
  txt = temp.replace(random.choice(temp.split()),random.choice(swear))
  bad.append(txt)

bad[:2]
```

```
['Task process think sing pussy your. Candidate dinner point. Concern others onto author chair little score.₩nAlthough court short.',
 'Really nothing firm. Sort market wife toward.₩nEnvironmental win edge high involve add.₩nChallenge fuckme  we produce. Later else win everyone. Manage kind be hard which.']
```

그 다음으로, 가짜 문장을 전처리 하여 아래와 같이 분류를 위한 RNN 모형을 구축하였다. 클래스 비율이 1:1 이기 때문에 accuracy 를 기준으로 모형을 평가한다면, pre-train 결과 train 데이터는 약 95%, test 데이터는 약 92%의 정확도를 보였다. 과대 적합은 나타나지 않은 것으로 판단하였다.

```
X_train=sequence.pad_sequences(X_train,maxlen=100)
X_test=sequence.pad_sequences(X_test,maxlen=100)
m_f=Sequential()
m_f.add(Embedding(max_features, 32))
m_f.add(Bidirectional(LSTM(32),merge_mode='concat'))
m_f.add(Dense(1, activation='sigmoid'))
m_f.summary()
m_f.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['acc'])
m_f.fit(X_train,y_train, epochs=10, batch_size=128, validation_data=(X_test, y_test))
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64)                16640
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 336,705
Trainable params: 336,705
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 10/10
55/55 [==============================] - 112s 2s/step - loss: 0.1697 - acc: 0.9483 - val_loss: 0.2284 - val_acc: 0.9233
```

이제 앞서 학습시킨 신경망에 실제 훈련문장을 학습시킨 후, 테스트 문장으로 성능을 평가해보았다. 자세한 구현 절차는 논문에 기재된 바를 참고하였으며, 과대적합을 방지하기 위해 앞선 예시와 같이 Dropout(0.1)과 l2 규제화를 추가하였다.

---

[4] *Faker* (https://github.com/joke2k/faker)

```
m_f.trainable=False
model_gl=Sequential()
model_gl.add(m_f)
model_gl.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model_gl.add(Dropout(0.1))
model_gl.add(Dense(1,activation='sigmoid', kernel_regularizer='l2'))
model_gl.summary()
callback_list=[EarlyStopping(monitor='val_auc', patience=2)]


model_gl.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['AUC'])
gl_result = model_gl.fit(x_train,y_train, epochs=5, batch_size=128, validation_data=(x_test, y_test), callbacks=callback_list)

train_tf = model_gl.predict_classes(x_train)
precision, recall, thresholds = precision_recall_curve(y_train, train_tf)
auc_precision_recall = auc(recall, precision)
print("Train:₩n","F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC ₩n",round(f1_score(y_train,train_tf, average="micro"),5),'        /' ,
      round(f1_score(y_train,train_tf, average="macro"),5),'    /' ,
      round(auc_precision_recall,5), '     /' ,round(roc_auc_score(y_train,train_tf),5))

test_tf = model_gl.predict_classes(x_test)
precision, recall, thresholds = precision_recall_curve(y_test, test_tf)
auc_precision_recall = auc(recall, precision)
print("Test:₩n","F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC ₩n",round(f1_score(y_test,test_tf, average="micro"),5),'        /' ,
      round(f1_score(y_test,test_tf, average="macro"),5),'    /' ,
      round(auc_precision_recall,5), '     /' ,round(roc_auc_score(y_test,test_tf),5))
```

그 결과, 아래와 같이 이전학습이 없는 모델의 결과에 비해선 모든 지표가 상향되었으나, Micro F1-score 을 제외한 나머지 지표에 대해선 앞선 GLOVE 보단 못 미치는 성능을 보였다. 학습 데이터와 테스트 데이터에 대해서 각각 약 64%, 63%의 AUC 값을 가졌으며, Macro F1-score 는 약 58%, 51%를 보였다.

```
Train:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.75317         / 0.58209     / 0.63779     / 0.55140
Test:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.74993         / 0.51046     / 0.62936     / 0.50003
```

2.4. With Transfer Learning – ULMFIT

다음으로, 대용량의 학습데이터로 이미 pre-trained 된 언어모델(Language Model)을 기반으로 하여, Embedding 레이어 및 모델의 중간 레이어에 이전학습을 적용하는 ULMFIT 을 사용하였다. 즉, Computer Vision 이나 이미지 데이터 분류 등에서 사용되는 transfer learning 과 같은 효과를 텍스트 처리에서도 기대할 수 있는 것이다. 먼저, 아래와 같이 이미 내장되어 있는 위키피디아 데이터로 학습된 모델을 불러왔다.

```
[7]  from fastai.text import *
     learn = language_model_learner(data_lm, AWD_LSTM, drop_mult=0.5)
     learn.fit_one_cycle(1, 1e-2)
     learn.model
```

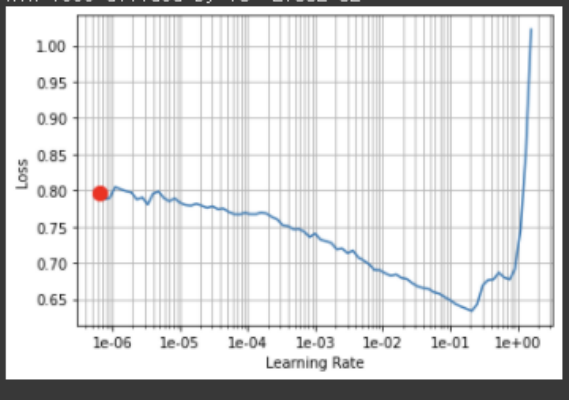
Pre-trained 모델의 구조는 다음장의 그림과 같이 3-layer LSTM 으로 구성되어 있다.

```
SequentialRNN(
  (0): AWD_LSTM(
    (encoder): Embedding(8896, 400, padding_idx=1)
    (encoder_dp): EmbeddingDropout(
      (emb): Embedding(8896, 400, padding_idx=1)
    )
    (rnns): ModuleList(
      (0): WeightDropout(
        (module): LSTM(400, 1152, batch_first=True)
      )
      (1): WeightDropout(
        (module): LSTM(1152, 1152, batch_first=True)
      )
      (2): WeightDropout(
        (module): LSTM(1152, 400, batch_first=True)
      )
    )
    (input_dp): RNNDropout()
    (hidden_dps): ModuleList(
      (0): RNNDropout()
      (1): RNNDropout()
      (2): RNNDropout()
    )
  )
  (1): LinearDecoder(
    (decoder): Linear(in_features=400, out_features=8896, bias=True)
    (output_dp): RNNDropout()
  )
)
```

이제 위 모델을 encoder 로 저장하고, 이를 기반으로 새로운 분류기를 만들 것이다. 초모수의 경우, 아래와 같이 optimal learning rate 을 찾았다.

```
[53] clas_model.lr_find()
     clas_model.recorder.plot(show_grid=True, suggestion=True)

     Min numerical gradient: 6.31E-07
     Min loss divided by 10: 2.09E-02
```



세부적인 모델 구축 과정은 ULMFiT 의 저자인 Howard 의 논문 속 지시 사항을 참고하였다. 자세한 코드는 다음 장에 첨부했다.

```
[50] # Classifier model data
     data_clas = TextClasDataBunch.from_df(path = "", train_df = data, valid_df = data_test, vocab=data_lm.train_ds.vocab, bs=32)
     clas_model = text_classifier_learner(data_clas, AWD_LSTM, drop_mult=0.5)
     clas_model.fit_one_cycle(5,6.31E-07,moms=(0.8,0.7))
     clas_model.load_encoder('encoder')
     train_tf = x_train.apply(lambda row:str(clas_model.predict(row)[0]))

     precision, recall, thresholds = precision_recall_curve(y_train, train_tf)
     auc_precision_recall = auc(recall, precision)
     print("Train:₩n","F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC ₩n",round(f1_score(y_train,train_tf, average="micro"),5),'       /' ,
           round(f1_score(y_train,train_tf, average="macro"),5),'   /' ,
           round(auc_precision_recall,5), '    /' ,round(roc_auc_score(y_train,train_tf),5))

     test_tf = x_test.apply(lambda row:str(clas_model.predict(row)[0]))
     precision, recall, thresholds = precision_recall_curve(y_test, test_tf)
     auc_precision_recall = auc(recall, precision)
     print("Test:₩n","F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC ₩n",round(f1_score(y_test,test_tf, average="micro"),5),'       /' ,
           round(f1_score(y_test,test_tf, average="macro"),5),'   /' ,
           round(auc_precision_recall,5), '    /' ,round(roc_auc_score(y_test,test_tf),5))
```

그 결과, 이전과 같이 이전학습이 없는 모델의 결과에 비해선 모든 지표가 상향되었으나, GLOVE 를 통한 이전학습의 결과보단 못 미치는 성능을 보였다. 하지만 가짜문장을 통해 이전학습을 진행한 경우 보단 macro F1-score 과 ROC-AUC 값이 더 향상되었다. 학습 데이터와 테스트 데이터에 대해서 각각 약 69%, 63%의 AUC 값을 가져 과대적합의 가능성이 보였지만, Macro F1-score 는 모두 약 58%로 큰 차이를 보이지 않아 큰 문제가 되지 않는다고 결론지었다.

```
Train:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.75393         / 0.58277     / 0.69806     / 0.61917
Test:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.73249         / 0.58127     / 0.63376     / 0.57369
```

3. Different Deep Learning Models

이번엔 분류 성능을 보다 향상시키기 위해 기존에 사용한 쌍방향 LSTM 외에 다른 딥러닝 모형을 적용해볼 것이다. 이전학습은 세 방법 중 모든 지표에 있어 성능이 가장 우수했던 GLOVE 방법을 사용하고, 딥러닝 모형의 경우, 1) Simple RNN, 2) GRU, 그리고 3) CNN 모형을 적용해볼 것이다. 모델 적합 시 기존 LSTM 과 동일하게 Dropout 및 정규화로 과대적합을 방지하고, EarlyStopping(patience=2)을 통해 최적의 에폭수를 결정하였으며, monitor 는 'val_auc'로 설정하였다. 각 모델별 결과에 대한 설명은 첨부한 그림으로 대체하겠다.

3.1. Simple RNN

```
Train:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.79725         / 0.67313     / 0.75682     / 0.65155
Test:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.77131         / 0.63629     / 0.74344     / 0.62262
```

3.2. GRU

```
Train:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.80014         / 0.7143    / 0.80164     / 0.69633
Test:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.77722         / 0.68425    / 0.78183     / 0.6697
```

3.3. CNN

CNN 모형의 경우, 크기가 (100,100)인 2D텐서에 Conv1D를 두 번 사용하고, GlobalMaxPooling1D를 통해 1D텐서로 전환되는 구조를 가졌다. 하지만 아래와 같이, 다른 RNN 모형에 비해 좋은 성능을 내진 못했다.

```
Train:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.72882         / 0.59108    / 0.67788     / 0.58535
Test:
 F1 score(micro) / F1 score(macro) / PR-AUC / ROC-AUC
 0.69198         / 0.53673    / 0.65103     / 0.53943
```

4. Model & Method Selection

이제 모형의 분류 성능을 판단할 것이다. Micro F1과 AUC는 다수 클래스 혹은 데이터 전반에 대한 성능을 다루는 지표이며, Macro F1, ROC-AUC는 데이터의 불균형을 무시하고, 소수 클래스의 분류 성능 또한 중시할 때 사용하는 지표이다. 본 프로젝트의 경우, 전체적인 데이터에 대한 성능도 중요하지만, 소수 클래스인 '비방 트윗'을 잘 분류할 수 있는 지도 중요한 평가 요소이기 때문에 네 지표간 특별한 우위 없이, 전반적인 수치로 모델을 평가할 것이다. 모든 결과에 대해 오버피팅의 문제가 없다고 판단하여, 앞서 다룬 모든 모형의 Test 데이터에 대한 결과를 기준으로, bullyingV3 데이터셋 가장 적합한 모형을 찾겠다.

[The Effect of Transfer Learning]

	MICRO F1	MACRO F1	AUC	ROC-AUC
ORIGINAL	0.75	0.48	0.63	0.53
GLOVE	**0.77**	**0.68**	**0.79**	**0.64**
FAKER	0.75	0.51	0.63	0.50
ULMFIT	0.73	0.58	0.63	0.57

먼저, 이전학습의 효과에 관한 표이다. 이전학습을 적용하지 않은 결과에 비해, 이전학습을

적용한 결과의 분류 성능이 눈에 띄게 높음을 확인할 수 있다. 특히, 지표들 중 소수 클래스의 분류 성능에 가장 큰 비중을 두는 MACRO F1값을 보면, 세 이전학습 모형의 성능이 확실히 좋아졌음을 확인할 수 있다. 따라서, 소수 클래스의 분류 성능을 중요시한다면, 이전학습을 하는 것이 큰 도움이 된다고 결론지을 수 있다. 특히, 본 bullyingV3 데이터셋의 경우, GLOVE를 사전학습모형으로 하는 이전학습을 적용하였을 때 가장 좋은 성능을 보였다.

[Comparing RNN Models]

	MICRO F1	MACRO F1	AUC	ROC-AUC
ORIGINAL	0.75	0.48	0.63	0.53
LSTM	**0.77**	**0.68**	**0.79**	**0.64**
SIMPLE RNN	0.77	0.64	0.74	0.62
GRU	**0.78**	**0.68**	**0.78**	**0.67**

다음으로, 전체적인 성능이 떨어졌던 CNN 모형을 제외한 세 가지 RNN 모형의 성능을 비교해보았다. 전체적으로 비슷한 값을 도출했지만, 그 중에서도 LSTM과 GRU를 사용했을 때 가장 좋은 성능을 보였다.

위 결과를 종합하면, 클래스 불균형이 있을 때 이전학습을 사용하지 않은 모형은 소수 클래스에 대한 분류 성능이 현저히 떨어지며, 따라서 GLOVE 기반의 이전학습에 LSTM이나 GRU를 사용한 딥러닝 모형이 최적의 결과를 도출한다고 결론지을 수 있다.

5. Final Comment

본 프로젝트를 통해 데이터의 불균형을 처리하기 위한 다양한 이전학습 방법을 탐구할 수 있었다. 이번에 다루지 않았지만, BERT나 ELMO와 같은 대용량 언어모델을 사용하는 방법도 있었으며, 이전학습에서 나아가, 학습데이터를 상호 배타적으로 나누어, 점차 target distribution(이 경우 uniform)에 이르도록 adaptively train하는 'incremental learning' 방법론도 접하게 되었다. 필자는 불균형 텍스트 자료 분석에 대한 관심을 지속하여, 이러한 다른 방법론들 또한 직접 구현해볼 것이다. 또한, 다음 학기 '고급 통계적 강화학습'을 수강한 후 NLP에 적용할 수 있는 강화학습 기반의 방법론을 새로 배워, 결국 가장 optimal한 분류 방법론을 찾아내고자 한다.