COSE362 기계학습 텀프로젝트

컴퓨터학과 2020320078 한지상

1. 실행환경

- Jupyter Notebook (Anaconda), RTX3070 Laptop GPU(8GB)

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 471.96       Driver Version: 471.96       CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ... WDDM  | 00000000:01:00.0  On |                  N/A |
| N/A   47C    P8    13W /  N/A |    279MiB /  8192MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      1544    C+G   Insufficient Permissions        N/A      |
|    0   N/A  N/A      7428    C+G   Insufficient Permissions        N/A      |
|    0   N/A  N/A     18788    C+G   ...h8wxbdkxb8p₩DCv2₩DCv2.exe    N/A      |
+-----------------------------------------------------------------------------+
```

- 이후 GPU메모리부족으로 Google Colab과 Kaggle notebook에서 실행하였음.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 495.44       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   68C    P8    32W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

2. 데이터에 대한 분석과 정제 방법

```
data = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
print(data.shape, test.shape)
```

```
(3620, 3) (1551, 2)
```

데이터 크기 확인

```
data.dropna(inplace=True)
test.dropna(inplace=True)
```

결측값 삭제

```
remove_non_alphabets =lambda x: re.sub(r'[^a-zA-Z]',' ',x)

tokenize = lambda x: word_tokenize(x)

ps = PorterStemmer()
stem = lambda w: [ ps.stem(x) for x in w ]

lemmatizer = WordNetLemmatizer()
leammtizer = lambda x: [ lemmatizer.lemmatize(word) for word in x ]

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
data['mail'] = data['mail'].apply(remove_non_alphabets)
test['mail'] = test['mail'].apply(remove_non_alphabets)

data['mail'] = data['mail'].apply(tokenize)
test['mail'] = test['mail'].apply(tokenize)

data['mail'] = data['mail'].apply(stem)
test['mail'] = test['mail'].apply(stem)

data['mail'] = data['mail'].apply(leammtizer)
test['mail'] = test['mail'].apply(leammtizer)

data['mail'] = data['mail'].apply(lambda x: ' '.join(x))
test['mail'] = test['mail'].apply(lambda x: ' '.join(x))
```

nltk 라이브러리를 사용하여 Non words를 삭제하고 소문자로 바꾼다. 그리고 토큰화(Tokenize), 어간추출(Stemming), 표제어 추출(Leammtization)을 하여 데이터를 정제하였다. 소문자로 바꿈으로써, 그리고 어간과 표제어를 추출함으로써 단어의 개수를 줄일 수 있다.

```
max_words = 40000
#cv = CountVectorizer(max_features=max_words, stop_words='english')
from sklearn.feature_extraction.text import TfidfVectorizer
cv = TfidfVectorizer(max_features=max_words, stop_words='english', ngram_range=(1,2))

train_vectors = cv.fit_transform(data['mail']).toarray()
test_vectors = cv.transform(test['mail']).toarray()

print(train_vectors.shape)
print(test_vectors.shape)
```

```
(3620, 40000)
(1551, 40000)
```

40000개의 단어를 사용하여 TF-IDF Vectorization을 진행하였다. 문장에서 자주 등장하지만 분석에 큰 도움이 되지 않는 stopwords를 제거하였고, 단어의 묶음을 1-2개로 설정하여 단어 묶음을 학습할 수 있도록 하였다.

```
x_train, x_test, y_train, y_test = train_test_split(train_vectors, np.array(data['label']))
```

마지막으로 Train set과 Test(Validation) set으로 데이터를 나누었다.

3. 구현한 모델에 대한 설명

```
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.linear1 = nn.Linear(40000, 10000)
        self.linear2 = nn.Linear(10000, 100)
        self.linear3 = nn.Linear(100, 10)
        self.linear4 = nn.Linear(10, 2)

    def forward(self, x):
        x = F.relu(self.linear1(x))
        x = F.relu(self.linear2(x))
        x = F.relu(self.linear3(x))
        x = self.linear4(x)
        return x
```

pyTorch를 사용하여 input node가 40000개, output node가 2개인 간단한 neural network를 모델로 사용하였다. Activation function은 ReLU를 사용하였다.

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters() , lr=0.01)
```

Loss function으로는 CrossEntropy를 사용하였고, Optimizer은 pyTorch의 Adam(learning rate 0.01)을 사용하여 학습을 진행하였다.

```
for phase in ['train', 'val']:
    if phase == 'train':
        model.train()
    else:
        model.eval()

    epoch_loss = 0.0
    epoch_corrects = 0

    if(epoch == 0) and (phase == 'train'):
        continue

    optimizer.zero_grad()

    with torch.set_grad_enabled(phase=='train'):
        if(phase == 'train'):
            output = model(x_train)
            loss = criterion(output, y_train)
            loss.backward()
            optimizer.step()

            pred = torch.max(output, 1)[1].eq(y_train).sum()
            acc = (pred * 100.0 / len(x_train)).cpu()
            print('{} Loss: {:f} Acc: {:f}'.format(phase, loss.item(), acc.numpy()))

        else:
            output = model(x_test)
            loss = criterion(output, y_test)
            loss_values.append(loss.item())

            pred = torch.max(output, 1)[1].eq(y_test).sum()
            acc = (pred * 100.0 / len(x_test)).cpu()
            print('{} Loss: {:f} Acc: {:f}'.format(phase, loss.item(), acc.numpy()))
            early_stopping(loss.item(), model)

            if early_stopping.early_stop:
                print("Early stopping")
                flag = 1
                break
```

학습 시 train과 validation을 함께 진행하여 training set으로 학습한 parameters로 validation set의 Accuracy를 확인하였다. Validation을 진행할 때는 parameters를 업데이트 하지 않고 training때와 동일한 식으로 정확도를 계산하여 training과 validation의 정확도를 비교하였다.

```
--------------------------
train Loss: 0.012089 Acc: 99.484352
val Loss: 0.049148 Acc: 98.342537
Validation loss decreased (0.050289 --> 0.049148).  Saving model ...
Epoch 9/100
--------------------------
train Loss: 0.005919 Acc: 99.484352
val Loss: 0.058324 Acc: 99.005524
EarlyStopping counter: 1 out of 20
Epoch 10/100
--------------------------
```

그리고 overtraining으로 인한 과적합을 방지하기 위해 validation error가 더 이상 감소하지 않을 때를 기준으로 EarlyStopping을 적용하였다. Validation error가 더 이상 감소하지 않는 시점을 기준으로 이후 20번의 patience를 가지고, 이때에도 감소하지 않는다면 정지하도록 하였다.

4. 성능 평가

- Accuracy

```
pred = torch.max(output, 1)[1].eq(y_test).sum()
acc = (pred * 100.0 / len(x_test)).cpu()
```

```
model.eval()
x_test = x_test.cuda()
y_test = y_test.cuda()
with torch.no_grad():
    y_pred = model(x_test)
    loss = criterion(y_pred, y_test)
    pred = torch.max(y_pred, 1)[1].eq(y_test).sum()
    print ("Accuracy : {}%".format(100*pred/len(x_test)))
loss
```

```
Accuracy : 98.34253692626953%
tensor(0.0491, device='cuda:0')
```

Expected value와 predicted value를 비교하여 같은 경우의 개수를 전체 set의 개수로 나누어 accuracy를 계산하였다.

```
model.load_state_dict(torch.load('../input/loading/tfidf40000_state.pt'))

model.eval()
x_test = x_test.cuda()
y_test = y_test.cuda()
with torch.no_grad():
    y_pred = model(x_test)
    loss = criterion(y_pred, y_test)
    pred = torch.max(y_pred, 1)[1].eq(y_test).sum()
    print ("Accuracy : {}%".format(100*pred/len(x_test)))
loss
```

```
Accuracy : 98.67402648925781%
tensor(0.0272, device='cuda:0')
```

Kaggle에 제출하여 가장 높은 점수를 받았을 때의 validation accuracy는 98.67였다.

- f1 score, accuracy_score, precision_score, recall_score, confusion_matrix

```
# f1 score, accuracy_score, precision_score, recall_score, confusion_matrix
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, confusion_matrix
print("f1_score : {}" .format(f1_score(y_test.cpu(), torch.max(y_pred, 1)[1].cpu())))
print("accuracy_score : {}" .format(accuracy_score(y_test.cpu(), torch.max(y_pred, 1)[1].cpu())))
print("precision_score : {}" .format(precision_score(y_test.cpu(), torch.max(y_pred, 1)[1].cpu())))
print("recall_score : {}" .format(recall_score(y_test.cpu(), torch.max(y_pred, 1)[1].cpu())))
print("confusion_matrix : {}" .format(confusion_matrix(y_test.cpu(), torch.max(y_pred, 1)[1].cpu())))

from sklearn.metrics import classification_report
print(classification_report(y_test.cpu(), torch.max(y_pred, 1)[1].cpu()))
```

```
f1_score : 0.9778597785977861
accuracy_score : 0.9867403314917127
precision_score : 1.0
recall_score : 0.9566787003610109
confusion_matrix : [[628   0]
 [ 12 265]]
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       628
           1       1.00      0.96      0.98       277

    accuracy                           0.99       905
   macro avg       0.99      0.98      0.98       905
weighted avg       0.99      0.99      0.99       905
```

sklearn.metrics를 사용하여 validation set에서의 성능지표를 구하여 성능을 평가하였다. y_test는 validation set의 expected value이고, torch.max(y_pred, 1)[1]은 predicted value이다.

Confusion matrix를 보면 positive를 negative로 예측한 것은 없었으나 negative를 positive로 잘못 예측한 경우가 있었다.

그리고 validation set을 바꾸어 여러 번 validation을 진행하였을 때 precision score은 대부분 1.0 이었다. 이는 positive라고 predict했을 때 실제로도 positive인 것으로 이에 대해서는 예측을 잘 하는 것을 볼 수 있다.

5. 결과 분석

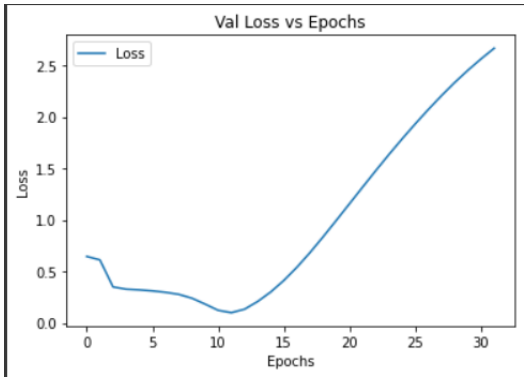
- 데이터 정제

기존에 데이터를 정제할 때 Vectorizer에 ngram을 적용하지 않았지만 두 개의 단어가 서로 연관되어 함께 등장하는 경우를 생각하여 ngram(1,2)를 추가하였다. 이때 결과가 더 좋게 나온 것을 확인할 수 있다.

Ngram을 적용하지 않으면 max_feature (단어 수)는 최대 30111개였다.

그러나 ngram(1,3)으로 1개-3개의 단어를 연관지어 학습한 결과 더 좋지 않은 결과를 확인하였다. Validation accuracy는 가장 높았지만 최종 결과가 낮은 것으로 보아 3개의 단어가 연관지어서 등장하는 빈도가 낮을 것으로 예측하였다.

- 학습 과정



Validation loss와 epochs사이의 관계를 시각화하였다. 기존에는 early stopping을 적용하지 않고 20번의 epochs를 사용하였는데 모델을 학습시킬 때마다 정확도가 매우 변화하였고 성능이 크게 개선되지 않았다. 그러나 11번째 Epochs에서 validation loss가 가장 낮은 것을 확인하였고 이때를 기준으로 early stopping을 한 결과 더 좋은 성능을 낼 수 있었다.

onground_30111feature_earlystoppingER20_trainval_cuda_9701.csv 2 months ago by Onground 이전과 동일조건 + early stopping (loss 0.20정도) 8에포크	0.98148
onground_30111feature_20epochs_trainval_cuda_9776.csv 2 months ago by Onground 30111 feature, 로지스틱회귀, trainval 함께 cuda 97.76 acc val	0.98297
onground.csv 2 months ago by Onground 로지스틱회귀 20000개 feature with cuda 20에포크 train + val 동시에 시도. val 점수 98.12점	0.93725
onground.csv 2 months ago by Onground 로지스틱회귀 + ReLU with cuda 에포크 100 feature 20000개 test acc 98.8	0.96356

그리고 max_feature (단어 수)를 20000개에서 최대개수인 30111개로 증가시킨 결과 점수가 증가한 것을 볼 수 있었다. 이후 ngram을 사용하였기 때문에 연관 단어를 추가한 40000개의 max_feature을 사용하여 학습시켰다.

Validation Accuracy와 f1_score이 높은 모델이 실제 Kaggle의 test set에서도 좋은 결과로 이어질 것이라고 생각했지만 그렇지 않았던 것으로 보아 validation set을 기준으로도 과적합이 되었다고 예측하였다.