Toxic Comment Classification Challenge

Member	㉧ 김보성
Notebook Link	https://www.kaggle.com/jagangupta/stop-the-s-toxic-comments-eda
Notebook Name	Stop the S@#$ - Toxic Comments EDA
Topic	NLP
Week	Week 4️⃣
추천	⭐⭐⭐⭐⭐

댓글의 '매운 맛'을 검출하는 대회.

NLP는 보통 전처리가 몹시 어려운 문제인 경우가 많은데, 올바른 전처리를 위해
올바른 EDA를 배워보자.

데이터셋 구성:

- toxic
- severe-toxic
- obscene (역겨운)
- threat
- insult
- identity_hate

데이터셋 레이블링 자체가 크라우드 소싱되어 있기에, 정확도가 어느정도 떨어질 가능성이 있다.
대회 중간에 새로운 테스트 데이터셋이 주최자들에 의해 만들어진 듯.

다음과 같은 순서로 데이터 분석이 이루어졌다.

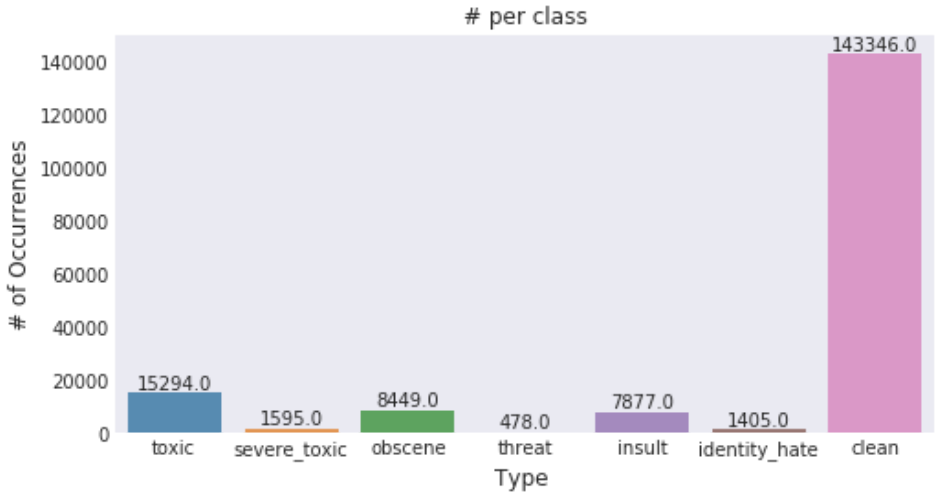
1. 데이터 셋 사이즈 확인

2. 일반적인 head, tail, column, rows 확인

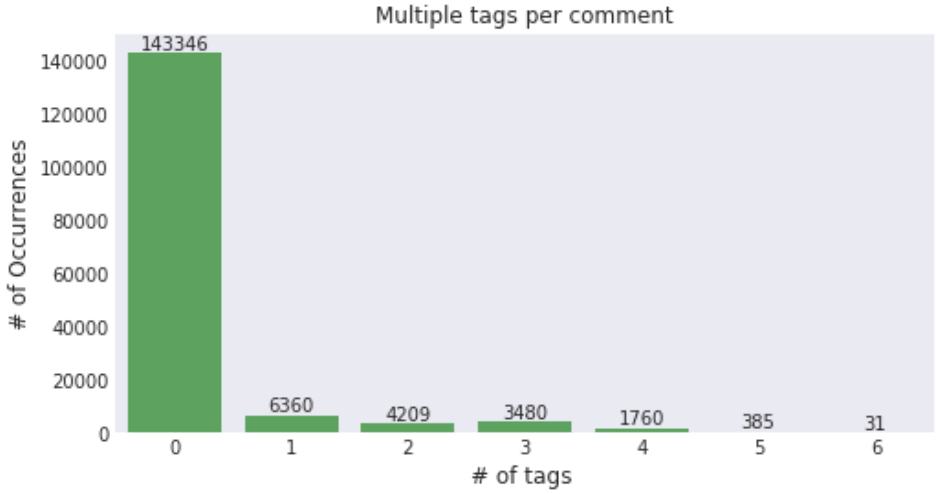
	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
159561	ffd2e85b07b3c7e4	"\nNo he did not, read it again (I would have ...	0	0	0	0	0	0
159562	ffd72e9766c09c97	"\n Auto guides and the motoring press are not...	0	0	0	0	0	0
159563	ffe029a7c79dc7fe	"\nplease identify what part of BLP applies be...	0	0	0	0	0	0
159564	ffe897e7f7182c90	Catalan independentism is the social movement ...	0	0	0	0	0	0
159565	ffe8b9316245be30	The numbers in parentheses are the additional ...	0	0	0	0	0	0
159566	ffe987279560d7ff	":::::And for the second time of asking, when ...	0	0	0	0	0	0
159567	ffea4adeee384e90	You should be ashamed of yourself \n\nThat is ...	0	0	0	0	0	0
159568	ffee36eab5c267c9	Spitzer \n\nUmm, theres no actual article for ...	0	0	0	0	0	0
159569	fff125370e4aaaf3	And it looks like it was actually you who put ...	0	0	0	0	0	0
159570	fff46fc426af1f9a	"\nAnd ... I really don't think you understand...	0	0	0	0	0	0

3. 클래스 분포 확인, 결측치 확인 및 'unknown' 대체
   : 이 시점에 클래스 불균형을 확인할 수 있었다. 해당 사실을 기반으로 이후 전처리에서 가중을 주어야 할듯.

```
Total comments =  159571
Total clean comments =  143346
Total tags = 35098
```



4. 클래스에 동시에 해당하는 댓글 확인



	severe_toxic		obscene		threat		insult		identity_hate	
severe_toxic	0	1	0	1	0	1	0	1	0	1
toxic										
0	144277	0	143754	523	144248	29	143744	533	144174	103
1	13699	1595	7368	7926	14845	449	7950	7344	13992	1302

'toxic' 클래스에 대한 다른 클래스들과의 크로스탭.

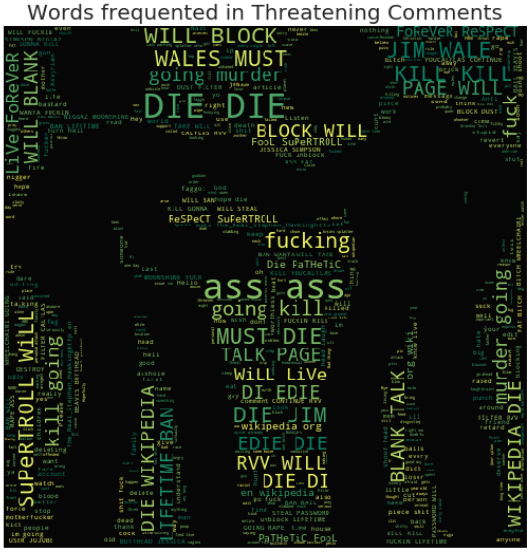
- 'severe toxic' 댓글은 항상 'toxic' 라벨 또한 가지고 있음.
- 다른 클래스들 또한, 'toxic' 클래스의 세부 세트로 파악해도 무방.
- 몇몇 댓글은 'ip 주소(개인 id 개념)' 을 가지고 있고, 이는 학습되기 너무 좋다(오버피팅 쉬움)

5. 자주 사용된 단어 시각화

Words frequented in Toxic Comments

Words frequented in Severe Toxic Comments

Words frequented in Threatening Comments



Words frequented in insult Comments

- 픽토그램을 베이스 이미지로, 그리고 픽토그램이 없는 곳에 wordcloud를 만드는 코드가 존재함. 중요한건 아니지만....?

6. Feature Engineering!

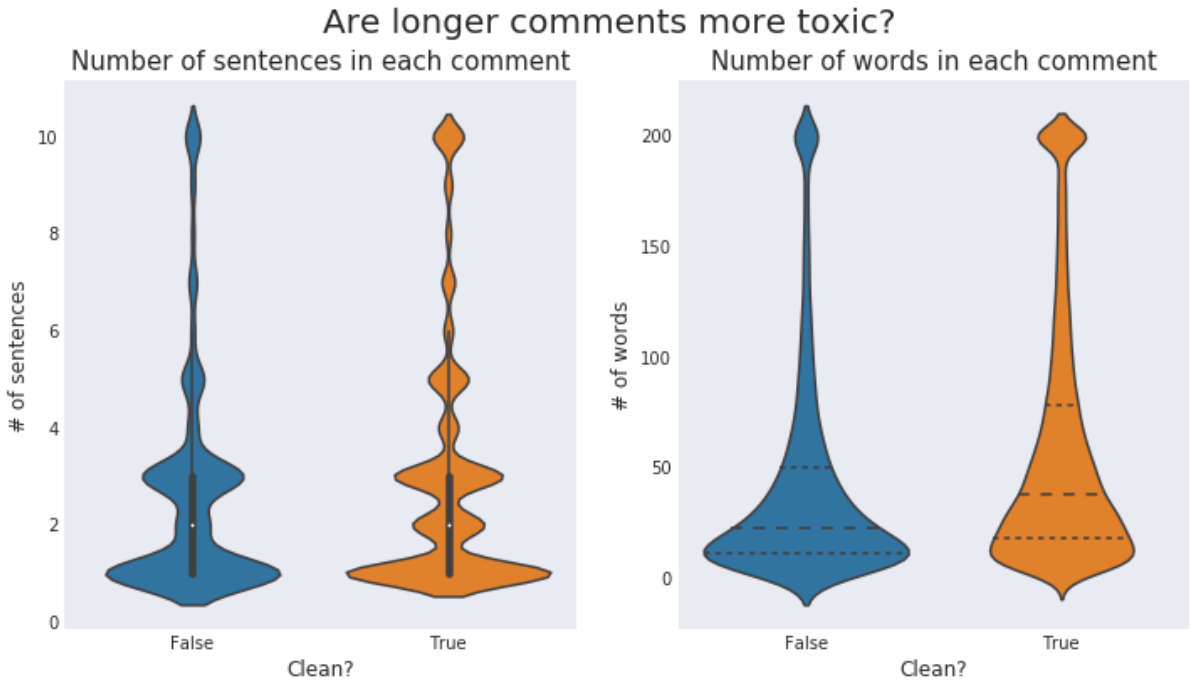
- 도움이 될 특성
  - 자주 사용된 단어 특성
    - 단순한 count?
    - 두 단어씩 묶어서 확인
    - 세 단어씩 묶어서 확인
  - Word2vec을 사용한 단어 간 거리 확인 (벡터화 되어 임베딩된 뒤 확인 가능할듯)
  - Sentiment score? (이후 나올 것 같다.)

- 도움이 될지 안될지 모르는 특성들
  - 문장/단어/고유어/글자/표현/대문자/불용어 의 수
  - 각 단어의 길이
- 쓰면 안되는 특성
  - 댓글에 언급된 ip
  - 댓글을 쓴 유저 id

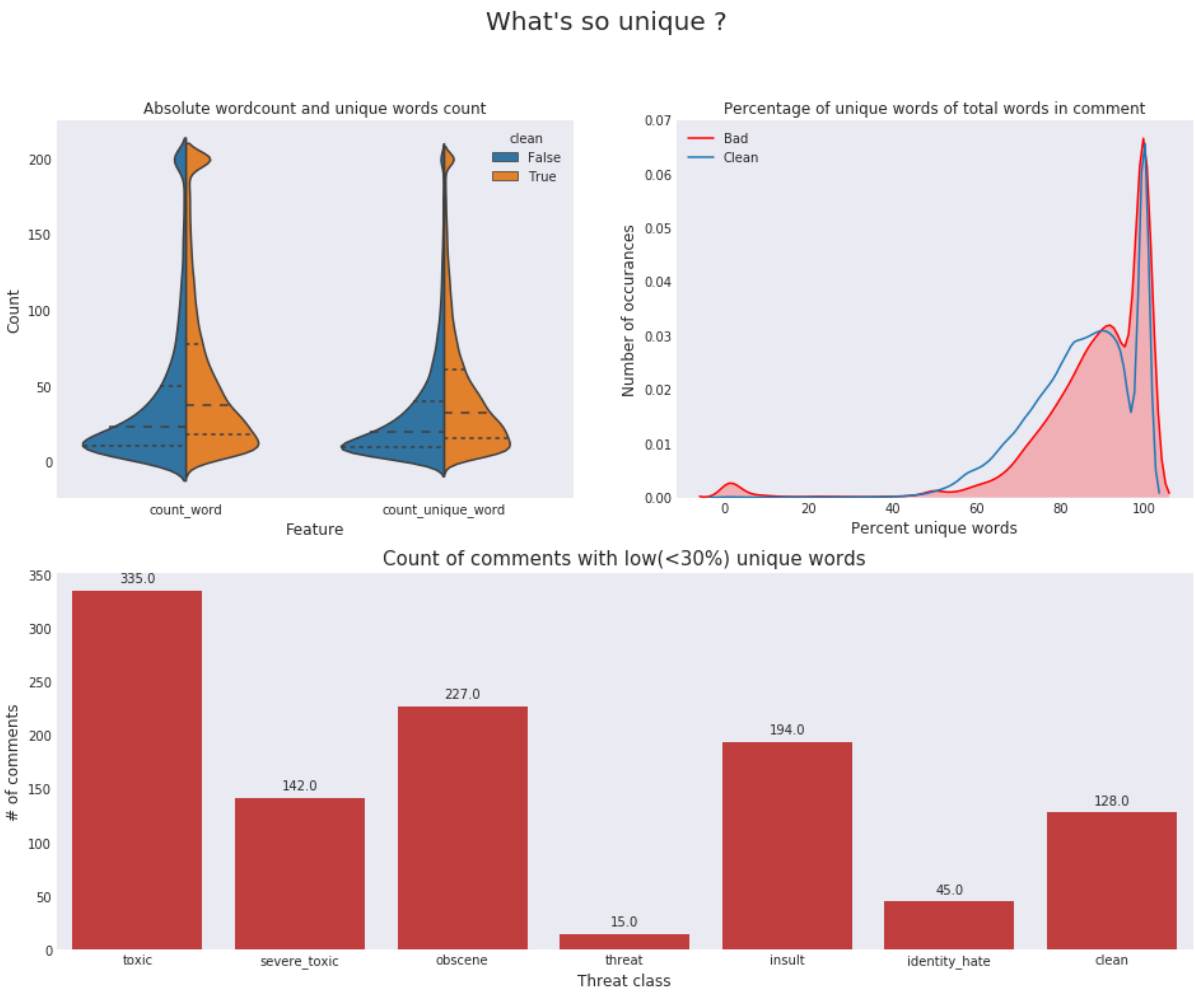
필자는 먼저 쓰면 안되는 특성과 도움이 될지 안될지 모르겠는 특성들 먼저 처리하고 말뭉치를 만들려고 하였는데,

1. 도움이 될 특성에는 깔-끔한 말뭉치에서 뽑아야 의미가 생기는 것들이 있다.

1. 데이터셋을 정돈하는 과정에서 도움이 될지 안될지 모를 특성들이 손실되는 정보를 어느정도 보상해줄 것으로 예상되기 때문



- 댓글에 단어가 많거나 문장이 길면 '덜 매운' 것을 확인할 수 있다.



- '고유어' : 한번만 쓰인 단어 로 생각해도 괜찮을듯.
- 두번째 차트에서, 고유어가 0~10퍼센트만 사용된 경우 거의 대부분이 나쁜 댓글이었다.

그럼 고유어가 0~10퍼센트 라는 이야기는?

'도배 댓글/복붙 댓글' 이 toxic할 확률이 높다는 것이다.

그런데 이 도배댓글이 또 문제가 되는 것이, 이게 모델로 들어갔다가는 도배 댓글에 들어가있는 애먼 단어가 'toxic'을 결정하게 될 수 있다는 것이다. 따라서 이를 어떻게든 처리해주어야 한다.

이후 ["ip","link","article_id","username",] 등의 feature 또한 확인 후 corpus cleaning 돌입.

7. Corpus Cleaning

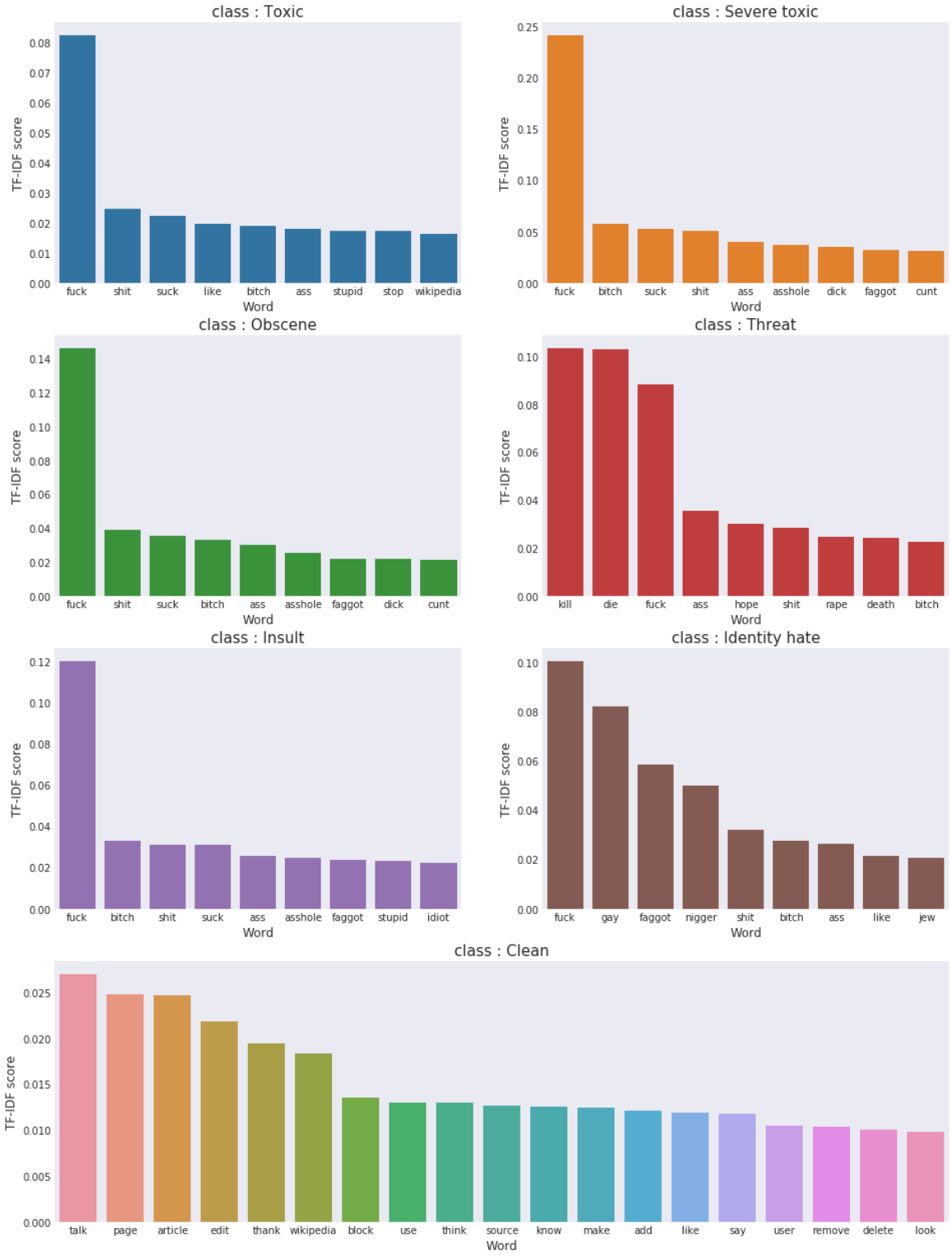
```
def clean(comment):
    """
    This function receives comments and returns clean word-list
```

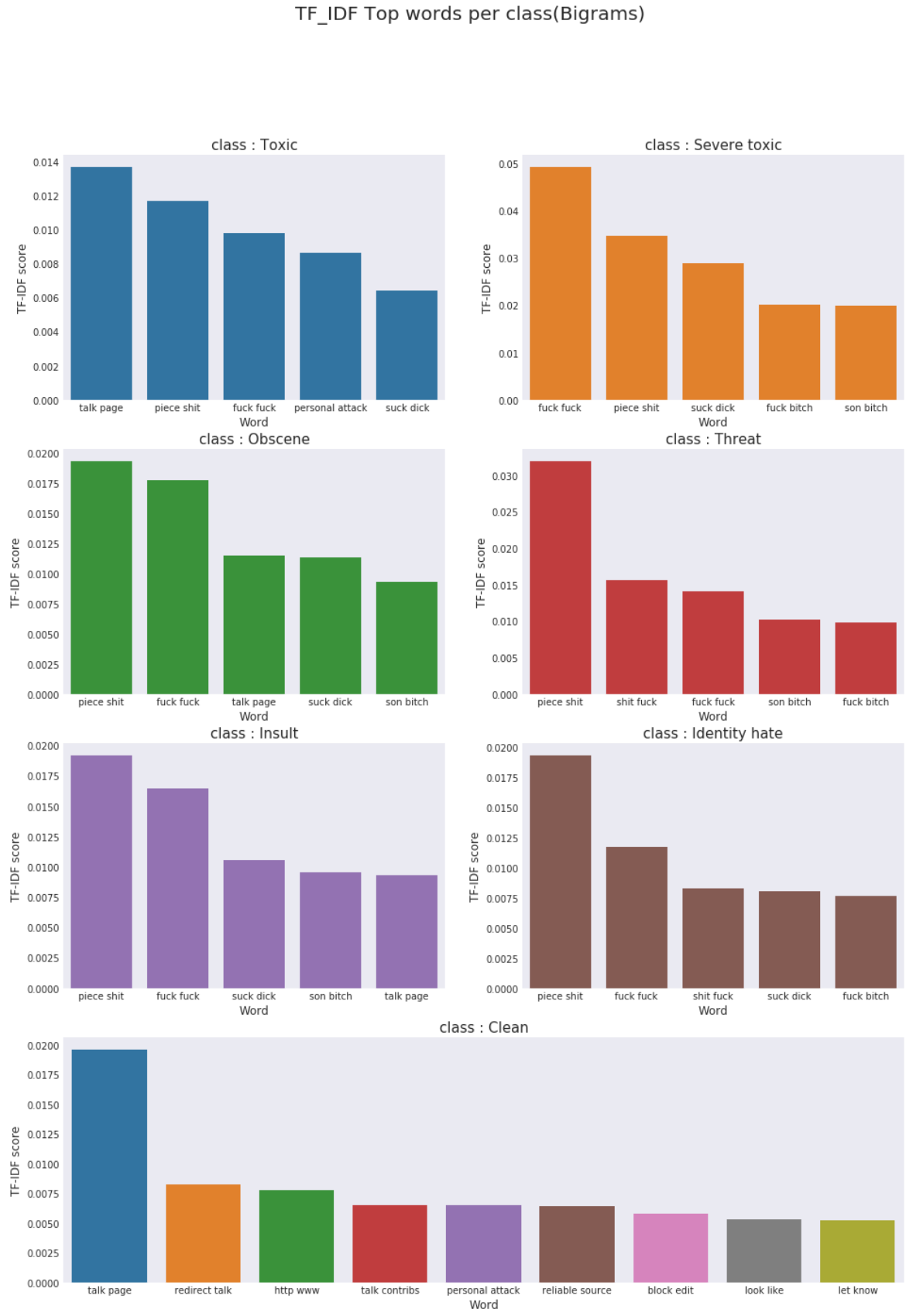
```
    """
    #Convert to lower case , so that Hi and hi are the same
    comment=comment.lower()
    #remove \n
    comment=re.sub("\\n","",comment)
    # remove leaky elements like ip,user
    comment=re.sub("\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}","",comment)
    #removing usernames
    comment=re.sub("\[\[.*\]","",comment)

    #Split the sentences into words
    words=tokenizer.tokenize(comment)

    # (')aphostophe  replacement (ie)   you're --> you are
    # ( basic dictionary lookup : master dictionary present in a hidden block of code)
    words=[APPO[word] if word in APPO else word for word in words]
    words=[lem.lemmatize(word, "v") for word in words]
    words = [w for w in words if not w in eng_stopwords]

    clean_sent=" ".join(words)
    # remove any non alphanum,digit character
    #clean_sent=re.sub("\W+"," ",clean_sent)
    #clean_sent=re.sub("  "," ",clean_sent)
    return(clean_sent)
```

8.Direct Features

다 날렸으니 이제 Direct Features, 사용 가능한 피쳐 사용합시다.

TF_IDF Top words per class(unigrams)




TF_IDF Top words per class(Bigrams)
class : Toxic
TF-IDF score
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
talk page
piece shit
fuck fuck
personal attack
suck dick
Word
class : Severe toxic
TF-IDF score
0.05
0.04
0.03
0.02
0.01
0.00
fuck fuck
piece shit
suck dick
fuck bitch
son bitch
Word
class : Obscene
TF-IDF score
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
piece shit
fuck fuck
talk page
suck dick
son bitch
Word
class : Threat
TF-IDF score
0.030
0.025
0.020
0.015
0.010
0.005
0.000
piece shit
shit fuck
fuck fuck
son bitch
fuck bitch
Word
class : Insult
TF-IDF score
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
piece shit
fuck fuck
suck dick
son bitch
talk page
Word
class : Identity hate
TF-IDF score
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
piece shit
fuck fuck
shit fuck
suck dick
fuck bitch
Word
class : Clean
TF-IDF score
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
talk page
redirect talk
http www
talk contribs
personal attack
reliable source
block edit
look like
let know
Word

9. Modelling

EDA만 하지는 않는군요?

SVM을 사용해 간단한 모델을 굴리셨습니다.

```
class NbSvmClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, C=1.0, dual=False, n_jobs=1):
        self.C = C
        self.dual = dual
        self.n_jobs = n_jobs

    def predict(self, x):
        # Verify that model has been fit
        check_is_fitted(self, ['_r', '_clf'])
        return self._clf.predict(x.multiply(self._r))

    def predict_proba(self, x):
        # Verify that model has been fit
        check_is_fitted(self, ['_r', '_clf'])
        return self._clf.predict_proba(x.multiply(self._r))

    def fit(self, x, y):
        # Check that X and y have correct shape
        y = y.values
        x, y = check_X_y(x, y, accept_sparse=True)

        def pr(x, y_i, y):
            p = x[y==y_i].sum(0)
            return (p+1) / ((y==y_i).sum()+1)

        self._r = sparse.csr_matrix(np.log(pr(x,1,y) / pr(x,0,y)))
        x_nb = x.multiply(self._r)
        self._clf = LogisticRegression(C=self.C, dual=self.dual, n_jobs=self.n_jobs).fit(x_nb, y)
        return self
```