ML-C7

22.08.01

텍스트 데이터 다루기

범주형 + 연속형 + 텍스트 데이터가 있다.

스팸메일 분류나, 이민정책에관한 정치인의견 분석을 위해 트윗 분석

-> 전처리가 필요하다.

텍스트 데이터 다루기

문자열 특성은 직접 살펴보는 수 밖에 없다.

1. 범주형 데이터
2. 범주에 의미를 연결시킬 수 있는 임의의 문자열
3. 구조화된 문자열 데이터
4. 텍스트 데이터

범주형 데이터

고정된 목록으로 구성된다.

Ex) 드롭다운 메뉴에서 “red”, “blue” , “yellow” …. 중 하나를 선택해야한다.
-> 당연히 범주형 변수로 원핫 인코딩 되어야한다.

-> 하지만 다른 문자열도 많이 나온다면, 이값들이 얼마나 자주 나타나는지 히스토그램을 그려볼 수 있다.

-> 또한 오타(검정인데 검점으로 작성한경우.) / 회색을 쥐색으로 작성한경우

=> 범주에 의미를 연결시킬 수 있는 임의의 문자열에 해당한다.

텍스트 데이터

문자열 데이터의 마지막종류는 자유로운 형태의 절과 문장으로 구성된 텍스트 데이터

트윗, 채팅, 호텔리뷰, 셰익스피어 작품, 위키백과 문서, 구텐베르— 프로젝트의 5만권의 전자책등.

->Natural Language Processing(NLP)

예제 어플리케이션 : 영화 리뷰 감성 분석

IMDB 사이트에서 1~10점중 1-4점은 음성 / 7-10점은 양성 (양성은 긍정, 음성이 부정)

텍스트 데이터

```
from sklearn.datasets import load_files

reviews_train = load_files("C:/Users/Windiws/Documents/aclImdb_v1/aclImdb/train")

text_train , y_train = reviews_train.data, reviews_train.target

print(type(text_train))
print(len(text_train))
print(y_train[6])
```

```
<class 'list'>
75000
b'Gloomy Sunday - Ein Lied von Liebe und Tod directed by Rolf Sch₩xc3₩xbcbel in 1999 is a romantic, absorbing, beautiful, and heartbreaking m
ovie. It started like Jules and Jim; it ended as one of Agatha Christie₩'s books, and in between it said something about love, friendship, de
votion, jealousy, war, Holocaust, dignity, and betrayal, and it did better than The Black Book which is much more popular. It is not perfect,
and it made me, a cynic, wonder in the end on the complexity of the relationships and sensational revelations, and who is who to whom but the
movie simply overwhelmed me. Perfect or not, it is unforgettable. All four actors as the parts of the tragic not even a triangle but a rectan
gle were terrific. I do believe that three men could fell deeply for one girl as beautiful and dignified as Ilona in a star-making performanc
e by young Hungarian actress Erica Marozs₩xc3₩xa1n and who would not? The titular song is haunting, sad, and beautiful, and no doubt deserves
the movie been made about it and its effect on the countless listeners. I love the movie and I am surprised that it is so little known in thi
s country. It is a gem.<br /><br />The fact that it is based on a story of the song that had played such important role in the lives of all c
haracters made me do some research, and the real story behind the song of Love and Death seems as fascinating as the fictional one. The song
was composed in 1930s by Rezs₩xc3₩xb6 Seress and was believed to have caused many suicides in Hungary and all over Europe as the world was mo
ving toward the most devastating War of the last century. Rezs₩xc3₩xb6 Seress, a Jewish-Hungarian pianist and composer, was thrown to the Con
centration Camp but survived, unlike his mother. In January, 1968, Seress committed suicide in Budapest by jumping out of a window. According
to his obituary in the New York Times, "Mr. Seres complained that the success of "Gloomy Sunday" actually increased his unhappiness, because
he knew he would never be able to write a second hit." <br /><br />Many singers from all over the world have recorded their versions of the s
ongs in different languages. Over 70 performers have covered the song since 1935, and some famous names include Billie Holiday, Paul Robeson,
Pyotr Leschenko (in Russian, under title "Mratschnoje Woskresenje"), Bjork, Sarah McLachlan, and many more. The one that really got to me and
made me shiver is by Diamanda Gal₩xc3₩xa1s, the Greek born American singer/pianist/performer with the voice of such tragic power that I still
can₩'t get over her singing. Gal₩xc3₩xa1s has been described as "capable of the most unnerving vocal terror", and in her work she mostly conc
entrates on the topics of "suffering, despair, condemnation, injustice and loss of dignity." When she sings the Song of Love and Death, her v
oice that could₩'ve belonged to the most tragic heroines of Ancient Greece leaves no hope and brings the horror and grief of love lost foreve
r to the unbearable and incomparable heights.<br /><br />8.5/10'
```

텍스트 데이터를 BOW로 표현하기.

BOW(Bags of Words)는 가장 간단하지만 효과적이면서 널리 쓰이는 방법이다.
이 방법을 쓰면, 장, 문단, 문장, 서식 같은 입력 텍스트의 구조 대부분을 잃고, 각 단어가 이 말 뭉치에 있는 텍스트에 얼마나 많이 나타나느지만 계산한다.

구조와 상관없이 단어의 출현 횟수만 세기 때문에 텍스트를 담는 가방으로 생각할 수 있다.

1. 토큰화 : 각 문서를 문서에 포함된 단어로 나눈다. Ex) 공백이나 구두점 등을 기준으로 분리.

2. 어휘 사전 구축 : 모든 문서에 나타난 모든 단어의 어휘를 모으고 번호를 매긴다.(알파벳순서)

3. 인코딩 : 어휘 사전의 단어가 문서마다 몇번이 나타나는지를 센다.

```
from sklearn.feature_extraction.text import CountVectorizer
bards_words = ["the fool doth think he is wise,","but the wise man knows himself to be a fool"]
vect = CountVectorizer()
vect.fit(bards_words)
vect.vocabulary_
```

```
{'the': 9, 'fool': 3, 'doth': 2, 'think': 10,
'he': 4, 'is': 6, 'wise': 12, 'but': 1, 'man':
8, 'knows': 7, 'himself': 5, 'to': 11, 'be': 0}
```

```
bow=vect.transform(bards_words)
#vect.transform의 output은 scipy의 희소행렬 형태이다. 다음은 이를 numpy행렬로 바꿔주었다.
bow.toarray()
```

```
array([[0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0,
1], [1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1]],
dtype=int64)
```

영화리뷰 데이터를 BOW로 표현하기.

```
vect = CountVectorizer().fit(text_train)
X_train=vect.transform(text_train)
X_train
```

```
<25000x74849 sparse matrix of type '<class
'numpy.int64'>' with 3445861 stored elements in
Compressed Sparse Row format>
```

```
vect = CountVectorizer().fit(text_train)
X_train=vect.transform(text_train)
feature_names = vect.get_feature_names()
feature_names[:20]
```

```
['00', '000', '0000000000001',
'00001', '00015', '000s', '001',
'003830', '006', '007', '0079',
'0080', '0083', '0093638', '00am',
'00pm', '00s', '01', '01pm', '02']
```

앞에 20개의 특성은 숫자이다. 의미없는 특성일 확률이 높다.

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
vect = CountVectorizer().fit(text_train)

X_train=vect.transform(text_train)

scores = cross_val_score(LogisticRegression(),X_train,y_train, cv=5)
print(np.mean(scores))
```

```
0.8815999999999999
```

영화리뷰 데이터를 BOW로 표현하기.(min_df)

```
param_grid = {'C' : [0.001,0.01,0.1,1,10]}
grid =
GridSearchCV(LogisticRegression(),param_grid,cv=5)
grid.fit(X_train,y_train)
print(grid.best_score_)
print(grid.best_params_)
```

```
0.88808 {'C': 0.1}
```

```
vect = CountVectorizer(min_df=5).fit(text_train)
X_train = vect.transform(text_train)
X_train

#min_df는 최소 등장횟수를 지정해줄 수 있음.

grid.fit(X_train,y_train)
grid.best_score_
```

```
<25000x27272 sparse matrix of
type '<class 'numpy.int64'>'
with 3368680 stored elements
in Compressed Sparse Row
format>

0.88816
```

불용어 제거를 통한 특성개수 줄이기.

```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

print(ENGLISH_STOP_WORDS)
Print(len(E ENGLISH_STOP_WORDS))
```

318


```
frozenset({'sincere', 'will', 'already', 'otherwise', 'now', 'hence', 'being', 'on', 'our', 'therein', 'hereafter', 'any', 'twenty', 'seeme
d', 'wherein', 'hasnt', 'describe', 'someone', 'once', 'are', 'again', 'your', 'whom', 'before', 'when', 'along', 'less', 'where', 'last', 'c
ouldnt', 'bill', 'own', 'might', 'take', 'elsewhere', 'still', 'herein', 'whatever', 'too', 'noone', 'twelve', 'eg', 'show', 'anyone', 'see
m', 'yet', 'thin', 'not', 'beyond', 'him', 'therefore', 'for', 'both', 'also', 'per', 'latter', 'never', 'yourself', 'whence', 'back', 'all',
'ever', 'through', 'except', 'system', 'as', 'each', 'around', 'cannot', 'nor', 'three', 'from', 'somewhere', 'bottom', 'made', 'other', 'wh
o', 'herself', 'over', 'see', 'next', 'same', 'while', 'has', 'most', 'cant', 'anywhere', 'alone', 'whether', 'off', 'her', 'if', 'whereby',
'them', 'give', 'meanwhile', 'empty', 'be', 'was', 'above', 'anything', 'seems', 'because', 'rather', 'then', 'should', 'become', 'themselve
s', 'whose', 'no', 'or', 'their', 'me', 'thick', 'that', 'indeed', 'may', 'have', 'he', 'nowhere', 'of', 'behind', 'found', 'but', 'its', 'si
nce', 'itself', 'yours', 'among', 'been', 'forty', 'thence', 'four', 'had', 'himself', 'front', 'against', 'the', 'name', 'than', 'very', 'on
ly', 'interest', 'fifty', 'after', 'well', 'many', 'up', 'upon', 'must', 'until', 'go', 'latterly', 'formerly', 'besides', 'always', 'thus',
'his', 'between', 'else', 'perhaps', 'two', 'beside', 'either', 'thereafter', 'seeming', 'enough', 'whenever', 'hereby', 'five', 'something',
'thereupon', 'neither', 'you', 'nine', 'amongst', 'nevertheless', 'amount', 'cry', 'least', 'mostly', 'call', 'beforehand', 're', 'a', 'ten',
'inc', 'an', 'etc', 'toward', 'however', 'wherever', 'these', 'how', 'could', 'none', 'it', 'into', 'fifteen', 'across', 'eleven', 'my', 'ful
l', 'whither', 'can', 'nobody', 'un', 'below', 'why', 'at', 'six', 'down', 'everything', 'serious', 'keep', 'i', 'get', 'fill', 'ours', 'wher
eupon', 'everywhere', 'hereupon', 'detail', 'within', 'whoever', 'would', 'eight', 'throughout', 'first', 'together', 'we', 'more', 'though',
'became', 'becomes', 'whereas', 'they', 'ltd', 'whole', 'were', 'sixty', 'and', 'thru', 'co', 'by', 'namely', 'moreover', 'done', 'what', 'so
me', 'to', 'several', 'few', 'she', 'without', 'due', 'with', 'which', 'de', 'part', 'often', 'anyhow', 'via', 'thereby', 'others', 'anothe
r', 'fire', 'sometime', 'towards', 'nothing', 'put', 'about', 'so', 'everyone', 'further', 'anyway', 'ourselves', 'find', 'out', 'yourselve
s', 'top', 'am', 'do', 'con', 'during', 'in', 'there', 'almost', 'side', 'mine', 'such', 'whereafter', 'even', 'please', 'under', 'is', 'some
times', 'afterwards', 'mill', 'much', 'although', 'third', 'former', 'ie', 'hers', 'every', 'those', 'becoming', 'move', 'amoungst', 'this',
'one', 'somehow', 'here', 'onto', 'hundred', 'myself', 'us'})
```


```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

vect = CountVectorizer(min_df=5, stop_words="english").fit(text_train)
X_train = vect.transform(text_train)

repr(X_train)

grid.fit(X_train,y_train)
print(grid.best_score_)
```

0.88288

Tf-idf로 데이터 스케일 변경하기.

중요치 않아 보이는 특성을 제외하는 대신 얼마나 으미 있는 특성인지를 계산해서 스케일을 조정하는 방식

말뭉치의 다른 문서보다 특정문서에 자주 나타나는 단어에 높은 가중치를 주는 방법.

->TfidTransformer는 CountVectorizer만든 회소행렬을 입력받아 변환한다. -> BOW특성추출과 tf-idf변환을 수행한다.

Tfidf(w,d) = tf*((log(N+1/Nw+1))+1)
W는 단어 , d는 대상문서, N은 훈련세트의 문서갯수

```
text_train , y_train = reviews_train.data, reviews_train.target
text_test , y_test = reviews_test.data, reviews_test.target


from sklearn.feature_extraction.text import TfidfVectorizer

pipe = make_pipeline(TfidfVectorizer(min_df=5),LogisticRegression())
param_grid = {'logisticregression__C' : [0.001,0.01,0.1,1,10]}

grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(text_train , y_train)

grid.best_score_
```

```
0.89192
```

Tf-idf로 데이터 스케일 변경하기.

```
vector = grid.best_estimator_.named_steps["tfidfvectorizer"]
X_train =vector.transform(text_train)
max_value = X_train.max(axis=0).toarray().ravel()

sorted_by_tf = max_value.argsort()
feature_names = np.array(vector.get_feature_names())

print("low_20 : ",feature_names[sorted_by_tf[:20]])
print("high_20 : ",feature_names[sorted_by_tf[-20:]])
```

```
low_20 : ['suplexes' 'gauche' 'hypocrites'
'oncoming' 'galadriel' 'songwriting'
'cataclysmic' 'sylvain' 'emerald'
'mclaughlin' 'oversee' 'pressuring' 'uphold'
'thieving' 'inconsiderate' 'ware' 'denim'
'booed' 'reverting' 'spacious']
```

```
high_20 : ['muppet' 'brendan' 'zatoichi'
'dev' 'demons' 'lennon' 'bye' 'weller' 'woo'
'sasquatch' 'botched' 'xica' 'darkman'
'casper' 'doodlebops' 'steve' 'smallville'
'wei' 'scanners' 'pokemon']
```

#tf-idf가 낮은 특성은 전체 문서에 걸쳐 매우 많이 나타나거나, 조금씩만 사용되거나, 매우 긴 문서에만 사용.

높은 특성은 어떤쇼나 영화를 나타내는 경우가 많다. 이런 단어들은 특정 쇼나 드라마에 대한 리뷰에서만 나타나지만, 이 특정 리뷰에서 매우 자주 나타나는 경향이 있다.

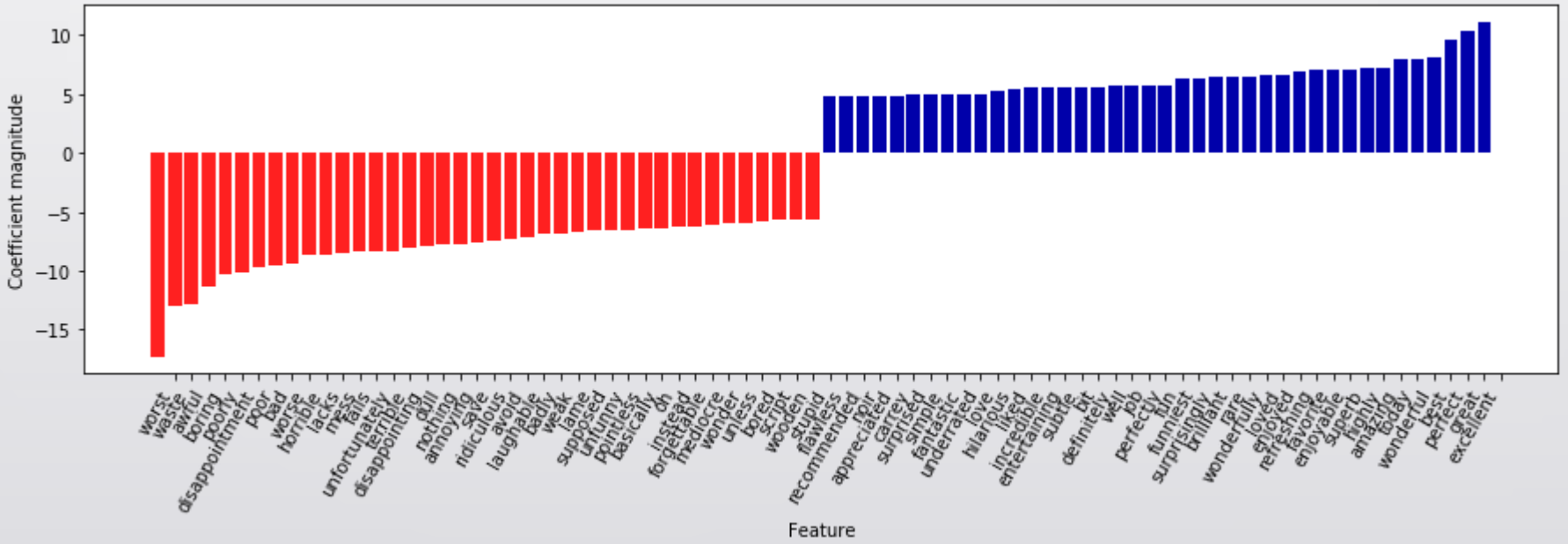
Tf-idf로 데이터 스케일 변경하기.

```
sorted_by_idf = np.argsort(vector.idf_)

print("low_100 : ",feature_names[sorted_by_idf[:100]])

# 불용어가 대다수 이지만, good, great, bad와 같은
감성분석에 유용할만한 특성도 있다. 하지만
tf_idf관점에서는 덜 중요한 단어로 취급되었다.
```

```
low_100 : ['the' 'and' 'of' 'to' 'this' 'is' 'it'
'in' 'that' 'but' 'for' 'with' 'was' 'as' 'on'
'movie' 'not' 'br' 'have' 'one' 'be' 'film'
'are' 'you' 'all' 'at' 'an' 'by' 'so' 'from' 'like'
'who' 'they' 'there' 'if' 'his' 'out' 'just'
'about' 'he' 'or' 'has' 'what' 'some' 'good'
'can' 'more' 'when' 'time' 'up' 'very'
'even' 'only' 'no' 'would' 'my' 'see' 'really'
'story' 'which' 'well' 'had' 'me' 'than'
'much' 'their' 'get' 'were' 'other' 'been'
'do' 'most' 'don' 'her' 'also' 'into' 'first'
'made' 'how' 'great' 'because' 'will'
'people' 'make' 'way' 'could' 'we' 'bad'
'after' 'any' 'too' 'then' 'them' 'she'
'watch' 'think' 'acting' 'movies' 'seen'
'its']
```

모델 계수 조사

```
mglearn.tools.visualize_coefficients(
    grid.best_estimator_.named_steps["logisticregression"].coef_[0],feature_names,n_top_features=40)
```



여러 단어로 만든 BOW

```
cv = CountVectorizer(ngram_range=(1,1)).fit(bards_words)
print(len(cv.vocabulary_))
print(cv.get_feature_names())
cv.transform(bards_words).toarray()
```

```
13 ['be', 'but', 'doth', 'fool', 'he', 'himself', 'is', 'knows', 'man', 'the', 'think', 'to', 'wise']
```

```
14 ['be fool', 'but the', 'doth think', 'fool doth', 'he is', 'himself to', 'is wise', 'knows himself', 'man knows', 'the fool', 'the wise', 'think he', 'to be', 'wise man']
```

```
array([[0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0], [1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1]], dtype=int64)
```

```
bards_words = ["the fool doth think he is wise,","but the wise man knows himself to be a fool"]
cv = CountVectorizer(ngram_range=(1,3)).fit(bards_words)
print(len(cv.vocabulary_))
print(cv.get_feature_names())
```

```
39

['be', 'be fool', 'but', 'but the', 'but the wise', 'doth', 'doth think', 'doth think he', 'fool', 'fool doth', 'fool doth think',
'he', 'he is', 'he is wise', 'himself', 'himself to', 'himself to be', 'is', 'is wise', 'knows', 'knows himself', 'knows himself
to', 'man', 'man knows', 'man knows himself', 'the', 'the fool', 'the fool doth', 'the wise', 'the wise man', 'think', 'think
he', 'think he is', 'to', 'to be', 'to be fool', 'wise', 'wise man', 'wise man knows']
```

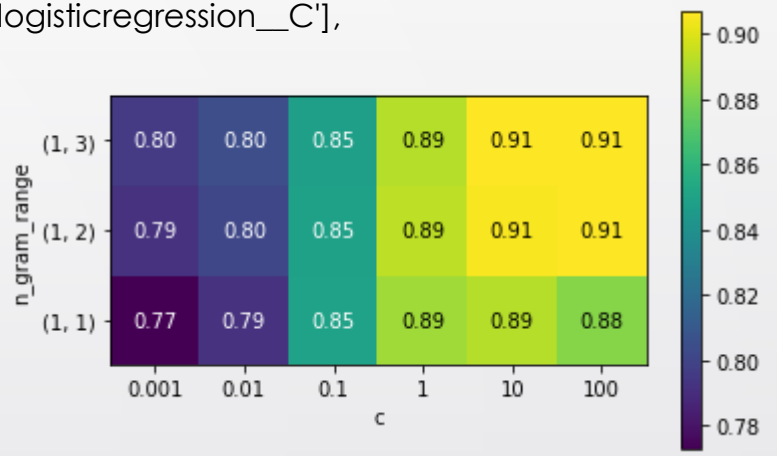
IMDb ->tfidVectorizer 적용, 그리드서치를 통한 최적의 n_gram범위 찾기

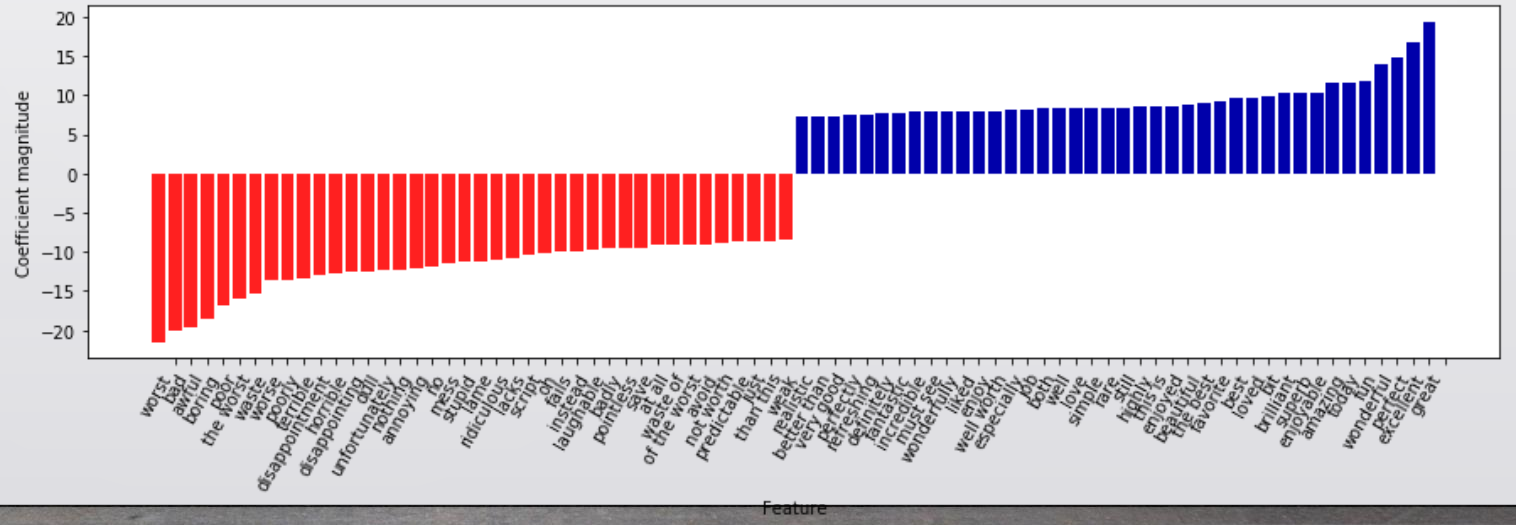
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

pipe = make_pipeline(TfidfVectorizer(min_df=5),LogisticRegression())
param_grid = {'logisticregression__C' : [0.001,0.01,0.1,1,10,100],
              'tfidfvectorizer__ngram_range':[(1,1),(1,2),(1,3)]}
#모델__매개변수명 일치해야 작동함.

grid = GridSearchCV(pipe,param_grid,cv=5)
grid.fit(text_train,y_train)
print(grid.best_score_)
print(gird.best_params_)
```

0.90664

IMDb ->tfidVectorizer 적용, 그리드서치를 통한 최적의 n_gram범위 찾기

```
scores = grid.cv_results_['mean_test_score'].reshape(-1,3).T

heatmap = mglearn.tools.heatmap(scores,xlabel="c",ylabel="n_gram_range",xticklabels =param_grid['logisticregression__C'],
                                yticklabels=param_grid['tfidfvectorizer__ngram_range'])

plt.colorbar(heatmap)

vect = grid.best_estimator_.named_steps['tfidfvectorizer']
feature_names = np.array(vect.get_feature_names())
coef = grid.best_estimator_.named_steps['logisticregression'].coef_

mglearn.tools.visualize_coefficients(coef[0],feature_names,n_top_features=40)
```

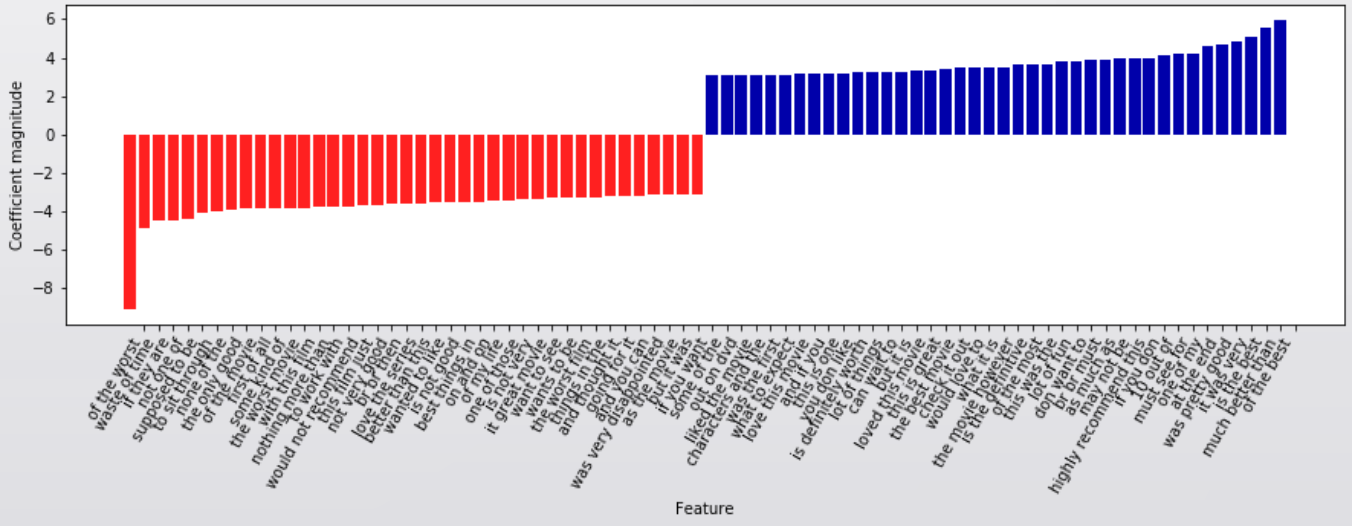




트라이그램에서 중요도 보기.

```
vect = grid.best_estimator_.named_steps['tfidfvectorizer']
feature_names = np.array(vect.get_feature_names())

mask = np.array([len(feature.split(" "))for feature in feature_names]) == 3

mglearn.tools.visualize_coefficients(coef.ravel()[mask],feature_names[mask],n_top_features
=40)
```

#유니 그램에서 트라이그램으로 가도 큰 성능 변화 가 없는 이유는 대부분 독립적일때 의미가 없는 경우가 많기 때문이다.



고급 토큰화

각 단어의 어간을 추출해야한다.
Placed , placing, … 어간 : place -> 어간추출방식

Replace사전 : replaced, replacement, replacing -> 표제어 추출.

(정규화의 한형태이다.)

-> spacy패키지에 구현된 어간추출기와 표제어 추출방식

	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밋었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1
5	5403919	막 걸음마 뗀 3세부터 초등학교 1학년생인 8살용영화.ㅋㅋㅋ...별반개도 아까움.	0
6	7797314	원작의 긴장감을 제대로 살려내지못했다.	0
7	9443947	별 반개도 아깝다 욕나온다 이응경 길용우 연기생활이몇년인지..정말 발로해도 그것보단...	0
8	7156791	액션이 없는데도 재미 있는 몇안되는 영화	1
9	5912145	왜케 평점이 낮은건데? 꽤 볼만한데.. 헐리우드식 화려함에만 너무 길들여져 있나?	1

```
df_train = pd.read_csv('C:/Users/Windiws/Documents/ratings_train.txt',delimiter='\t',keep_default_na=False)
df_train.head(n=10)
```

한글데이터(네이버영화 평점 데이터셋)

```
text_train, y_train = df_train['document'].values , df_train['label'].values
df_test =
pd.read_csv('C:/Users/Windiws/Documents/ratings_test.txt',delimiter='\t',keep_
default_na=False)
text_test, y_test = df_test['document'].values , df_test['label'].values

print(len(y_train), np.bincount(y_train))

print("test : ",len(y_test),np.bincount(y_test))
```

150000 [75173 74827] test : 50000 [24827 25173]

한글데이터(네이버영화 평점 데이터셋) -> Konlpy(train_1000개 활용)

```
from konlpy.tag import Okt

okt_tag = Okt()

def okt_tokenizer(text):
    return okt_tag.morphs(text)

okt_param_grid = {'tfidfvectorizer__min_df':[3,5,7],
                  'tfidfvectorizer__ngram_range':[(1,1),(1,2),(1,3)],
                  'logisticregression__C':[0.1,1,10]}

okt_pipe = make_pipeline(TfidfVectorizer(tokenizer=okt_tokenizer),
                         LogisticRegression(solver='liblinear'))

okt_grid = GridSearchCV(okt_pipe,okt_param_grid,cv=3)

okt_grid.fit(text_train[0:1000],y_train[0:1000])

print("best_validation_score : ",okt_grid.best_score_)
print("best_parameter :",okt_grid.best_params_)
```

```
best_validation_score : 0.704 best_parameter :
{'logisticregression__C': 1, 'tfidfvectorizer__min_df': 3,
'tfidfvectorizer__ngram_range': (1, 1)}
```

한글데이터(네이버영화 평점 데이터셋) -> Konlpy

```
from konlpy.tag import Okt

okt_tag = Okt()

def okt_tokenizer(text):
   return okt_tag.morphs(text)

okt_param_grid = {'tfidfvectorizer__min_df':[3,5,7],
            'tfidfvectorizer__ngram_range':[(1,1),(1,2),(1,3)],
            'logisticregression__C':[0.1,1,10]}

okt_pipe = make_pipeline(TfidfVectorizer(tokenizer=okt_tokenizer),
                  LogisticRegression(solver='liblinear'))

okt_grid = GridSearchCV(okt_pipe,okt_param_grid,cv=3)

okt_grid.fit(text_train[0:1000],y_train[0:1000])
```

best_validation_score : 0.704 best_parameter : {'logisticregression__C': 1, 'tfidfvectorizer__min_df': 3, 'tfidfvectorizer__ngram_range': (1, 1)}

```
0.705
```

```
X_test_okt = okt_grid.best_estimator_.named_steps["tfidfvectoriz
er"].transform(text_test)
score = okt_grid.best_estimator_.named_steps["logisticregression
"].score(X_test_okt,y_test)
print("{:.3f}".format(score))
```

한글데이터(네이버영화 평점 데이터셋) -> Konlpy

```
from konlpy.tag import Okt
from konlpy.tag import Mecab

from sklearn.pipeline import make_pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

mecab = Mecab()

def mecab_tokenizer(text):
  return mecab.morphs(text)
```

```
mecab_param_grid = {'tfidfvectorizer__min_df':[3,5,7],
                    'tfidfvectorizer__ngram_range':[(1,1),(1,2),(1,3)],
                    'logisticregression__C':[0.1,1,10]}

mecab_pipe = make_pipeline(TfidfVectorizer(tokenizer=mecab_tokenizer),
                           LogisticRegression(solver='liblinear'))

mecab_grid = GridSearchCV(mecab_pipe,mecab_param_grid,cv=3,n_jobs=1)

mecab_grid.fit(text_train,y_train)

print("best_validation_score : ",mecab_grid.best_score_)
print("best_parameter :",mecab_grid.best_params_)
```

```
best_validation_score :  0.8699066666666667
best_parameter : {'logisticregression__C': 10,
'tfidfvectorizer__min_df': 3,
'tfidfvectorizer__ngram_range': (1, 3)}
```

```
X_test_mecab = mecab_grid.best_estimator_.named_steps["tfidfvectorizer"].t
ransform(text_test)
score = mecab_grid.best_estimator_.named_steps["logisticregression"].score
(X_test_mecab,y_test)
print("{:.3f}".format(score))
```

```
0.875
```

토픽 모델링 / 문서 군집화

같은 토픽으로 문서를 묶는것 -> 토픽 모델링

Latent Dirichlet Allocation (LDA) ->

각 문서에 토픽의 일부가 혼합되어 있다고 간주함.

여기서 말하는 토픽은 주제가 아니다.

Ex) 기자가 자주쓰는 단어 일 수도 있다는 것.

```
vect = CountVectorizer(max_features=10000,max_df = .15)
X = vect.fit_transform(text_train)

from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components=10, learning_method="batch",m
ax_iter=25,random_state=0)

document_topics = lda.fit_transform(X)

lda.components_.shape
```

```
(10, 10000)
```

토픽 모델링 / 문서 군집화

```
from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components=10,
learning_method="batch",max_iter=25,random_state=0)

document_topics = lda.fit_transform(X)

sorting= np.argsort(lda.components_,axis=1)[:,::-1]
feature_names = np.array(vect.get_feature_names())

mglearn.tools.print_topics(topics=range(10),feature_names=feature_names,sorting=sorting,t
opics_per_chunk=5,n_words=10)
```

topic 0	topic 1	topic 2	topic 3	topic 4
너무	영화	ㅋㅋ	영화	영화를
스토리	평점이	너무	있는	다시
없고	너무	정말	그리고	보고
연기	만든	최고	대한	많이
연기도	가장	봤는데	이야기	정말
스토리가	여운이	ㅋㅋㅋ	아름다운	봐도
내용도	봤는데	재밌게	작품	이런
연기가	별로	ㅎㅎ	싶다	마지막
연출	보면	그래도	재밌어요	영화가
스토리도	남는	재미있게	없는	하지만

topic 5	topic 6	topic 7	topic 8	topic 9
정말	진짜	그냥	좋은	너무
영화	보는	영화	이게	이런
최고의	완전	쓰레기	ㅠㅠ	영화는
드라마	ㅡㅡ	없는	재밌다	영화가
평점	내내	아깝다	이건	역시
이렇게	보다가	재미	너무	없다
내가	이거	아니고	영화라고	이거
10점	최고다	시간	보다	좋다
끝까지	이제	아까운	이걸	말이
최악의	제발	1점도	ㅜㅜ	무슨