멀티캠퍼스
딥러닝 기반 AI엔지니어링

인터넷 악성 댓글 분류 필터

3조 욕하지말아조

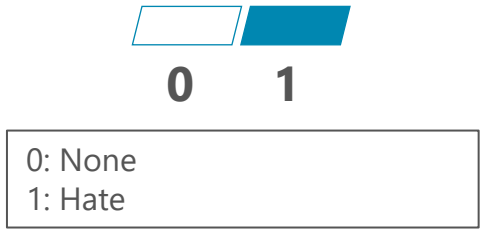
조윤석, 조원우, 이동진, 안애솔

Elevator Pitch

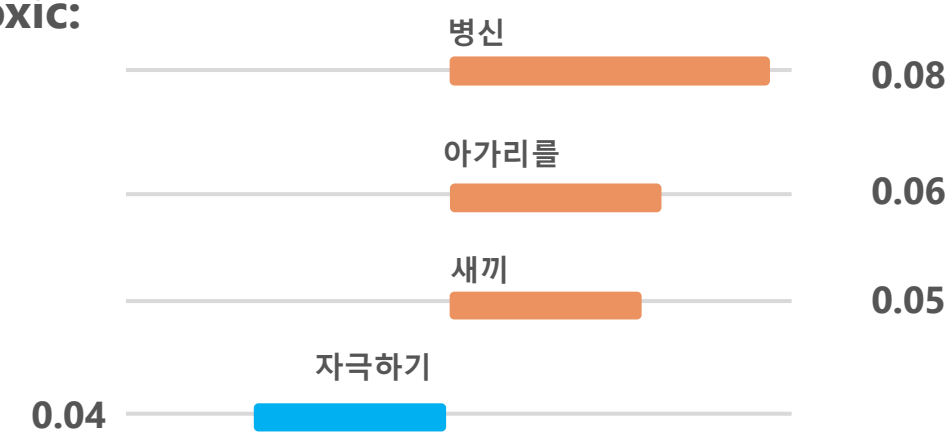
Comment:

정 그리 민감하고 자극하기 뭐하면 그냥 아가리를 쳐 닫고 있든가실리도 없고 명분도 없고 그냥 병신 새끼

Predict:



Toxic:



Text with highlighted words:

정 그리 민감하고 자극하기 뭐하면 그냥 아가리를 쳐 닫고 있든가실리도 없고 명분도 없고 그냥 병신 새끼

필터 처리 후:

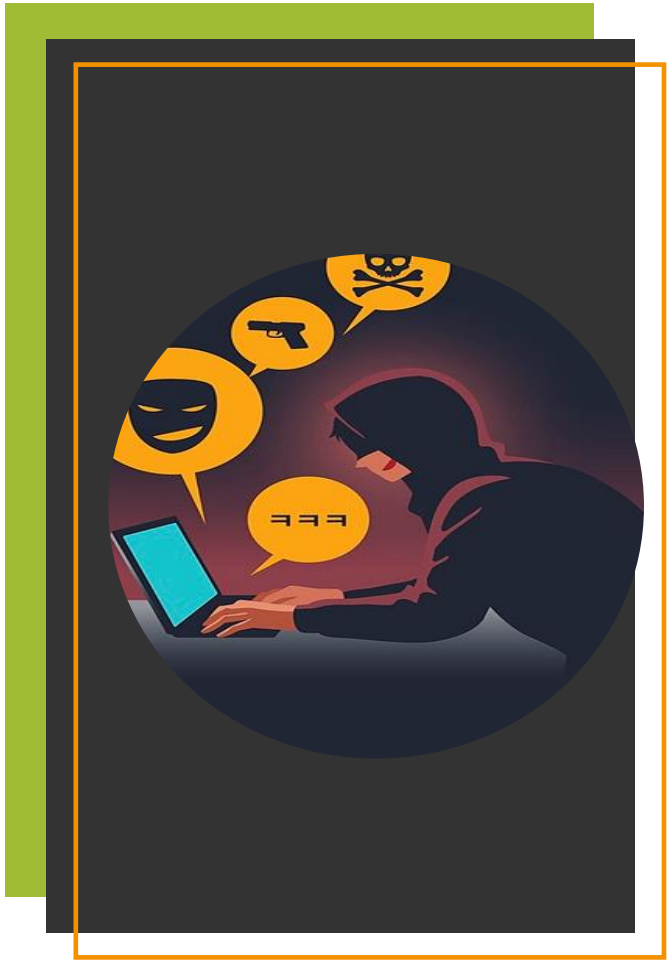
정 그리 민감하고 자극하기 뭐하면 그냥 ** * ** 있든가실리도 없고 *** 없고 그냥 ** **

원본:

Content

Ⅰ. 프로젝트 배경

Team **욕하지말아조**



관심도

이번 프로젝트의 주제가 텍스트에 관한 것인 만큼, 우리가 항상 접하는 댓글에 대해 분석해보고 싶었다.
그 중 많은 사람들이 상처를 받는 악성 댓글을 분류하는 소위 '악성 댓글 분류 필터'를 만드는 것을 목표로 했다.

사회현상

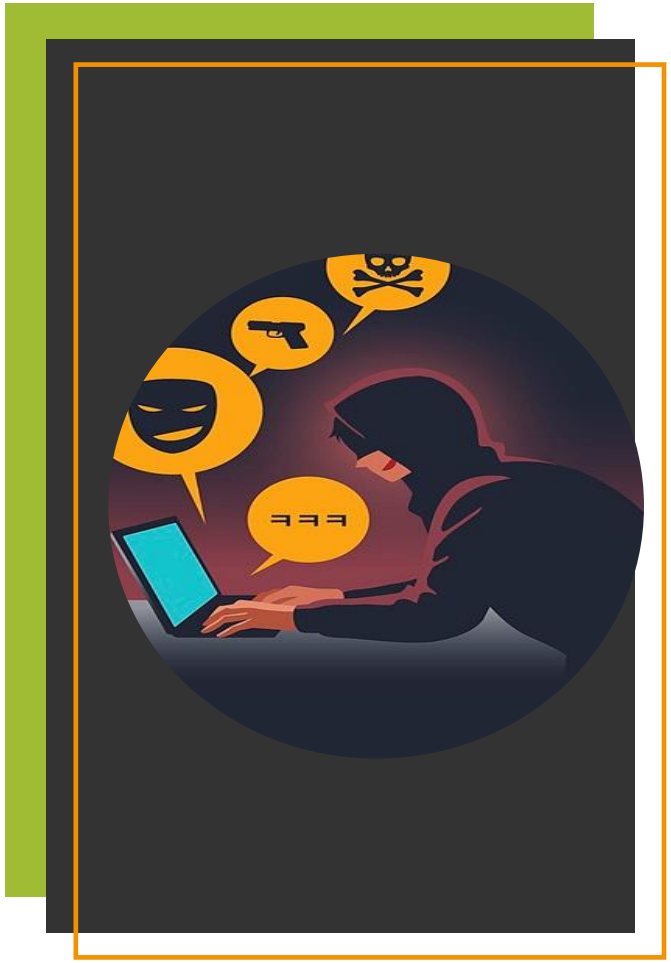
온라인 소통의 영향력이 그 어느 때보다 커진 현대 사회에서 중요하게 다루어 지는 이슈이다.

데이터셋

선례 연구들을 통해 데이터를 확보하기 용이했다.

Ⅱ. 프로젝트 팀 구성 및 역할

Team **욕하지말아조**



Leader

조윤석

형태소분석기: Khaiii, Okt
알고리즘: LSTM
데이터 전처리, tf-idf 활용 단어 중요도 분석, WordCloud

team

조원우

형태소 분석기: Mecab
알고리즘: simpleRNN, CNN1D
데이터 전처리

team

이동진

알고리즘: NB-SVM
데이터 전처리

team

안애솔

데이터 전처리
프로젝트 배경 및 PPT 제작

Ⅲ. 프로젝트 수행절차 및 방법

Team **욕하지말아조**

구분	기간	활동	비고
사전 기획	• 10/29(목) ~ 11/01(일)	• 프로젝트 기획 및 주제 선정 • 기획안 작성	• 아이디어 선정 • 데이터 확보(캐글)
개발	• 11/02(월) ~ 11/10(화)	• 데이터 전처리(형태소 별) • 모델 선정 및 프로젝트 진행	• 형태소 분석기 Khaiii, Okt, Mecab • 알고리즘 CNN, RNN, LSTM, NB-SVM, Tf-Idf
수정/보완	• 11/10(화) ~ 11/11(수)	• 미비한 부분 보완 및 최종보고서 작성	
총 개발기간	• 10/29(목) ~ 11/11(수)(2주)	• 악성댓글 분류 필터 완성!	

Ⅳ. 프로젝트 수행 결과

악성 댓글 분류 필터 제작 프로세스

A 전처리

댓글 중복 여부, Null 값 제거, 한글과 공백 이외의 것들을 제외함

B 토큰화

['정말', '재밌다', '연기', '좋고', '디카프리오', '짱']

C 정수인코딩

[[21, 17, 54, 8, 3090, 130]]

D 패딩

```
array([   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,
         21,   17,   54,    8, 3090,  130])
```

E 알고리즘을 통한 분류

```
정말 재밌다 연기도 좋고 디카프리오 짱: 99.99% ++.
```

Ⅳ. 프로젝트 수행 결과

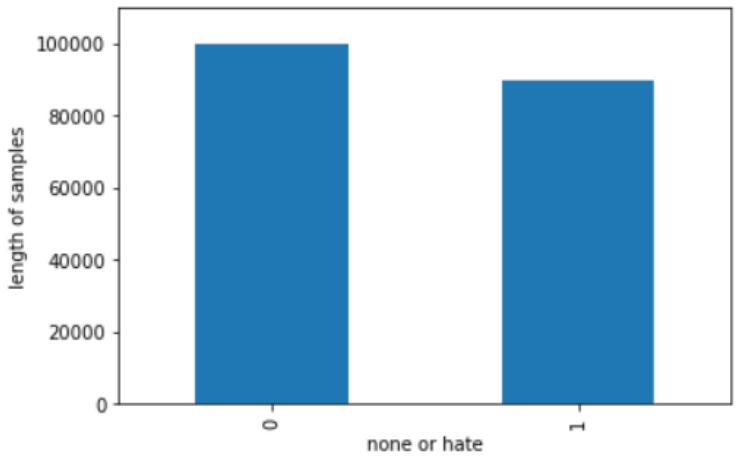
데이터 소개 및 전처리

전처리

댓글 중복 여부, Null 값 제거, 한글과 공백 이외의 것들을 제외함

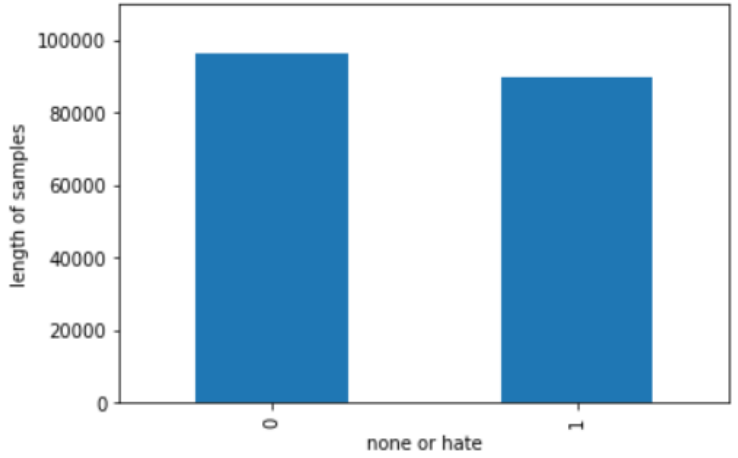
0: None
1: Hate

train_data = 150,000
Test_data = 36,000



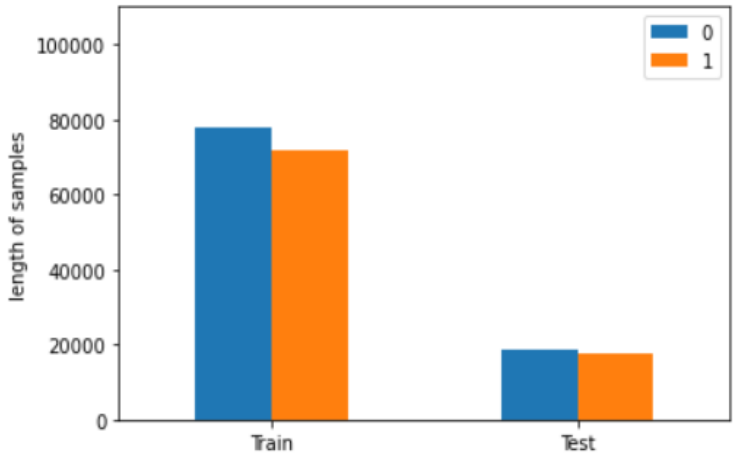
Raw data

0: 100,000개
1: 90,000개
총 190,000개



전처리 후 data

0: 97,223개
1: 89,920개
총 187,143개



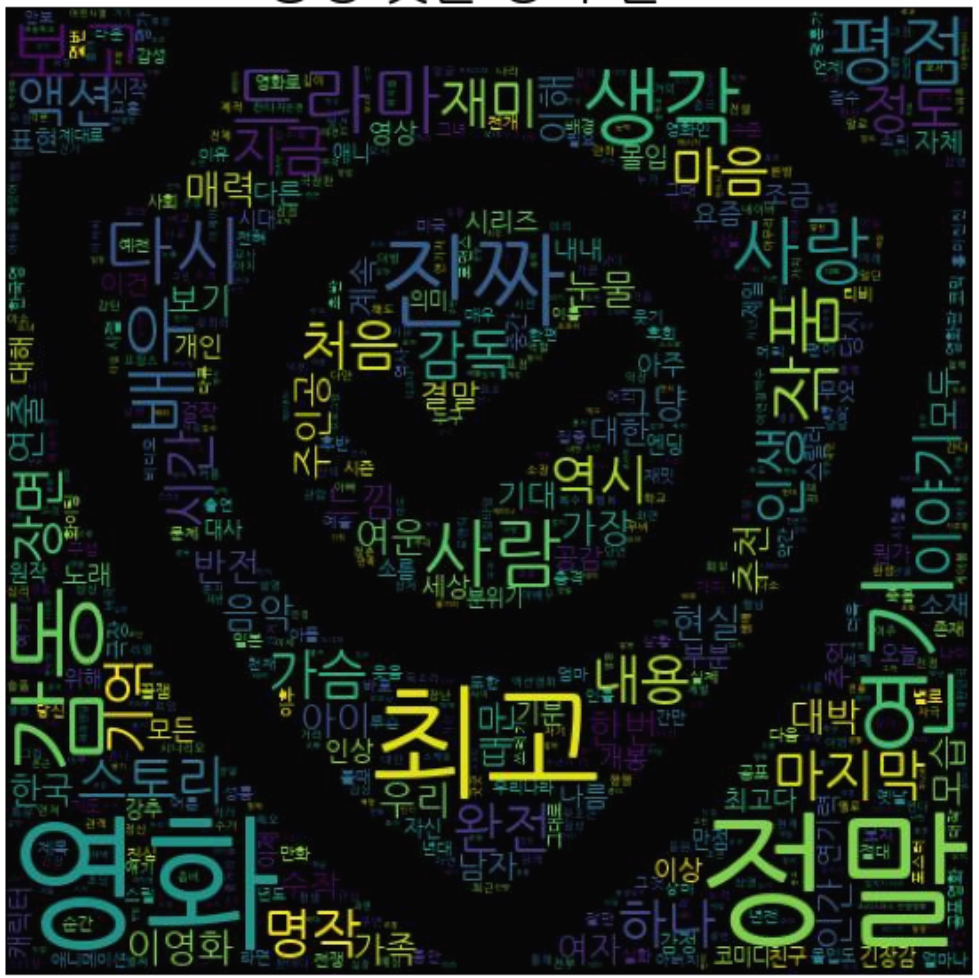
Train/Test data

	0	1
Train	78008	71977
Test	18550	17571

참고자료> 댓글 data 명사 빈도 Word Cloud

부정 댓글 명사 빈도

긍정 댓글 명사 빈도



Ⅳ. 프로젝트 수행 결과

형태소 분석기 소개

토큰화

3가지 형태소 분석기를 사용함
Khaiii / Okt / Mecab

Khaiii 형태소 분석기 **Mecab 형태소 분석기**

'중국 공산당 보는것 같다'

['중국', '공산당', '보', '것', '같', '다']

Okt 형태소 분석기

'중국 공산당 보는것 같다'

['중국', '공산당', '보다', '같다']

이러한 3가지 형태소 분석기로 전처리한 데이터를 분석했다

Ⅳ. 프로젝트 수행 결과

데이터 분석

토큰화

3가지 형태소 분석기를 사용함
Khaiii / Okt / Mecab

*희귀단어: 등장 빈도가 2번 이하인 단어

	Khaiii	Okt	Mecab
단어집합 (vocabulary) 크기	129,184	56,246	62,345
총 *희귀 단어의 수	100,312	31,118	34,398
단어 집합 내에서 희귀단어 비율	77.65%	55.32%	55.18%
전체 등장 빈도 중 희귀 단어 빈도 비율	4.68%	2.11%	1.91%
필터 된 단어 집합 크기	28,874	25,130	27,949

Ⅳ. 프로젝트 수행 결과

데이터 분석

토큰화

3가지 형태소 분석기를 사용함
Khaiii / Okt / Mecab

*희귀단어: 등장 빈도가 2번 이하인 단어

	Khaiii	Okt	Mecab
전체 등장 빈도 중 희귀 단어 빈도 비율	4.68%	2.11%	1.91%
필터 된 단어 집합 크기	28,874	25,130	27,949

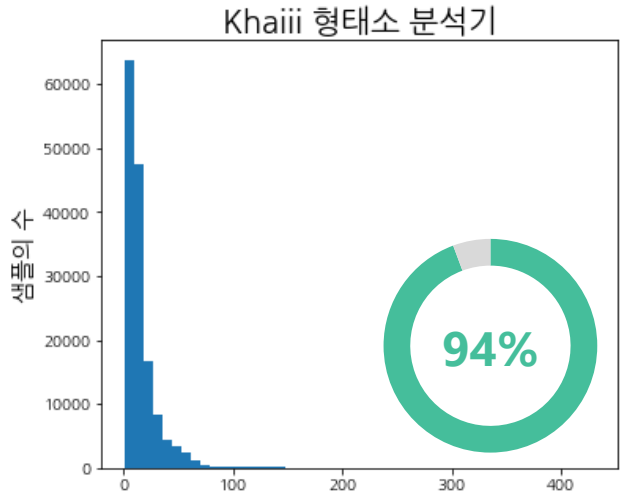
- ✓ 각 단어의 총 집합에서 희귀 단어의 등장 빈도 비율이 낮음
- ✓ 그래서 제거하고자 했고, 그 결과 단어 집합의 크기를 평균 27,000개까지 정리함

Ⅳ. 프로젝트 수행 결과

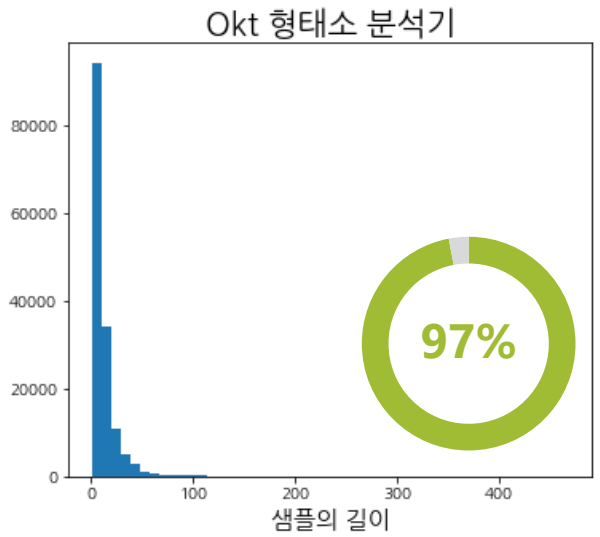
데이터 분석

패딩

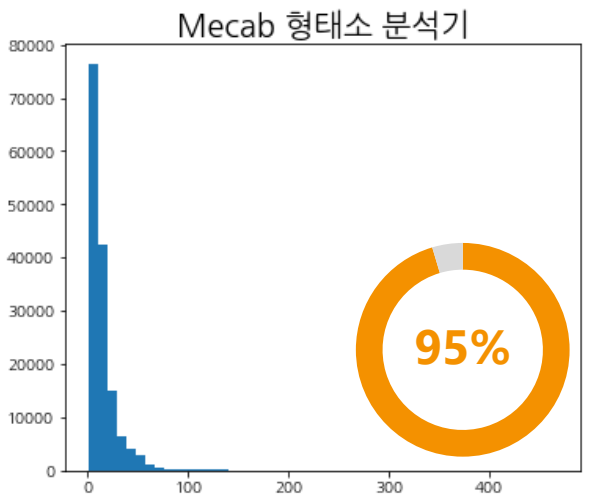
3가지 형태소 분석기를 사용함
Khaiii / Okt / Mecab



리뷰의 최대 길이: 431
리뷰의 평균 길이: 16.29
길이가 45 이하인 샘플의 비율: 94.33%



리뷰의 최대 길이: 468
리뷰의 평균 길이: 12.07
길이가 45 이하인 샘플의 비율: 97.85%



리뷰의 최대 길이: 468
리뷰의 평균 길이: 15.18
길이가 45 이하인 샘플의 비율: 95.39%

95.42%

✓ Comment의 평균 길이는 15이고, 전체 샘플 중 길이가 45이하인 샘플만 사용하기로 함
✓ Comment 길이를 45로 했을 때 **Khaiii는 94% / Okt는 97% / Mecab은 95%**의 비중을 차지한다.

Ⅳ. 프로젝트 수행 결과

알고리즘

알고리즘을 통한 분류

4가지 알고리즘을 사용함
CNN / **RNN** / **LSTM** / **NB-SVM**

•CNN(Convolution Neural Network)
주로 이미지 처리에 쓰이지만 자연어 처리에도 좋은 성능을 보임.

```
Model: "sequential_42"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_41 (Embedding)     (None, 50, 100)           2513000
_________________________________________________________________
conv1d_48 (Conv1D)           (None, 46, 50)            25050
_________________________________________________________________
max_pooling1d_29 (MaxPooling (None, 23, 50)            0
_________________________________________________________________
flatten_27 (Flatten)         (None, 1150)              0
_________________________________________________________________
dense_49 (Dense)             (None, 10)                11510
_________________________________________________________________
dense_50 (Dense)             (None, 1)                 11
=================================================================
Total params: 2,549,571
Trainable params: 2,549,571
Non-trainable params: 0
```

사용예시

•RNN(Recurrent Neural Networks)
히든 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는 인공신경망의 한 종류.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, None, 100)         2887400
_________________________________________________________________
simple_rnn_5 (SimpleRNN)     (None, 128)               29312
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 2,916,841
Trainable params: 2,916,841
Non-trainable params: 0
_________________________________________________________________
```

사용예시

Ⅳ. 프로젝트 수행 결과

알고리즘

알고리즘을 통한 분류

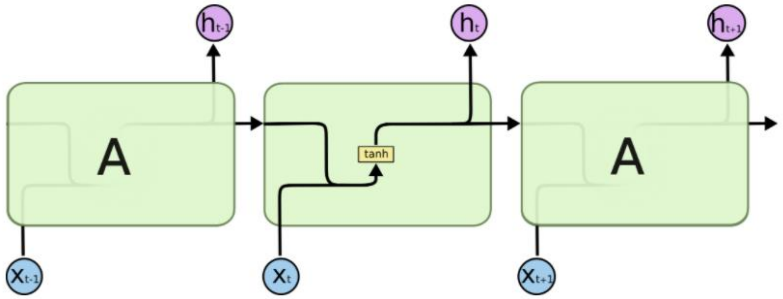
4가지 알고리즘을 사용함
CNN / RNN / LSTM / NB-SVM

•LSTM(Long Short-Term Memory)
RNN의 단점인 긴 시퀀스에서 성능을 발휘하지 못하는 점을 개선한 알고리즘

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, None, 100)	2513000
lstm (LSTM)	(None, 128)	117248
dense (Dense)	(None, 1)	129

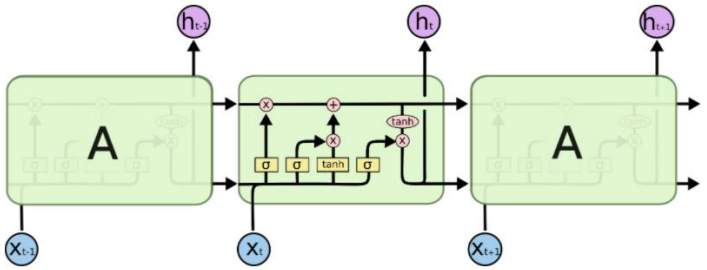
Total params: 2,630,377
Trainable params: 2,630,377
Non-trainable params: 0

사용예시



simple RNN(vanila RNN)

simple RNN에
input gate, forget gate, output gate를
추가하여 필요/불필요한 정보를
나누어 기억함



LSTM

출처 :http://colah.github.io/posts/2015-08-Understanding-LSTMs/

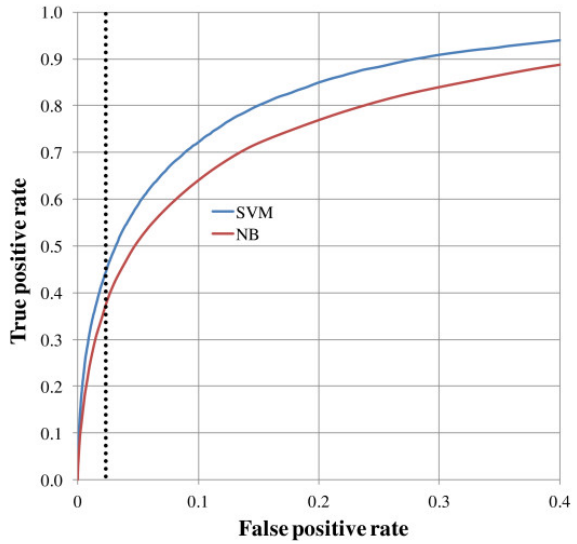
Ⅳ. 프로젝트 수행 결과

알고리즘

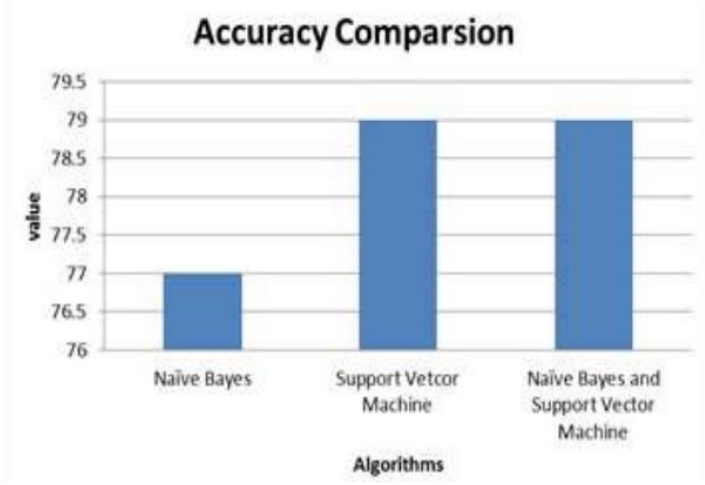
알고리즘을 통한 분류

4가지 알고리즘을 사용함
CNN / RNN / LSTM / NB-SVM

- ✓ NB 와 SVM 은 각각에 대한 커널 함수 에 대한 옵션 가지고 있고 매개 변수 최적화에 민감하다
- ✓ NB-SVM 주로 텍스트 분류 및 감성 분석 연구방법 사용됩니다



NB 와 SVM 비교 분석



나이브베이즈 서포트벡터머신을 같이 사용했을 때 정확도가 더 높아서 함께 사용된다.

Ⅳ. 프로젝트 수행 결과

알고리즘

알고리즘을 통한 분류

4가지 알고리즘을 사용함
CNN / RNN / LSTM / NB-SVM

```
print(confusion_matrix(pred_khaiii_NB_SVM, yTest))
print(classification_report(pred_khaiii_NB_SVM, yTest))
```

```
[[16782   898]
 [ 1768 16673]]
              precision    recall  f1-score   support

           0       0.90      0.95      0.93     17680
           1       0.95      0.90      0.93     18441

    accuracy                           0.93     36121
   macro avg       0.93      0.93      0.93     36121
weighted avg       0.93      0.93      0.93     36121
```

```
print(confusion_matrix(pred_mecab_NB_SVM, yTest))
print(classification_report(pred_mecab_NB_SVM, yTest))
```

```
[[17141   798]
 [ 1409 16773]]
              precision    recall  f1-score   support

           0       0.92      0.96      0.94     17939
           1       0.95      0.92      0.94     18182

    accuracy                           0.94     36121
   macro avg       0.94      0.94      0.94     36121
weighted avg       0.94      0.94      0.94     36121
```

```
print(confusion_matrix(pred_okt_NB_SVM, yTest))
print(classification_report(pred_okt_NB_SVM, yTest))
```

```
[[17011   906]
 [ 1539 16665]]
              precision    recall  f1-score   support

           0       0.92      0.95      0.93     17917
           1       0.95      0.92      0.93     18204

    accuracy                           0.93     36121
   macro avg       0.93      0.93      0.93     36121
weighted avg       0.93      0.93      0.93     36121
```

NB - SVM

>

```
print(confusion_matrix(pred_khaiii_NB, yTest))
print(classification_report(pred_khaiii_NB, yTest))
```

```
[[17520  2429]
 [ 1030 15142]]
              precision    recall  f1-score   support

           0       0.94      0.88      0.91     19949
           1       0.86      0.94      0.90     16172

    accuracy                           0.90     36121
   macro avg       0.90      0.91      0.90     36121
weighted avg       0.91      0.90      0.90     36121
```

```
print(confusion_matrix(pred_mecab_NB, yTest))
print(classification_report(pred_mecab_NB, yTest))
```

```
[[17759  2027]
 [  791 15544]]
              precision    recall  f1-score   support

           0       0.96      0.90      0.93     19786
           1       0.88      0.95      0.92     16335

    accuracy                           0.92     36121
   macro avg       0.92      0.92      0.92     36121
weighted avg       0.92      0.92      0.92     36121
```

```
print(confusion_matrix(pred_okt_NB, yTest))
print(classification_report(pred_okt_NB, yTest))
```

```
[[17737  2332]
 [  813 15239]]
              precision    recall  f1-score   support

           0       0.96      0.88      0.92     20069
           1       0.87      0.95      0.91     16052

    accuracy                           0.91     36121
   macro avg       0.91      0.92      0.91     36121
weighted avg       0.92      0.91      0.91     36121
```

NB

Ⅳ. 프로젝트 수행 결과

알고리즘

알고리즘을 통한 분류

4가지 알고리즘을 사용함
CNN / RNN / LSTM / **NB-SVM**

Naive_bayes

```
from sklearn.naive_bayes import CategoricalNB
```

```
model=CategoricalNB()
model.fit(xTrain, yTrain)
```

```
CategoricalNB()
```

```
pred_NB=model.predict(xTest)
```

```
test['pred_NB']=pred_NB
(test['label']==test['pred_NB']).sum()/test.shape[0]
```

```
0.893
```

NB-SVM

```
def pr(y_i, y): # NB 함수 정의
    p = train_dtm[y==y_i].sum(0)
    return (p+1) / ((y==y_i).sum()+1)
```

```
def get_mdl(y):
    y = y.values
    r = np.log(pr(1,y) / pr(0,y)) # NB 함수를 이용한 2진화
    m = LogisticRegression(C = 0.1, dual = True) # 로지스틱 회귀
    x_nb = train_dtm.multiply(r)
    return m.fit(x_nb, y), r
```

```
m,r = get_mdl(train['label']) #ㅍ
preds=m.predict_proba(test_dtm.multiply(r)) #예측
```

```
pred_NB_SVM=[]
for i in preds:
    pred_NB_SVM.append(i.argmax())

test['pred_NB_SVM']=pred_NB_SVM
(test['label']==test['pred_NB_SVM']).sum()/test.shape[0]
```

```
0.919
```

Ⅳ. 프로젝트 수행 결과

Confusion Matrix Visualization

Group names, Counts and Percentages

Okt – CNN1D

True – Neg 17,312 47.93%	False – Pos 1,238 3.42%
False – Neg 831 2.30%	True – Pos 16,740 46.34%

Mecab – RNN1D

True – Neg 17,489 48.42%	False – Pos 1,061 2.94%
False – Neg 893 2.47%	True – Pos 16,678 46.17%

Mecab – LSTM

True – Neg 17,623 48.79%	False – Pos 927 2.57%
False – Neg 784 2.17%	True – Pos 16,787 46.47%

Ⅳ. 프로젝트 수행 결과

Classification Report

형태소 분석기 평가 분석표

Accuracy, precision, recall, f1-score

Items	CNN1D			RNN			TFIDF-NB			LSTM		
	Khaiii	Mecab	Okt	Khaiii	Mecab	Okt	Khaiii	Mecab	Okt	Khaiii	Mecab	Okt
accuracy	0.9212	0.9323	0.9337	0.8884	0.9459	0.9315	0.9340	0.9491	0.9474	0.9399	0.9528	0.9479
precision	0.9201	0.9329	0.9336	0.8889	0.9458	0.9316	0.9351	0.9494	0.9475	0.9399	0.9527	0.9478
recall	0.9203	0.9318	0.9338	0.8890	0.9459	0.9319	0.9334	0.9487	0.9472	0.9399	0.9529	0.9480
F1-score	0.9201	0.9326	0.9337	0.8884	0.9459	0.9314	0.9338	0.9490	0.9473	0.9399	0.9528	0.9479

Ⅳ. 프로젝트 수행 결과

Classification Report

형태소 분석기 별 성능

Mecab > Okt > Khaiii

알고리즘 별 성능

LSTM > RNN > CNN1D > NB-SVM

형태소 분석기 평가 분석표

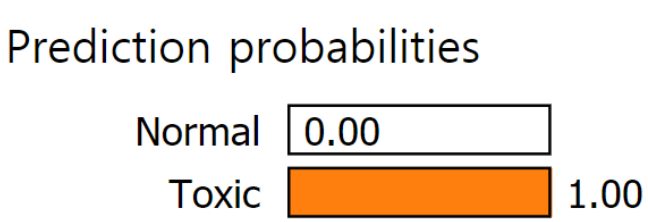
Accuracy, precision, recall, f1-score

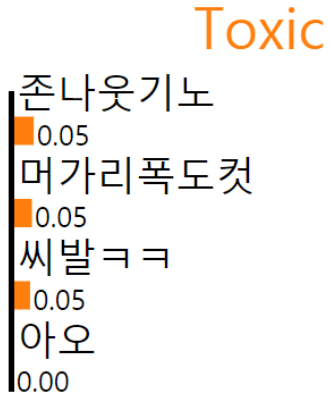
Items	CNN1D			RNN			TFIDF-NB			**LSTM**		
	Khaiii	**Mecab**	Okt	Khaiii	**Mecab**	Okt	Khaiii	**Mecab**	Okt	**Khaiii**	**Mecab**	**Okt**
accuracy	0.9212	**0.9323**	0.9337	0.8884	**0.9459**	0.9315	0.9340	**0.9491**	0.9474	**0.9399**	**0.9528**	**0.9479**
precision	0.9201	**0.9329**	0.9336	0.8889	**0.9458**	0.9316	0.9351	**0.9494**	0.9475	**0.9399**	**0.9527**	**0.9478**
recall	0.9203	**0.9318**	0.9338	0.8890	**0.9459**	0.9319	0.9334	**0.9487**	0.9472	**0.9399**	**0.9529**	**0.9480**
F1-score	0.9201	**0.9326**	0.9337	0.8884	**0.9459**	0.9314	0.9338	**0.9490**	0.9473	**0.9399**	**0.9528**	**0.9479**

Ⅳ. 프로젝트 수행 결과

악성 댓글 필터링

Lime





Text with highlighted words

머가리폭도컷 아오 씨발ㅋㅋ 존나웃기노

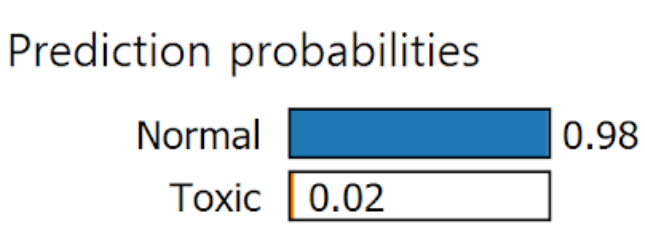
Filtered comment

해당 댓글은 블라인드 처리 되었습니다.

Ⅳ. 프로젝트 수행 결과

악성 댓글 필터링

Lime

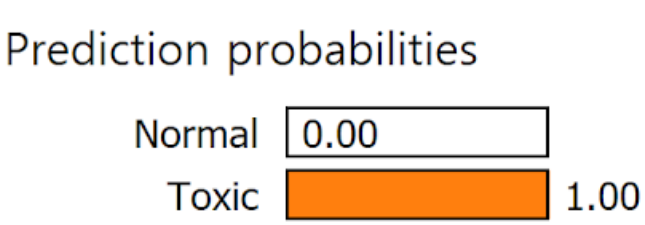


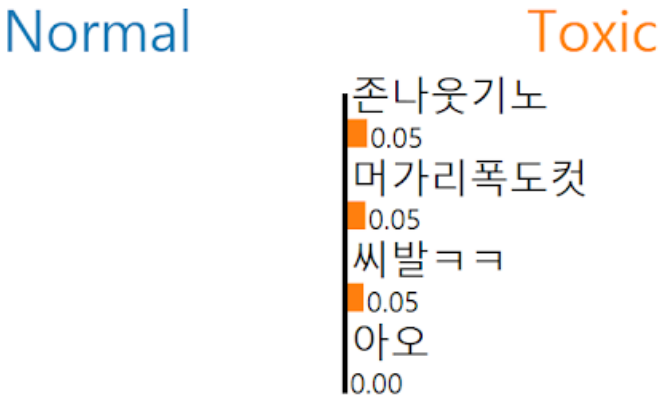
Text with highlighted words

완전 감동이다

Filtered comment

완전 감동이다





Text with highlighted words

머가리폭도컷 아오 씨발ㅋㅋ 존나웃기노

Filtered comment

****** 아오 **** *****

Ⅳ. 프로젝트 수행 결과

악성 댓글 필터링

Lime

Local
Interpretable
Model-agnostic
Explanations

✓ 로컬대리분석 기법 중 하나

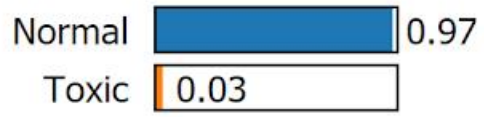
✓ 먼저 만든 분류기와 비슷한 예측결과를 만드는 설명 가능한 선형 분류 모델을 내부적으로 만든 후,
이 설명 가능한 분류기를 통해서 기존에 먼저 만들어 놓은 분류기 결과를 설명함

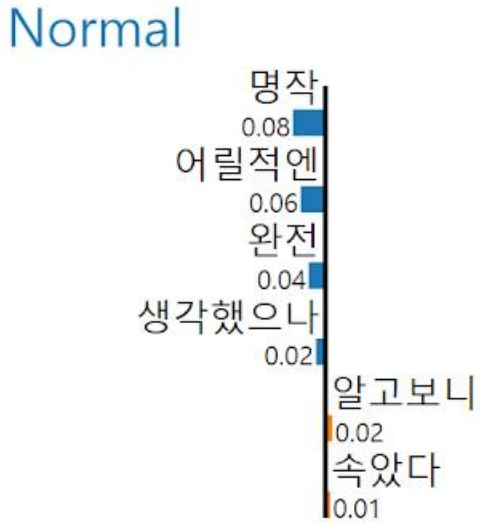
Ⅳ. 프로젝트 수행 결과

악성 댓글 필터링

Lime

Prediction probabilities





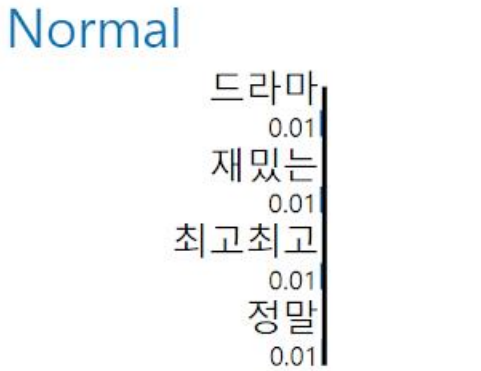
Text with highlighted words

어릴적엔 속았다 생각했으나 알고보니 완전 명작

Filtered comment

어릴적엔 속았다 생각했으나 알고보니 완전 명작

Prediction probabilities



Text with highlighted words

최고최고 정말 재밌는 드라마

Filtered comment

최고최고 정말 재밌는 드라마

Ⅴ. 느낀점

(1) 모델 설명력을 키우는 것이 관건이었다. 2가지 방법으로 해결하고자 했고, 기존 정확도 0.7에서 0.9까지 향상시켰다.

Khaiii,Okt + LSTM

```
[32]: loaded_model = load_model('LSTM_single_layer_model.h5')
      print("\n 테스트 정확도: %.4f" % (loaded_model.evaluate(X_test, y_test)[1]))

39521/39521 [==============================] - 7s 189us/sample - loss: 0.1415 - acc: 0.9446

 테스트 정확도: 0.9446
```

Mecab + simple RNN

```
1 loaded_model = load_model('best_model.h5')
2 print("\n 테스트 정확도: %.4f" % (loaded_model.evaluate(X_test, y_test)[1]))

1170/1170 [==============================] - 2s 2ms/step - loss: 0.1099 - acc: 0.9583

 테스트 정확도: 0.9583
```

Model accuracy

기존 알고리즘 정확도가 0.7 정도로 설명력이 부족하다고 판단, 2가지 방법으로 개선하고자 하였다.

1. **데이터의 양을 늘리는 것**
2. 다른 알고리즘을 사용하는 것

Data amount

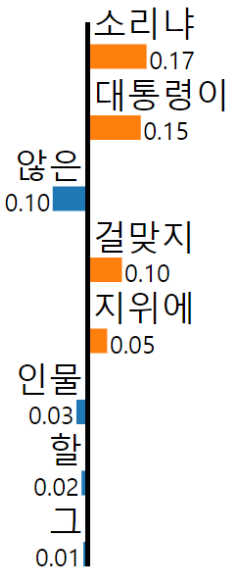
190,000

Model accuracy

95%

Ⅴ. 느낀점

(2) TF-IDF 모델의 한계



Text with highlighted words

대통령이 할 소리냐 그 지위에 걸맞지 않은 인물

Filtered comment

**** 할 *** 그 지위에 *** 않은 인물

TF-IDF

문서의 빈도수만 고려함

-> 단어의 의미, 문맥적 관계를 고려하지 못함

댓글을 자체적으로 필터링하는데 한계가 있다.

모델 성능

성능 면에서 LSTM이나 RNN을 이용한 모델들이 더 좋다.