HW4 - Text Classification

Student Name: Chunran Yao, Ze Chen

In [20]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegressionCV
import warnings
warnings.filterwarnings('ignore')
from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransf
ormer
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import Normalizer
from scipy.sparse import coo_matrix, hstack
from gensim import models
import re
import gensim.parsing.preprocessing as preprocessing
import matplotlib.pyplot as plt
```

import data

In [44]:
```
train_data = pd.read_csv("reddit_200k_train.csv", encoding = 'latin1')
train_data.shape
```

Out[44]:
```
(167529, 8)
```

In [5]:
```
train_data.head()
```

Out[5]:

	Unnamed: 0	body	score.x	parent_id.x	id	created_utc.x	retrieved_on	REMOVED
0	1	I've always been taught it emerged from the ea...	2	t3_81u15i	dv551g6	1520121101	1524782256	False
1	2	As an ECE, my first feeling as "HEY THAT'S NOT...	2	t3_72sk35	dnl66g6	1506533157	1507150439	True
2	3	Monday: Drug companies stock dives on good new...	5	t3_8o88yr	e02sjhz	1528087570	1532170350	True
3	4	i learned that all hybrids are unfertile i won...	0	t3_6xg9t8	dmfojjp	1504290041	1506407514	False
4	5	Well i was wanting to get wasted tonight. Not...	3	t3_99wi9m	e4rtew8	1535140675	1537893540	False

In [6]:
```
test_data = pd.read_csv("reddit_200k_test.csv",encoding='latin1')
#test_data = pd.read_csv("reddit_200k_test.csv",usecols = ['body','REMOVED'],encoding = 'unicode_escape')
```

In [7]:
```
test_data.head()
```

Out[7]:

	Unnamed: 0	body	score.x	parent_id.x	id	created_utc.x	retrieved_on	REMOVED
0	1	Hi Larpo_Nadar, your submission has been remov...	0	t3_74udg6	do15nly	1507377013	1509603985	True
1	2	So out of every 10,000 children with autism wh...	8	t3_879uw5	dwc3dps	1522107010	1525623538	False
2	3	When I was pregnant, I was warned against eati...	4	t3_5qo49s	dd1wtw2	1485686073	1486529379	False
3	4	Imagine if this find was the bug that eradicat...	14	t3_5qmr9c	dd0qpcr	1485618726	1486509114	True
4	5	Is it a myth that the math says it would take ...	0	t3_6wtiwg	dmb0sg4	1504050679	1504495504	False

In [8]:
```
data = train_data[['body','REMOVED']]
```

Task 1 Bag of Words and simple Features

1.1 Create a baseline model using a bag-of-words approach and a linear model.

In [96]:
```
text_train = data[['body']]
y_train = data[['REMOVED']]
```

Prepare X: change X to list of strings, length is 167529

In [97]:
```
text_train_lst = []
for item in np.array(text_train).tolist():
    text_train_lst.append(item[0])

assert len(text_train_lst)==train_data.shape[0]
```

In [98]:
```
vect = CountVectorizer(token_pattern=r"\b\w+\b")
X_train = vect.fit_transform(text_train_lst)
```

In [99]:
```
X_train
```

Out[99]:
```
<167529x113865 sparse matrix of type '<class 'numpy.int64'>'
        with 5286601 stored elements in Compressed Sparse Row format>
```

Loook at first 20 feature names

In [13]:
```
feature_names = vect.get_feature_names()
print(feature_names[:20])
```

```
['0', '00', '000', '0000', '00000000000000000134', '0000000000000065',
'0000000000000296', '0000000000000327', '0000000000000381', '0000000000
000394', '0000000000000583', '00000000000007', '0000000000000767', '000
0000000000821', '0000000000001014', '0000000000001041', '00000000000017
44', '0000000000002162', '0000000000002565', '0000000000003559']
```

Baseline Model: Bag-of-Words Approach and Logistic Regression

In [61]:
```
param_grid = {"logisticregression__C": [10, 1, 0.1, 0.01, 0.001]}
grid_baseline = GridSearchCV(make_pipeline(CountVectorizer(analyzer="wor
d",stop_words='english',
                                                          token_pattern
=r"\b\w+\b"),
                                           LogisticRegression()),
                    param_grid=param_grid, cv=5, scoring="f1_macro" )
```

In [62]:
```
grid_baseline.fit(text_train_lst, y_train)
```

Out[62]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('countvectorizer', CountVectorizer(analyzer='word', binary
=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words='english...pe
nalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False))]),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'logisticregression__C': [10, 1, 0.1, 0.01, 0.001]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [64]:
```
grid_baseline.best_params_
```

Out[64]:
```
{'logisticregression__C': 1}
```

In [63]:
```
grid_baseline.best_score_
```

Out[63]:
```
0.6620432978080867
```

1.2 Try using n-grams, characters, tf-idf rescaling and possibly other ways to tune the BoW model. Be aware that you might need to adjust the (regularization of the) linear model for different feature sets.

Use ngram

In [66]:
```
param_grid = {"logisticregression__C": [10, 1, 0.1, 0.01, 0.001],
              "countvectorizer__ngram_range": [(1, 1), (1, 2),(2, 3)]}
grid_n_gram = GridSearchCV(make_pipeline(CountVectorizer(analyzer="word"
,stop_words='english',
                                                         token_pattern
=r"\b\w+\b"),
                                         LogisticRegression()),
                           param_grid=param_grid, cv=5, scoring="f1_macro" )
```

In [67]:
```
grid_n_gram.fit(text_train_lst, y_train)
```

Out[67]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('countvectorizer', CountVectorizer(analyzer='word', binary
=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words='english...pe
nalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False))]),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'logisticregression__C': [10, 1, 0.1, 0.01, 0.001],
'countvectorizer__ngram_range': [(1, 1), (1, 2), (2, 3)]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [68]:
```
grid_n_gram.best_params_
```

Out[68]:
```
{'countvectorizer__ngram_range': (1, 2), 'logisticregression__C': 1}
```

In [69]:
```
grid_n_gram.best_score_
```

Out[69]:
```
0.6735237298008734
```

Use min_df

Since the default min_df is 1, I didn't assign min_df = 1 in parameter tuning.

In [70]:
```
param_grid = {"logisticregression__C": [10, 1, 0.1, 0.01, 0.001],
              "countvectorizer__min_df": [2,3,4]}
grid_min_df = GridSearchCV(make_pipeline(CountVectorizer(analyzer="word"
,stop_words='english',
                                                       ngram_range = (
1,2),token_pattern=r"\b\w+\b"),
                                         LogisticRegression()),
                    param_grid=param_grid, cv=5, scoring="f1_macro" )
```

In [71]:
```
grid_min_df.fit(text_train_lst, y_train)
```

Out[71]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('countvectorizer', CountVectorizer(analyzer='word', binary
=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 2), preprocessor=None, stop_words='english...pe
nalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False))]),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'logisticregression__C': [10, 1, 0.1, 0.01, 0.001],
'countvectorizer__min_df': [2, 3, 4]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [72]:
```
grid_min_df.best_params_
```

Out[72]:
```
{'countvectorizer__min_df': 2, 'logisticregression__C': 1}
```

In [73]:
```
grid_min_df.best_score_
```

Out[73]:
```
0.6675357530828799
```

Compared to default min_df, min_df = 1 is better.

Use TF-IDF

In [76]:
```
param_grid = {"logisticregression__C": [10, 1, 0.1, 0.01, 0.001]}
grid_tf_idf = GridSearchCV(make_pipeline(TfidfVectorizer(analyzer="word"
,stop_words='english',
                                                       ngram_range = (
1,2),token_pattern=r"\b\w+\b"),
                                         LogisticRegression(),
                                         memory="cache_folder"),
                    param_grid=param_grid, cv=5, scoring="f1_macro" )
```

In [77]:
```
grid_tf_idf.fit(text_train_lst, y_train)
```

Out[77]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Pipeline(memory='cache_folder',
     steps=[('tfidfvectorizer', TfidfVectorizer(analyzer='word', binary
=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='conten
t',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 2), norm='l2', preprocessor=None, smooth...pena
lty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False))]),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'logisticregression__C': [10, 1, 0.1, 0.01, 0.001]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [78]:
```
grid_tf_idf.best_params_
```

Out[78]:
```
{'logisticregression__C': 10}
```

In [79]:
```
grid_tf_idf.best_score_
```

Out[79]:
```
0.673138309227291
```

Use character

not use stopwords：

In [82]:
```
param_grid = {"logisticregression__C": [10, 1, 0.1, 0.01, 0.001],
              "tfidfvectorizer__analyzer":['char','char_wb']}
grid_char = GridSearchCV(make_pipeline(TfidfVectorizer(ngram_range = (2,
3)),
                                       LogisticRegression(),
                                       memory="cache_folder"),
                         param_grid=param_grid, cv=5, scoring="f1_macro" )
```

In [83]:
```
grid_char.fit(text_train_lst, y_train)
```

Out[83]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Pipeline(memory='cache_folder',
     steps=[('tfidfvectorizer', TfidfVectorizer(analyzer='word', binary
=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='conten
t',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(2, 3), norm='l2', preprocessor=None, smooth...pena
lty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False))]),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'logisticregression__C': [10, 1, 0.1, 0.01, 0.001],
'tfidfvectorizer__analyzer': ['char', 'char_wb']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [84]:
```
grid_char.best_params_
```

Out[84]:
```
{'logisticregression__C': 1, 'tfidfvectorizer__analyzer': 'char_wb'}
```

In [85]:
```
grid_char.best_score_
```

Out[85]:
```
0.6867830555892265
```

use stop words:

In [12]:
```
param_grid = {"logisticregression__C": [10, 1, 0.1, 0.01, 0.001],
              "tfidfvectorizer__analyzer":['char','char_wb']}
grid_char = GridSearchCV(make_pipeline(TfidfVectorizer(ngram_range = (2,
3), stop_words='english'),
                                       LogisticRegression(),
                                       memory="cache_folder"),
                         param_grid=param_grid, cv=5, scoring="f1_macro" )
```

In [13]:
```
grid_char.fit(text_train_lst, y_train)
```

Out[13]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Pipeline(memory='cache_folder',
     steps=[('tfidfvectorizer', TfidfVectorizer(analyzer='word', binary
=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='conten
t',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(2, 3), norm='l2', preprocessor=None, smooth...pena
lty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False))]),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'logisticregression__C': [10, 1, 0.1, 0.01, 0.001],
'tfidfvectorizer__analyzer': ['char', 'char_wb']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [14]:
```
grid_char.best_params_
```

Out[14]:
```
{'logisticregression__C': 1, 'tfidfvectorizer__analyzer': 'char_wb'}
```

In [15]:
```
grid_char.best_score_
```

Out[15]:
```
0.6867830555892265
```

1.3 Explore other features you can derive from the text, such as html, length, punctuation, capitalization or other features you deem important from exploring the dataset

In [ ]:
```
Explore Features by LASSO logistic coefficients
```

In [14]:
```
text_train_13 = text_train.copy()
```

In [15]:
```
lr_pipe = Pipeline(steps=[('cv',CountVectorizer()),
                          ('lr',LogisticRegression(penalty='l1'))])
lr_pipe.fit(text_train_lst, y_train)
```

Out[15]:
```
Pipeline(memory=None,
     steps=[('cv', CountVectorizer(analyzer='word', binary=False, decod
e_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_a...penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False))])
```

In [16]:
```
coef = lr_pipe.named_steps['lr'].coef_
```

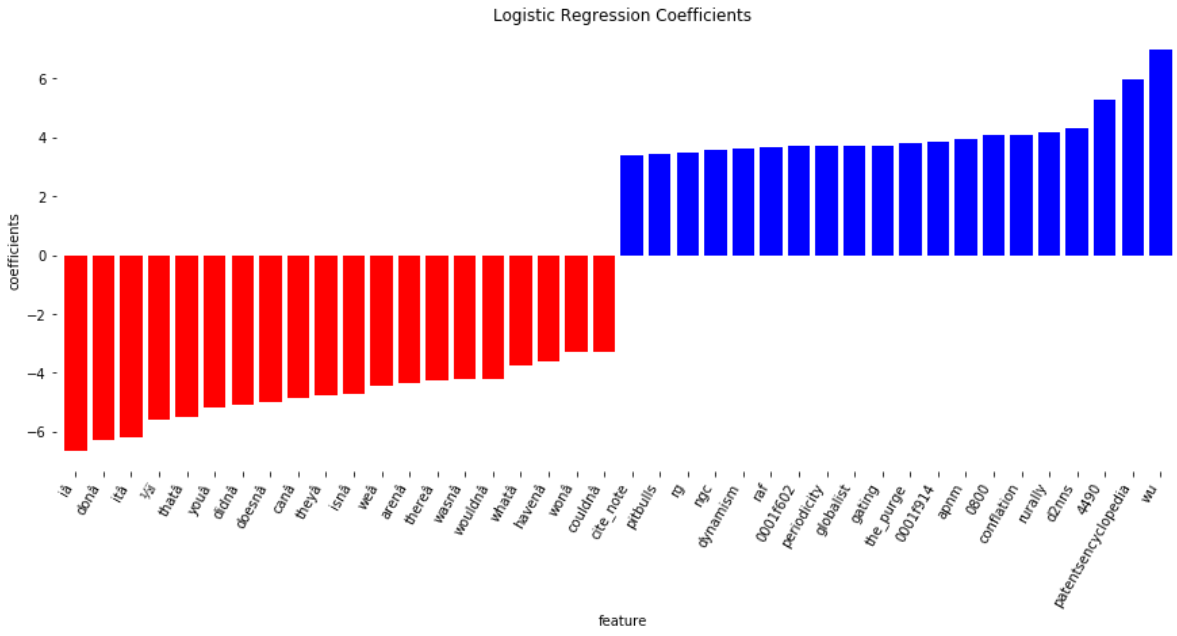
In [17]:
```
feature_names = lr_pipe.named_steps['cv'].get_feature_names()
```

In [26]:
```
def plot_important_features(coef, feature_names, top_n=20, ax=None, rota
tion=60):
    if ax is None:
        ax = plt.gca()

    inds = np.argsort(coef)
    low = inds[:top_n]
    high = inds[-top_n:]
    important = np.hstack([low, high])
    myrange = range(len(important))
    colors = ['red'] * top_n + ['blue'] * top_n

    ax.bar(myrange, coef[important], color=colors)
    ax.set_xticks(myrange)
    ax.set_xticklabels(feature_names[important], rotation=rotation, ha=
"right")
    ax.set_xlim(-.7, 2 * top_n)
    ax.set_frame_on(False)
    ax.set_title('Logistic Regression Coefficients')
    ax.set_ylabel('coefficients')
    ax.set_xlabel('feature')
```

In [27]:
```
plt.figure(figsize=(15,6))
plot_important_features(coef.ravel(), np.array(feature_names),top_n=20,
ax=None, rotation=60 )
```



In [28]:
```
pd.DataFrame({'feature name': feature_names,
              'coefficient': coef.ravel().tolist()}).nlargest(20,'coeffi
cient')
```

Out[28]:

	feature name	coefficient
112178	wu	7.006169
78187	patentsencyclopedia	5.978584
8277	4490	5.288131
33420	d2nns	4.318528
89966	rurally	4.154242
30523	conflation	4.104860
1339	0800	4.078258
17561	apnm	3.945404
189	0001f914	3.833906
102253	the_purge	3.825181
47806	gating	3.717765
48948	globalist	3.717126
79069	periodicity	3.697982
134	0001f602	3.692670
85380	raf	3.652455
38921	dynamism	3.625575
73032	ngc	3.598444
88580	rg	3.470108
80466	pitbulls	3.438484
28483	cite_note	3.380403

In [29]:
```
pd.DataFrame({'feature name': feature_names,
              'coefficient': coef.ravel().tolist()}).nsmallest(20,'coeff
icient')
```

Out[29]:

	feature name	coefficient
58581	iâ	-6.648753
37765	donâ	-6.265144
58501	itâ	-6.210470
113699	½ï	-5.577377
102173	thatâ	-5.484909
113045	youâ	-5.162987
36057	didnâ	-5.086880
37557	doesnâ	-4.979570
25855	canâ	-4.872234
102622	theyâ	-4.764307
58331	isnâ	-4.708644
110695	weâ	-4.423216
18044	arenâ	-4.344898
102508	thereâ	-4.247995
110063	wasnâ	-4.214284
111990	wouldnâ	-4.194286
110790	whatâ	-3.725483
51676	havenâ	-3.592163
111721	wonâ	-3.278747
31783	couldnâ	-3.263693

Use TfidfVectorizer to manipulate 'body' column

In [100]:
```
tfidf = TfidfVectorizer(analyzer="char_wb",ngram_range=(2,3),stop_words=
'english',token_pattern=r"\b\w+\b")
train = tfidf.fit_transform(text_train_lst)
train
```

Out[100]:
```
<167529x77522 sparse matrix of type '<class 'numpy.float64'>'
        with 39879417 stored elements in Compressed Sparse Row format>
```

In [101]:
```
text_train['top20words'] = text_train.loc[:,'body'].str.contains('wu|pat
entsencyclopedia|d2nns|rurally|conflation|apnm|the_purge|gating|globalis
t|periodicity|raf|dynamism|ngc|rg|pitbulls|cite_note')
```

In [102]:
```
text_train['!count'] = text_train.loc[:,'body'].str.count('!')
```

In [103]:
```
text_train['syb_count'] = text_train.loc[:,'body'].str.count('&|@|#|\\
$|%|\\*|\\^')
```

In [104]:
```
text_train['html'] = text_train.loc[:,'body'].str.contains('http:|html')
text_train['dirtywords'] = text_train.loc[:,'body'].str.contains('fuck|s
hit|damn|bitches')
text_train['@'] = text_train.loc[:,'body'].str.contains('@')
text_train['!'] = text_train.loc[:,'body'].str.contains('!')
text_train['length'] = text_train.loc[:,'body'].str.len()
#text_train = text_train.drop(columns=['body'])
```

In [105]:
```
text_train.head()
```

Out[105]:

	body	top20words	!count	syb_count	html	dirtywords	@	!	length
0	I've always been taught it emerged from the ea...	True	0	0	False	False	False	False	125
1	As an ECE, my first feeling as "HEY THAT'S NOT...	False	0	0	False	False	False	False	229
2	Monday: Drug companies stock dives on good new...	False	0	0	False	False	False	False	61
3	i learned that all hybrids are unfertile i won...	False	0	0	False	False	False	False	139
4	Well i was wanting to get wasted tonight. Not...	False	0	0	False	False	False	False	84

In [109]:
```
text_train = text_train.drop(columns=['body'])
```

In [108]:
```
text_train[['html','dirtywords','@','!','top20words']] = (
    text_train[['html','dirtywords','@','!','top20words']]  == True).ast
ype(int)
#text_train = coo_matrix(text_train)
```

Scenario1: Combine 'body' column and dereived features

In [85]:
```
train = hstack([train,text_train])
```

In [86]:
```
param_grid = {"C": [1000,100,10, 1, 0.1, 0.01]}
grid = GridSearchCV(LogisticRegression(),param_grid=param_grid, cv=5, sc
oring="f1_macro" )
```

In [87]:
```
grid.fit(train,y_train)
```

Out[87]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [1000, 100, 10, 1, 0.1, 0.01]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [90]:
```
grid.best_params_
```

Out[90]:
```
{'C': 1}
```

In [91]:
```
grid.best_score_
```

Out[91]:
```
0.6829471558880378
```

Scenario 2: do not include 'body' column, only use derived feature

In [111]:
```
text_train.head()
```

Out[111]:

	top20words	!count	syb_count	html	dirtywords	@	!	length
0	1	0	0	0	0	0	0	125
1	0	0	0	0	0	0	0	229
2	0	0	0	0	0	0	0	61
3	0	0	0	0	0	0	0	139
4	0	0	0	0	0	0	0	84

In [112]:
```
param_grid = {"C": [1000,100,10, 1, 0.1, 0.01]}
grid = GridSearchCV(LogisticRegression(),param_grid=param_grid, cv=5, sc
oring="f1_macro" )
```

In [116]:
```
grid.fit(text_train,y_train)
```

Out[116]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [1000, 100, 10, 1, 0.1, 0.01]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [117]:
```
grid.best_params_
```

Out[117]:
```
{'C': 1000}
```

In [118]:
```
grid.best_score_
```

Out[118]:
```
0.4541557516222559
```

Use Gradient Boosting

In [119]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [120]:
```
param_grid = {"learning_rate": [1, 0.5, 0.25, 0.1, 0.05, 0.01]}
grid_gbdt = GridSearchCV(GradientBoostingClassifier(),param_grid=param_g
rid, scoring="f1_macro" )
```

In [121]:
```
grid_gbdt.fit(text_train,y_train)
```

Out[121]:
```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', i
nit=None,
                  learning_rate=0.1, loss='deviance', max_depth=3,
                  max_features=None, max_leaf_nodes=None,
                  min_impurity_decrease=0.0, min_impurity_split=None,
                  min_samples_leaf=1, min_sampl...      subsample=1.0, tol=
0.0001, validation_fraction=0.1,
                  verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'learning_rate': [1, 0.5, 0.25, 0.1, 0.05, 0.01]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [122]:
```
grid_gbdt.best_params_
```

Out[122]:
```
{'learning_rate': 1}
```

In [123]:
```
grid_gbdt.best_score_
```

Out[123]:
```
0.5922428955492742
```

Comments:

'Body' column is important for classificantion when use logistic regression; If we drop body column and only use derived features, then gradient boosting perform much better than logistic regression for this problem.

Task 2 Word Vector

In [92]:
```
for i in range(0,len(text_train_lst)):
    text_train_lst[i] = preprocessing.remove_stopwords(text_train_lst[i
])
    text_train_lst[i] = preprocessing.strip_punctuation(text_train_lst[i
])
```

In [93]:
```
texts = [[token for token in re.split('\W+',doc)] for doc in text_train_
lst]
```

Word2vec pre-trained

In [89]:
```
#use the GoogleNew dictionary
w = models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negativ
e300.bin', binary=True)
```

In [94]:
```
#deal with the words not in w
for i in range(0, len(texts)):
    for j in range(0,len(texts[i])):
        if texts[i][j] not in w.vocab:
            texts[i][j] = 'unknown'
```

In [95]:
```
 X_train = np.vstack([np.mean(w[doc], axis=0) for doc in texts])
```

In [104]:
```
param_grid = {"C": [10, 1, 0.1, 0.01, 0.001]}
grid_w2v = GridSearchCV(LogisticRegression(),param_grid=param_grid, cv=5
, scoring="f1_macro")
```

In [105]:
```
grid_w2v.fit(X_train, y_train)
```

Out[105]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [10, 1, 0.1, 0.01, 0.001]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [106]:
```
grid_w2v.best_params_
```

Out[106]:
```
{'C': 10}
```

In [107]:
```
grid_w2v.best_score_
```

Out[107]:
```
0.6397384179079968
```

Fasttext pre-trained crawl

In [171]:
```
#use the crawl-300d-2M dictionary, fasetext pre-trained
#https://github.com/RaRe-Technologies/gensim/issues/814
w2 = models.KeyedVectors.load_word2vec_format('crawl-300d-2M.vec', binar
y=False)
```

In [182]:
```
texts = [[token for token in re.split('\W+',doc)] for doc in text_train_
lst]
```

In [184]:
```
#deal with the words not in w2
for i in range(0, len(texts)):
    for j in range(0,len(texts[i])):
        if texts[i][j] not in w2.vocab:
            texts[i][j] = 'unknown'
```

In [185]:
```
 X_train = np.vstack([np.mean(w2[doc], axis=0) for doc in texts])
```

In [186]:
```
grid_w2v.fit(X_train, y_train)
```

Out[186]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [10, 1, 0.1, 0.01, 0.001]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [187]:
```
grid_w2v.best_params_
```

Out[187]:
```
{'C': 10}
```

In [188]:
```
grid_w2v.best_score_
```

Out[188]:
```
0.6568765489096293
```

Fasttext pre-trained wiki-news with subword

In [189]:
```
#use the wikipedia dictionary, fasetext pre-trained
w3 = models.KeyedVectors.load_word2vec_format('wiki-news-300d-1M-subwor
d.vec', binary=False)
```

In [190]:
```
texts = [[token for token in re.split('\W+',doc)] for doc in text_train_
lst]
```

In [191]:
```
#deal with the words not in w2
for i in range(0, len(texts)):
    for j in range(0,len(texts[i])):
        if texts[i][j] not in w3.vocab:
            texts[i][j] = 'unknown'
```

In [192]:
```
 X_train = np.vstack([np.mean(w3[doc], axis=0) for doc in texts])
```

In [193]:
```
grid_w2v.fit(X_train, y_train)
```

Out[193]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [10, 1, 0.1, 0.01, 0.001]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_macro', verbose=0)
```

In [194]:
```
grid_w2v.best_params_
```

Out[194]:
```
{'C': 10}
```

In [195]:
```
grid_w2v.best_score_
```

Out[195]:
```
0.6374199796830509
```

Comments

They tend to not have significant improvement for classification.