AML_HW5 2

April 16, 2018

#
AML HW 5
####
April 16, 2018
Jingyu Ren UNI:jr3738
Chi Zhang UNI: cz2481

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score
from scipy.sparse import hstack
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
```

1 Task 1

In [2]:
```
train = pd.DataFrame.from_csv('/Users/jingyu/Desktop/hw5_data_train.csv',index_col=None
test = pd.DataFrame.from_csv('/Users/jingyu/Desktop/hw5_data_test.csv',index_col=None)

y_train = train['Recommended']
y_test = test['Recommended']

#1) Use the title only
vect = CountVectorizer()
title_train = vect.fit_transform(train['Title'])
name1 = vect.get_feature_names()

#2) Use the review body only
vect = CountVectorizer()
review_train = vect.fit_transform(train['Review'])
name2 = vect.get_feature_names()
```

```
#3) Concatenate the title and review to a single text and analyze that (discarding the
vect = CountVectorizer()
titleReview_train = vect.fit_transform(train['Title'].map(str) + ' ' + train['Review']]
name3 = vect.get_feature_names()
titleReview_test = test['Title'].map(str) + ' ' + test['Review']

#4) Vectorizing title and review individually and concatenating the vector representat
vect1 = CountVectorizer()
vect2 = CountVectorizer()
title_review_train = hstack((vect1.fit_transform(train['Title']),vect2.fit_transform(tr
name4 = name1 + name2
```

```
/Users/jingyu/anaconda/envs/python3/lib/python3.5/site-packages/ipykernel_launcher.py:1: Future
  """Entry point for launching an IPython kernel.
/Users/jingyu/anaconda/envs/python3/lib/python3.5/site-packages/ipykernel_launcher.py:2: Future
```

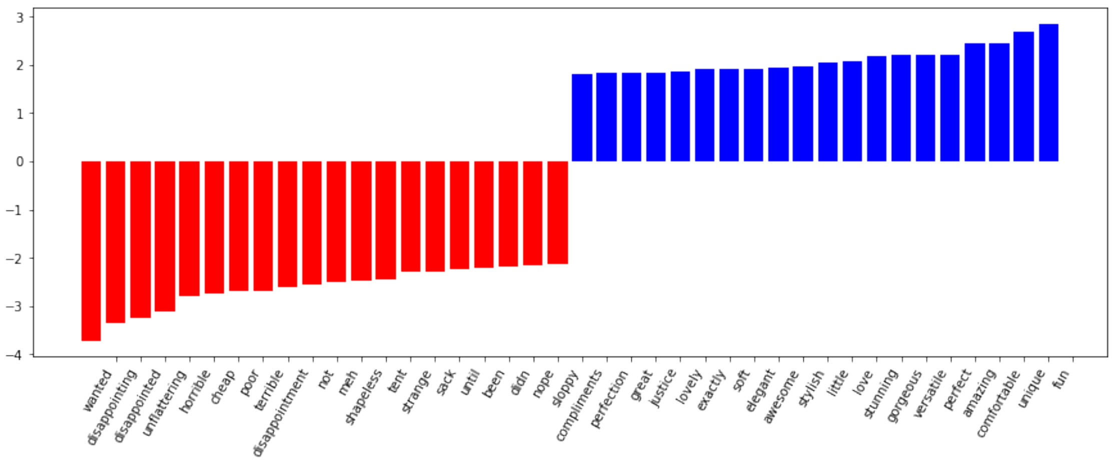
In [176]:
```
pipe1 = make_pipeline(CountVectorizer(), LogisticRegression())
print('#1 Title Only cross validation score:\n{}'.format(np.mean(cross_val_score(Log

pipe2 = make_pipeline(CountVectorizer(), LogisticRegression())
print('#2 Review Only cross validation score:\n{}'.format(np.mean(cross_val_score(Log

pipe3 = make_pipeline(CountVectorizer(), LogisticRegression())
print('#3 Concatenate Title and Review cross validation score:\n{}'.format(np.mean(cr

print('#4 Concatenate Title and Review Vectors cross validation score:\n{}'.format(np
```

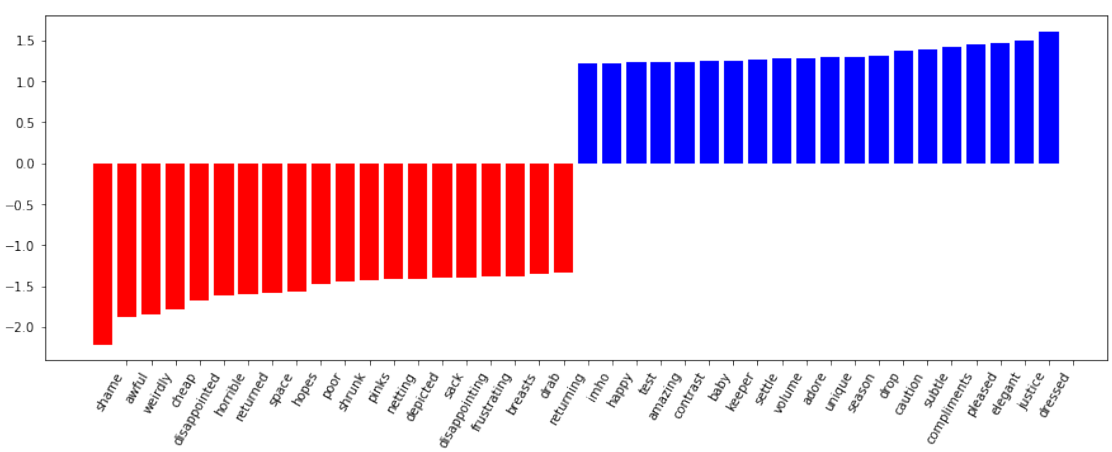
```
#1 Title Only cross validation score:
0.920447284709158
#2 Review Only cross validation score:
0.9119736515693934
#3 Concatenate Title and Review cross validation score:
0.933310639278875
#4 Concatenate Title and Review Vectors cross validation score:
0.9383480297268004
```

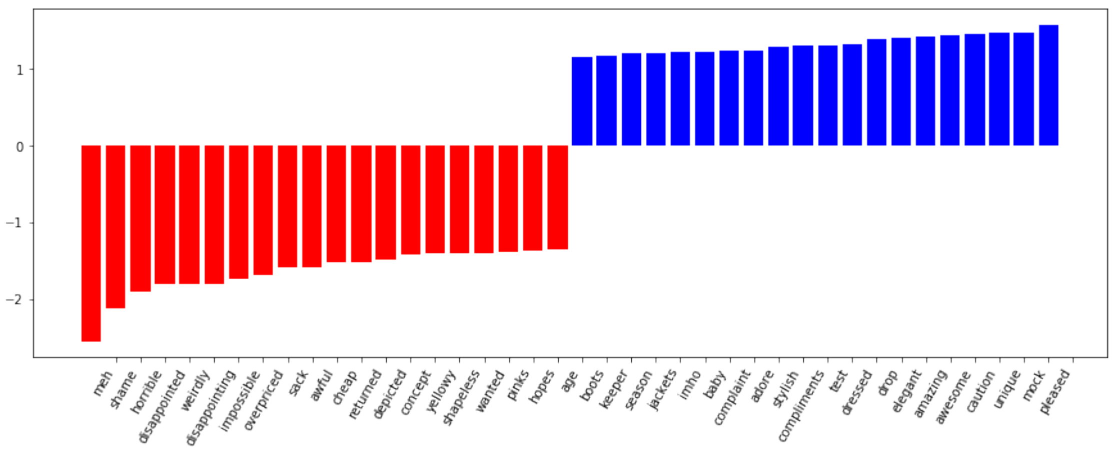
In [8]:
```
def plot_coefficients(classifier, feature_names, top_features=20):
    coef = classifier.coef_.ravel()
    top_positive_coefficients = np.argsort(coef)[-top_features:]
    top_negative_coefficients = np.argsort(coef)[:top_features]
    top_coefficients = np.hstack([top_negative_coefficients, top_positive_coefficients]
    plt.figure(figsize=(15, 5))
    colors = ['red' if c < 0 else 'blue' for c in coef[top_coefficients]]
    plt.bar(np.arange(2 * top_features), coef[top_coefficients], color=colors)
    feature_names = np.array(feature_names)
```

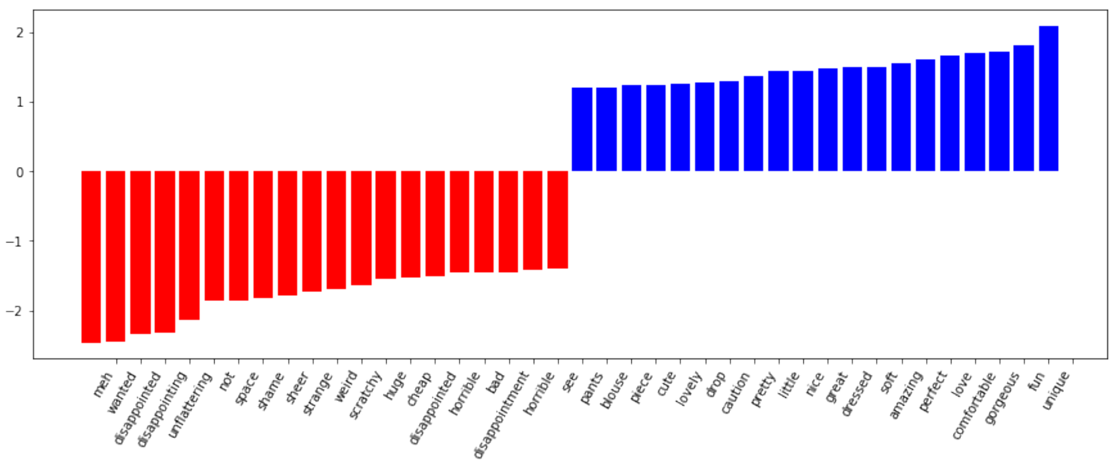
```
    plt.xticks(np.arange(1, 1 + 2 * top_features), feature_names[top_coefficients], ro
    plt.show()
```

```
In [182]: xList=[title_train, review_train, titleReview_train, title_review_train]
          names=[name1,name2,name3,name4]

          for i in range(4):
              lr = LogisticRegression()
              lr.fit(xList[i], y_train)
              plot_coefficients(lr, names[i])
              i = i + 1
```









1.1 GridSearch to tune the regularization parameter

```
In [301]: y_train = train['Recommended']
          y_test = test['Recommended']

          #1) Use the title only
          title_train = train['Title']
          title_test = test['Title']

          #2) Use the review body only
          review_train = train['Review']
          review_test = test['Review']

          #3) Concatenate the title and review to a single text and analyze that (discarding t
```

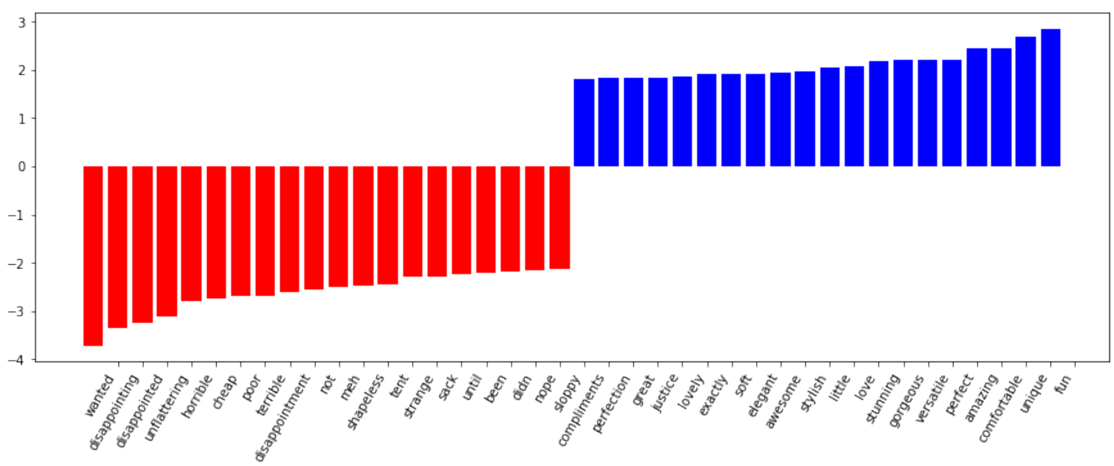
```
titleReview_train = train['Title'].map(str) + ' ' + train['Review']
titleReview_test = test['Title'].map(str) + ' ' + test['Review']

#4) Vectorizing title and review individually and concatenating the vector represent
vect1 = CountVectorizer()
vect2 = CountVectorizer()
title_review_train = hstack((vect1.fit_transform(train['Title']),vect2.fit_transform
title_review_test = hstack((vect1.transform(test['Title']),vect2.transform(test['Rev
```

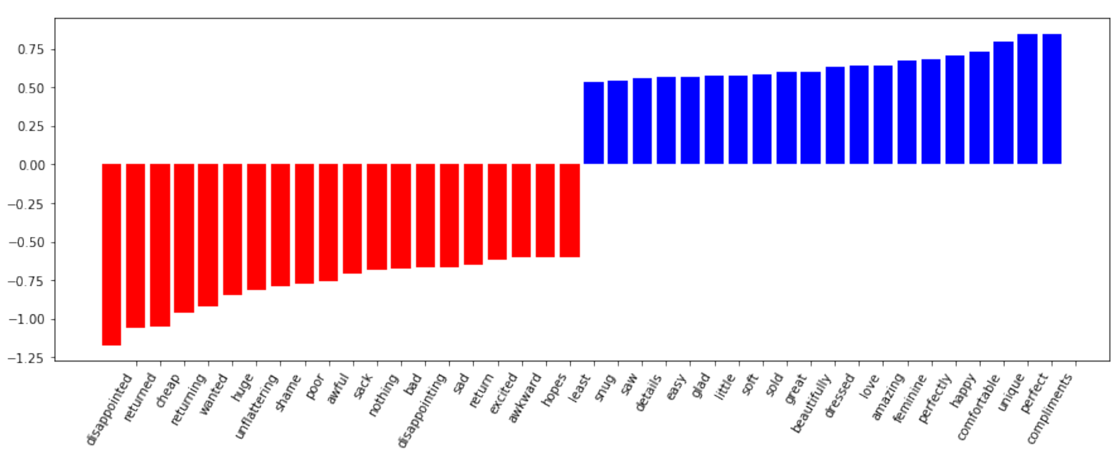
In [303]:
```
grid = {'logisticregression__C': [0.01, 0.1, 1, 10, 100]}
pipe = make_pipeline(CountVectorizer(),LogisticRegression())
gs = GridSearchCV(pipe, grid, scoring='roc_auc', cv=5)
gs.fit(title_train, y_train)
print ('#1 best_parameter_:', gs.best_params_)
print ('#1 best_cv_score_:', gs.best_score_)
print ('#1 predict_score_:', gs.score(title_test, y_test))

grid = {'logisticregression__C': [0.01, 0.1, 1, 10, 100]}
pipe = make_pipeline(CountVectorizer(),LogisticRegression())
gs = GridSearchCV(pipe, grid, scoring='roc_auc', cv=5)
gs.fit(review_train, y_train)
print ('#2 best_parameter_:', gs.best_params_)
print ('#2 best_cv_score_:', gs.best_score_)
print ('#2 predict_score_:', gs.score(review_test, y_test))

grid = {'logisticregression__C': [0.01, 0.1, 1, 10, 100]}
pipe = make_pipeline(CountVectorizer(),LogisticRegression())
gs = GridSearchCV(pipe, grid, scoring='roc_auc', cv=5)
gs.fit(titleReview_train, y_train)
print ('#3 best_parameter_:', gs.best_params_)
print ('#3 best_cv_score_:', gs.best_score_)
print ('#3 predict_score_:', gs.score(titleReview_test, y_test))

grid = {'C': [0.01, 0.1, 1, 10, 100]}
lr = LogisticRegression()
gs = GridSearchCV(lr, grid, scoring='roc_auc', cv=5)
gs.fit(title_review_train, y_train)
print ('#4 best_parameter_:', gs.best_params_)
print ('#4 best_score_:', gs.best_score_)
print ('#4 predict_score_:', gs.score(title_review_test, y_test))
```

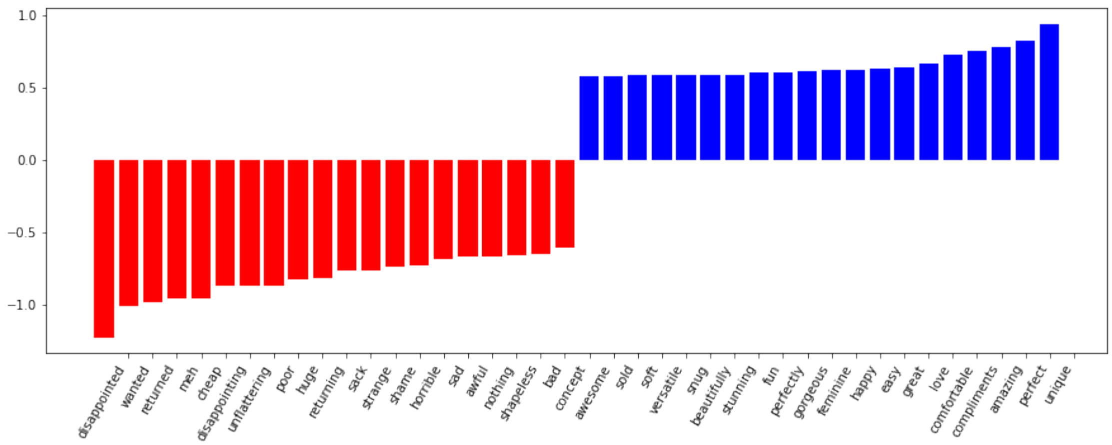
```
#1 best_parameter_: {'logisticregression__C': 1}
#1 best_cv_score_: 0.9204469952584714
#1 predict_score_: 0.9214984071094153
#2 best_parameter_: {'logisticregression__C': 0.1}
#2 best_cv_score_: 0.923761847972498
#2 predict_score_: 0.9221422856845667
#3 best_parameter_: {'logisticregression__C': 0.1}
```

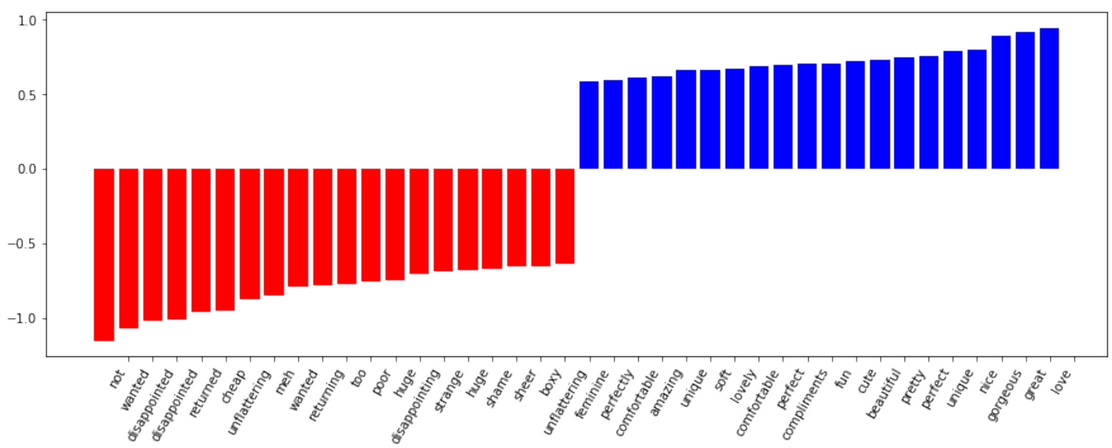
```
#3 best_cv_score_: 0.942410344658611
#3 predict_score_: 0.9388955774186873
#4 best_parameter_: {'C': 0.1}
#4 best_score_: 0.9436032417207872
#4 predict_score_: 0.9431204167740951
```

In [184]:
```python
xList=[title_train, review_train, titleReview_train, title_review_train]
names=[name1,name2,name3,name4]
params=[1,0.1,0.1,0.1]

for i in range(4):
    lr = LogisticRegression(C=params[i])
    lr.fit(xList[i], y_train)
    plot_coefficients(lr, names[i])
    i = i + 1
```









1.1.1 Based on the above results, the fourth way is best.

2 Task 2

2.1 2.1 TfidfVectorizer

```
In [5]: titleReview_train = train['Title'].map(str) + ' ' + train['Review']
        titleReview_val = test['Title'].map(str) + ' ' + test['Review']
```

```
In [15]: param_grid_log = { 'logisticregression__C': [0.01,0.1,1,10]}
         pipe_log_vect = make_pipeline(CountVectorizer(), LogisticRegression(), memory="cache_
         grid_log_vect = GridSearchCV(pipe_log_vect, param_grid_log, scoring='roc_auc',cv=5)
         grid_log_vect.fit(titleReview_train, y_train)
```

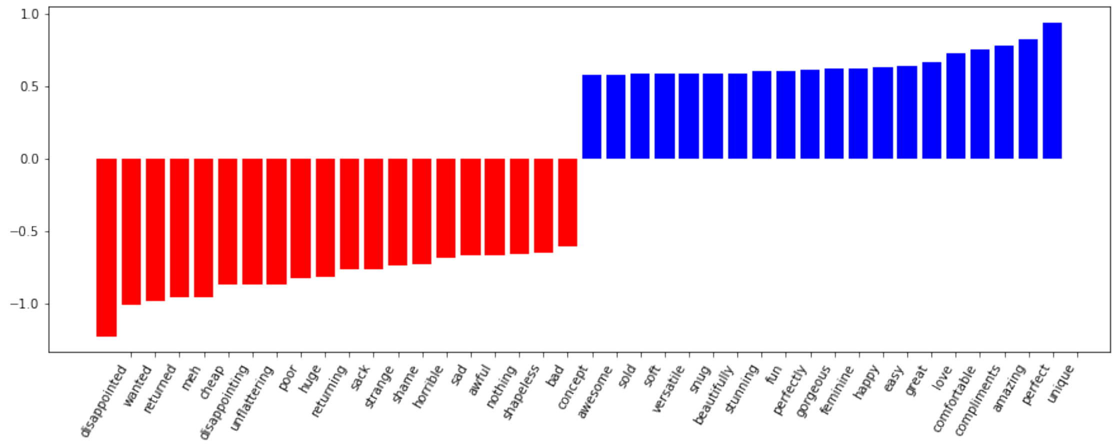
```
Out[15]: GridSearchCV(cv=5, error_score='raise',
                estimator=Pipeline(memory='cache_folder',
```

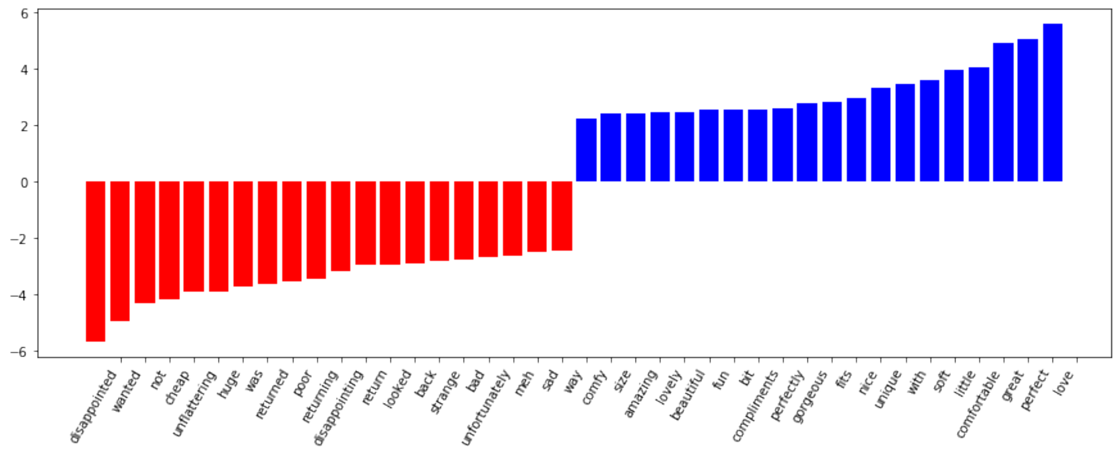
```
steps=[('countvectorizer', CountVectorizer(analyzer='word', binary=False, decode_
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
...ty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False))]),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'logisticregression__C': [0.01, 0.1, 1, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

```
In [16]: param_tfid = { 'logisticregression__C': [0.01, 0.1, 1, 10, 100]}
         pipe_tfid = make_pipeline(TfidfVectorizer(), LogisticRegression(), memory="cache_folde
         grid_tfid = GridSearchCV(pipe_tfid, param_tfid, scoring='roc_auc', cv=5)
         grid_tfid.fit(titleReview_train, y_train)
         print ('grid_log_vect best_score_:', grid_log_vect.best_score_)
         print ('grid_log_vect predict_score_:', grid_log_vect.score(titleReview_test, y_test))
         print ('grid_tfid best_parameter_:', grid_tfid.best_params_)
         print ('grid_tfid best_score_:', grid_tfid.best_score_)
         print ('grid_tfid predict_score_:', grid_tfid.score(titleReview_test, y_test))
```

```
grid_log_vect best_score_: 0.942410344658611
grid_log_vect predict_score_: 0.9388955774186873
grid_tfid best_parameter_: {'logisticregression__C': 1}
grid_tfid best_score_: 0.9498551621199149
grid_tfid predict_score_: 0.9454635538445706
```

```
In [19]: plot_coefficients(grid_log_vect.best_estimator_.named_steps['logisticregression'], gri
         plot_coefficients(grid_tfid.best_estimator_.named_steps['logisticregression'], grid_ti
```





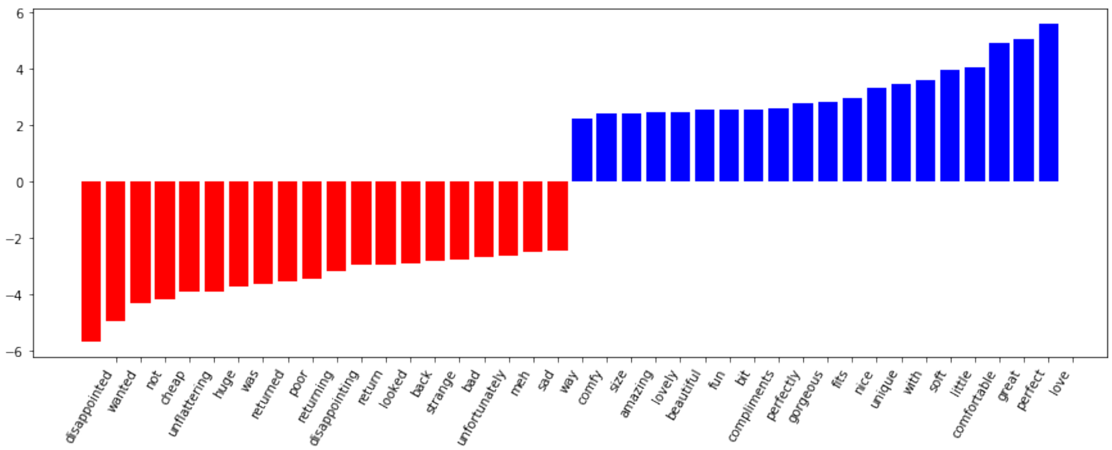
Compared with part 1.3, using tfidfvectorizer, the cv score on training set has been improved from 0.9424 to 0.9499. The predict score has benn improve from 0.9389 to 0.9456. Also the important features with related coefficients changed. For example, the new words has high weight such as love, fits and so on.

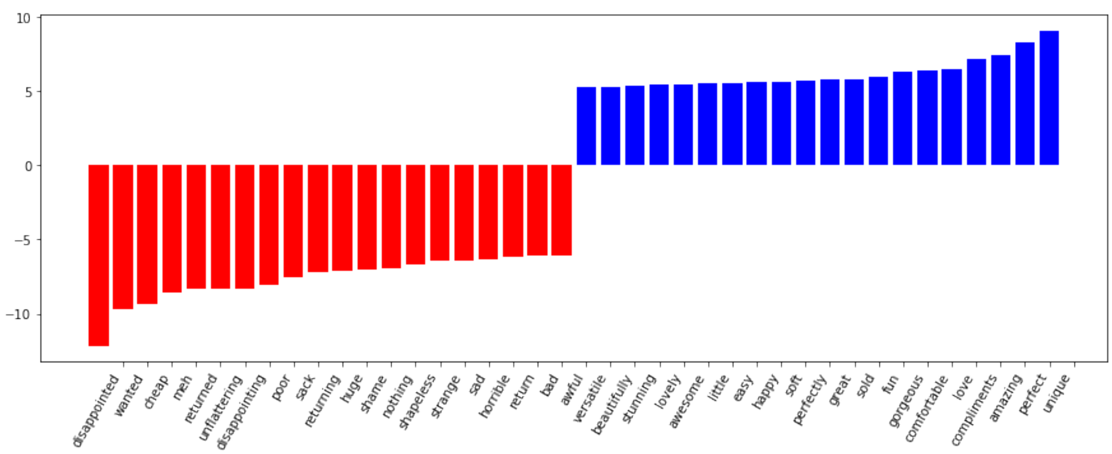
2.2 2.2 Normalizer with CountVectorizer

In [17]:
```
param_norm = { 'logisticregression__C': [0.01, 0.1, 1, 10, 100]}
pipe_norm = make_pipeline(CountVectorizer(), Normalizer(), LogisticRegression(), memo
grid_norm = GridSearchCV(pipe_norm, param_norm, scoring='roc_auc', cv=5)
grid_norm.fit(titleReview_train, y_train)
print ('grid_tfid best_score_:', grid_tfid.best_score_)
print ('grid_tfid predict_score_:', grid_tfid.score(titleReview_test, y_test))
print ('grid_norm best_parameter_:', grid_norm.best_params_)
print ('grid_norm best_score_:', grid_norm.best_score_)
print ('grid_norm predict_score_:', grid_norm.score(titleReview_test, y_test))
```

```
grid_tfid best_score_: 0.9498551621199149
grid_tfid predict_score_: 0.9454635538445706
grid_norm best_parameter_: {'logisticregression__C': 10}
grid_norm best_score_: 0.9476031785092168
grid_norm predict_score_: 0.9441320876438387
```

In [20]:
```
plot_coefficients(grid_tfid.best_estimator_.named_steps['logisticregression'], grid_t
plot_coefficients(grid_norm.best_estimator_.named_steps['logisticregression'], grid_n
```





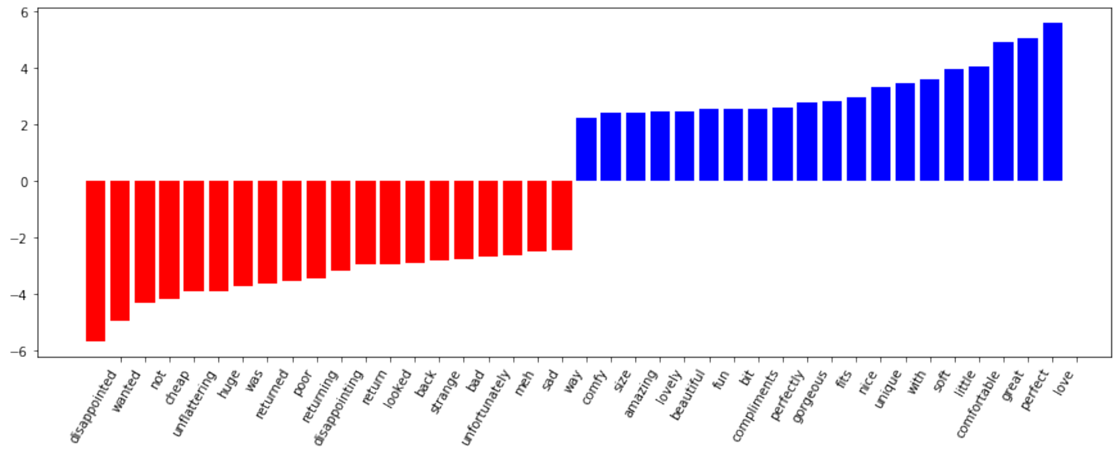
the CV score and predict score didn't improve. But the important feature coef changed.

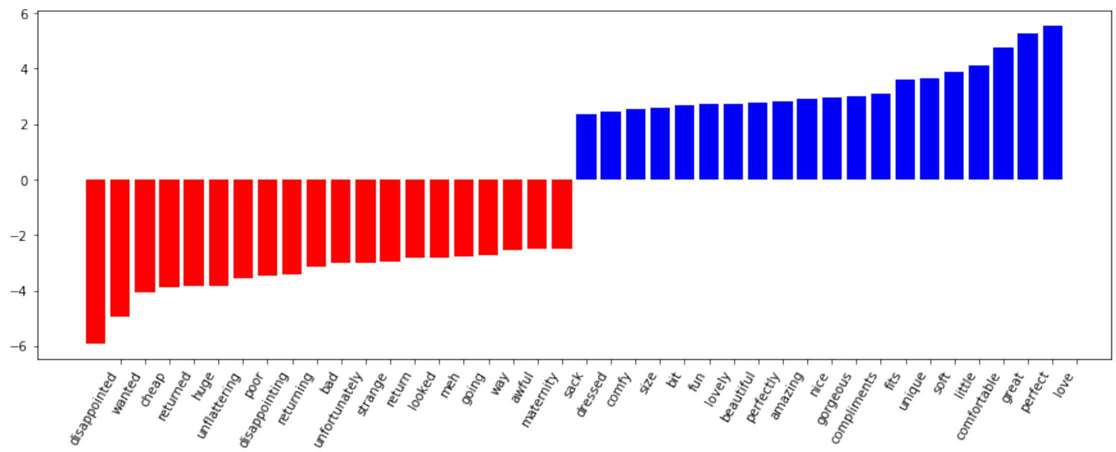
2.3 2.3 Stop-word

In [21]:
```
param_stop = { 'logisticregression__C': [0.01, 0.1, 1, 10, 100]}
pipe_stop = make_pipeline(TfidfVectorizer(stop_words='english'), LogisticRegression()
grid_stop = GridSearchCV(pipe_stop, param_stop, scoring='roc_auc', cv=5)
grid_stop.fit(titleReview_train, y_train)
print ('grid_tfid best_score_:', grid_tfid.best_score_)
print ('grid_tfid predict_score_:', grid_tfid.score(titleReview_test, y_test))
print ('grid_stop best_parameter_:', grid_stop.best_params_)
print ('grid_stop best_score_:', grid_stop.best_score_)
print ('grid_stop predict_score_:', grid_stop.score(titleReview_test, y_test))
```

```
grid_tfid best_score_: 0.9498551621199149
grid_tfid predict_score_: 0.9454635538445706
grid_stop best_parameter_: {'logisticregression__C': 1}
grid_stop best_score_: 0.9434995623247132
grid_stop predict_score_: 0.9397407112735047
```

```
In [22]: plot_coefficients(grid_tfid.best_estimator_.named_steps['logisticregression'], grid_t
         plot_coefficients(grid_stop.best_estimator_.named_steps['logisticregression'], grid_s
```





the CV score and predict score didn't improve. But the important feature coef didn't change.

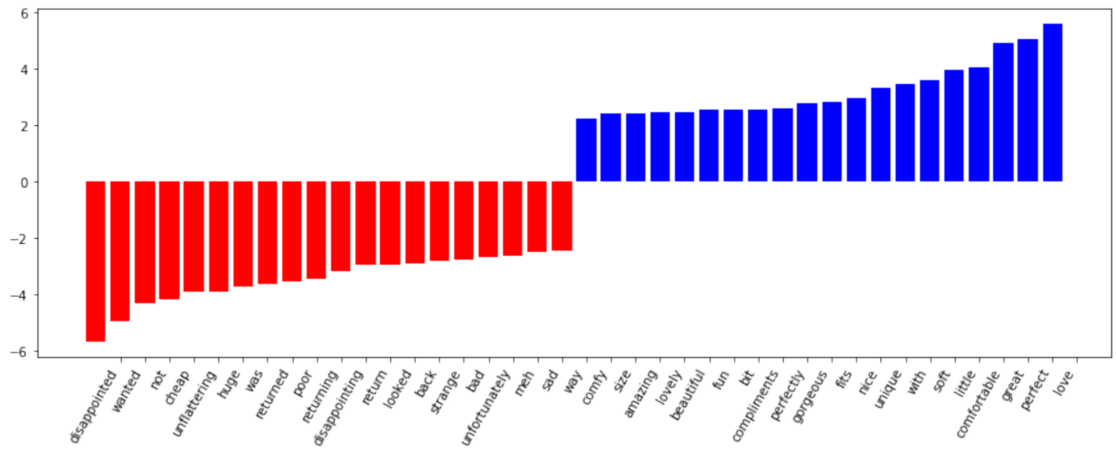
2.4 2.4 min_df

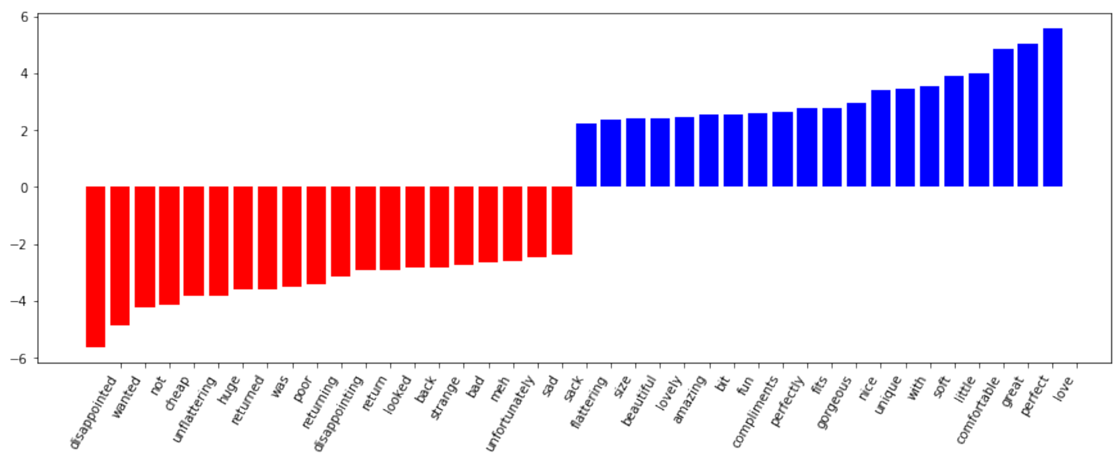
```
In [23]: param4min = { 'logisticregression__C': [0.01, 0.1, 1, 10, 100]}
         pipe4min = make_pipeline(TfidfVectorizer(min_df=4), LogisticRegression(), memory="cacl
```

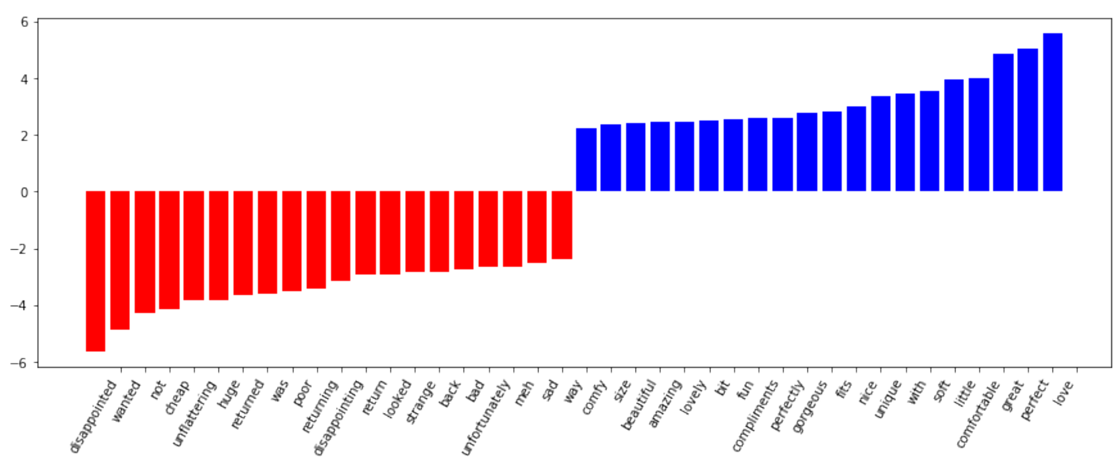
```
grid4min = GridSearchCV(pipe4min, param4min, scoring='roc_auc', cv=5)
grid4min.fit(titleReview_train, y_train)
param3min = { 'logisticregression__C': [0.01, 0.1, 1, 10, 100]}
pipe3min = make_pipeline(TfidfVectorizer(min_df=3), LogisticRegression(), memory="cacl
grid3min = GridSearchCV(pipe3min, param3min, scoring='roc_auc', cv=5)
grid3min.fit(titleReview_train, y_train)
param2min = { 'logisticregression__C': [0.01, 0.1, 1, 10, 100]}
pipe2min = make_pipeline(TfidfVectorizer(min_df=2), LogisticRegression(), memory="cacl
grid2min = GridSearchCV(pipe2min, param2min, scoring='roc_auc', cv=5)
grid2min.fit(titleReview_train, y_train)
print ('grid_tfid best_score_:', grid_tfid.best_score_)
print ('grid_tfid predict_score_:', grid_tfid.score(titleReview_test, y_test))
print ('grid4min best_parameter_:', grid4min.best_params_)
print ('grid4min best_score_:', grid4min.best_score_)
print ('grid4min predict_score_:', grid4min.score(titleReview_test, y_test))
print ('grid3min best_parameter_:', grid3min.best_params_)
print ('grid3min best_score_:', grid3min.best_score_)
print ('grid3min predict_score_:', grid3min.score(titleReview_test, y_test))
print ('grid2min best_parameter_:', grid2min.best_params_)
print ('grid2min best_score_:', grid2min.best_score_)
print ('grid2min predict_score_:', grid2min.score(titleReview_test, y_test))
```

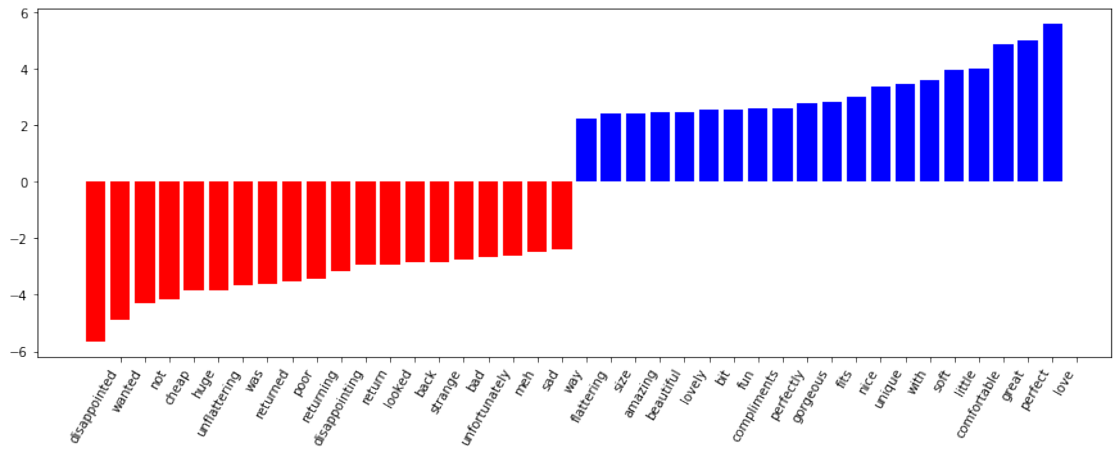
```
grid_tfid best_score_: 0.9498551621199149
grid_tfid predict_score_: 0.9454635538445706
grid4min best_parameter_: {'logisticregression__C': 1}
grid4min best_score_: 0.9500145082145014
grid4min predict_score_: 0.9454760026247561
grid3min best_parameter_: {'logisticregression__C': 1}
grid3min best_score_: 0.950026395977515
grid3min predict_score_: 0.9455927445189405
grid2min best_parameter_: {'logisticregression__C': 1}
grid2min best_score_: 0.9500371582729809
grid2min predict_score_: 0.9455227547103419
```

In [24]:
```
plot_coefficients(grid_tfid.best_estimator_.named_steps['logisticregression'], grid_t:
plot_coefficients(grid4min.best_estimator_.named_steps['logisticregression'], grid4mi
plot_coefficients(grid3min.best_estimator_.named_steps['logisticregression'], grid3mi
plot_coefficients(grid2min.best_estimator_.named_steps['logisticregression'], grid2mi
```


6
4
2
0
−2
−4
−6
disappointed
wanted
not
cheap
unflattering
huge
was
returned
poor
returning
disappointing
return
looked
back
strange
bad
unfortunately
meh
sad
way
comfy
size
amazing
lovely
beautiful
fun
bit
compliments
perfectly
gorgeous
fits
nice
unique
with
soft
little
comfortable
great
perfect
love


6
4
2
0
−2
−4
−6
disappointed
wanted
not
cheap
unflattering
huge
returned
was
poor
returning
disappointing
return
looked
back
strange
bad
meh
unfortunately
sad
sack
flattering
size
beautiful
lovely
amazing
bit
fun
compliments
perfectly
fits
gorgeous
nice
unique
with
soft
little
comfortable
great
perfect
love


6
4
2
0
−2
−4
−6
disappointed
wanted
not
cheap
unflattering
huge
returned
was
poor
returning
disappointing
return
looked
strange
back
bad
unfortunately
meh
sad
way
comfy
size
beautiful
amazing
lovely
bit
fun
compliments
perfectly
gorgeous
fits
nice
unique
with
soft
little
comfortable
great
perfect
love



with min_df, the CV score and predict score slightly improved. But the important feature coef didn't change.

3 Task 3.1

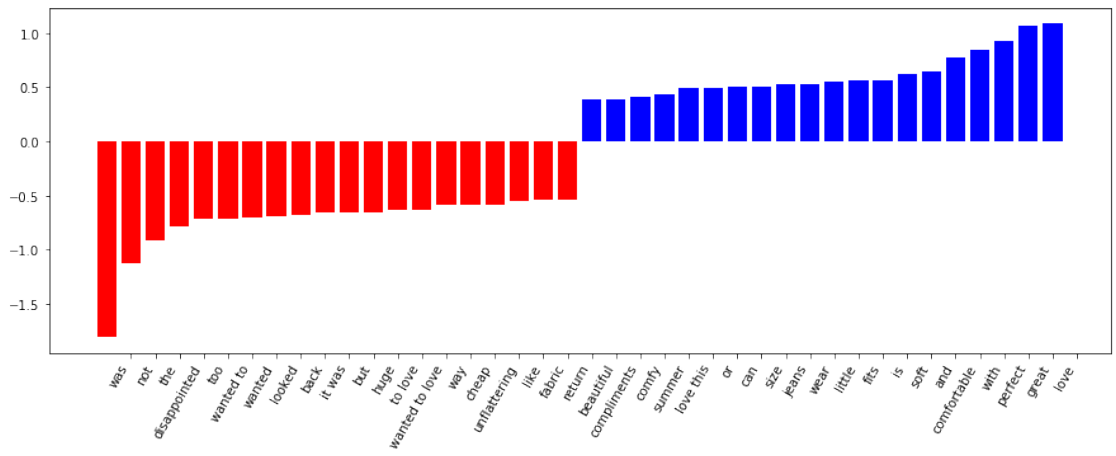
```
In [268]: X_train = train['Title'].map(str) + ' ' + train['Review']
```

```
In [271]: pipe1 = make_pipeline(TfidfVectorizer(ngram_range=(1, 1)),LogisticRegressionCV(), mem
          print(' unigrams score: {}'.format(np.mean(cross_val_score(pipe1, X_train , y_train,

          tfidf = TfidfVectorizer(ngram_range=(1, 4))
          titleReview_train_tfidf = tfidf.fit_transform(train['Title'].map(str) + ' ' + train[
          name = tfidf.get_feature_names()
          pipe3 = make_pipeline(TfidfVectorizer(ngram_range=(1, 4)),LogisticRegressionCV(), mem
          print('(1, 4)grams score: {}'.format(np.mean(cross_val_score(pipe3, X_train, y_train
```

```
 unigrams score: 0.9467156329833519
(1, 4)grams score: 0.9563276559749114
```

Thus, n-grams(here 4-grams) of varying length will give a best performance.

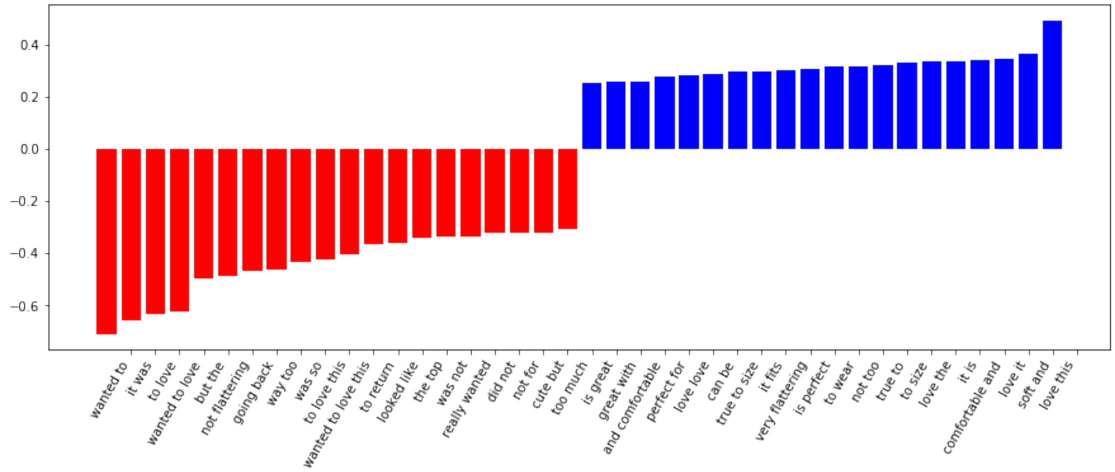
```
In [207]: tfidf = TfidfVectorizer(ngram_range=(1, 4))
          titleReview_train_tfidf = tfidf.fit_transform(train['Title'].map(str) + ' ' + train[
          name = tfidf.get_feature_names()

          lr = LogisticRegression(C=0.1)
          lr.fit(titleReview_train_tfidf, y_train)
          plot_coefficients(lr, name)
```



3.0.1 Draw only non-unigrams

```
In [22]: def plot_higher_coefficients(classifier, feature_names, top_features=20):
             coef = classifier.coef_.ravel()
             ngrams_name=[]
             ngrams_coef=[]
             for name in feature_names:
                 if len(name.split())>1:
                     ngrams_name.append(True)
                 else:
                     ngrams_name.append(False)
             for i in range(len(feature_names)):
                 if ngrams_name[i]==True:
                     ngrams_coef.append(coef[i])
                 else:
                     ngrams_coef.append(0)

             ngrams_coef = np.asarray(ngrams_coef)

             top_positive_coefficients = np.argsort(ngrams_coef)[-top_features:]
             top_negative_coefficients = np.argsort(ngrams_coef)[:top_features]
             top_coefficients = np.hstack([top_negative_coefficients, top_positive_coefficients
             plt.figure(figsize=(15, 5))
             colors = ['red' if c < 0 else 'blue' for c in coef[top_coefficients]]
             plt.bar(np.arange(2 * top_features), coef[top_coefficients], color=colors)
             feature_names = np.array(feature_names)
             plt.xticks(np.arange(1, 1 + 2 * top_features), feature_names[top_coefficients], r
             plt.show()

In [242]: tfidf = TfidfVectorizer(ngram_range=(1, 4))
          titleReview_train_tfidf = tfidf.fit_transform(train['Title'].map(str) + ' ' + train[
```

```
name = tfidf.get_feature_names()

lr = LogisticRegression(C=0.1)
lr.fit(titleReview_train_tfidf, y_train)
plot_higher_coefficients(lr, name)
```

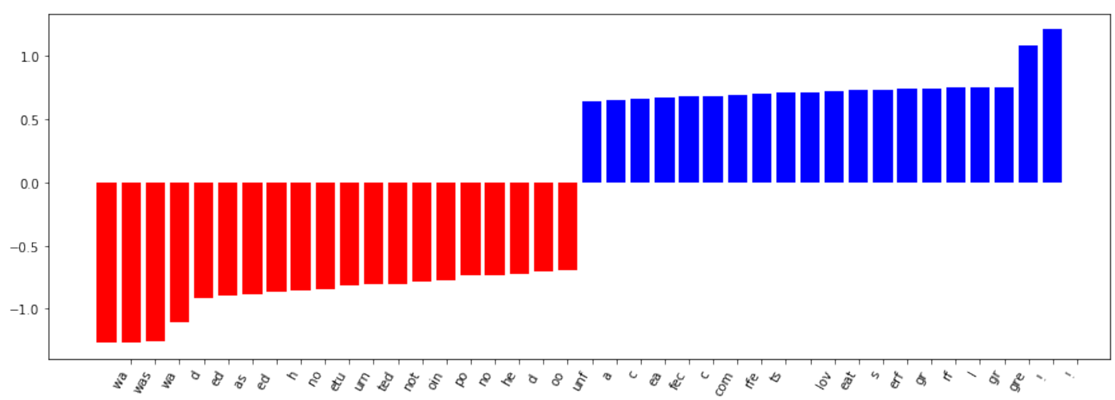


4 Task 3.2

```
In [20]: tfidf = TfidfVectorizer(ngram_range=(1, 3), analyzer="char_wb")
         titleReview_train_tfidf = tfidf.fit_transform(train['Title'].map(str) + ' ' + train['
         titleReview_test_tfidf = tfidf.transform(test['Title'].map(str) + ' ' + test['Review']
         name = tfidf.get_feature_names()
```

```
In [25]: lr = LogisticRegression(C=0.1)
         lr.fit(titleReview_train_tfidf, y_train)
         print('Using character n-grams test score: {}'.format(lr.score(titleReview_test_tfidf
         plot_coefficients(lr, name)
```

```
Using character n-grams test score: 0.8172196214125789
```



From the plot, we can tell that:

1.reviews or titles that have '!' in content, will lead to a recommendation.

2.reviews or titles that have 'was'-like words(such as 'wa' and 'was') in content, will not lead to a recommendation.

3.reviews or titles that have 'no'-like words(such as 'no' and 'not') in content, will not lead to a recommendation.

4.character n-grams's performance is worse than same n-grams of words.

5 Task 3.3

5.0.1 impact of min_df with n-grams

```
In [26]: X_train = train['Title'].map(str) + ' ' + train['Review']
         X_test = test['Title'].map(str) + ' ' + test['Review']
```

```
In [29]: pipe = make_pipeline(TfidfVectorizer(ngram_range=(1, 2), min_df=2), LogisticRegressio
         tfidf = TfidfVectorizer(ngram_range=(1, 2), min_df=2)
         tfidf.fit(X_train,y_train)
         print('(1, 2), min_df=2: {}'.format(len(tfidf.vocabulary_)))
         print('(1, 2), min_df=2 score: {}'.format(np.mean(cross_val_score(pipe, X_train, y_tra
         print('(1, 2), min_df=2 test score: {}'.format(pipe.fit(X_train,y_train).score(X_test

         pipe = make_pipeline(TfidfVectorizer(ngram_range=(1, 2), min_df=4), LogisticRegressio
         tfidf = TfidfVectorizer(ngram_range=(1, 2), min_df=4)
         tfidf.fit(X_train,y_train)
         print("(1, 2), min_df=4: {}".format(len(tfidf.vocabulary_)))
         print('(1, 2), min_df=4 score: {}'.format(np.mean(cross_val_score(pipe, X_train, y_tra
         print('(1, 2), min_df=4 test score: {}'.format(pipe.fit(X_train,y_train).score(X_test
```

```
(1, 2), min_df=2: 70273
(1, 2), min_df=2 score: 0.9520422745830721
(1, 2), min_df=2 test score: 0.9098310604518625
(1, 2), min_df=4: 33764
(1, 2), min_df=4 score: 0.9497980487054856
(1, 2), min_df=4 test score: 0.909220435579076
```

Therefore, increase the value of min_df, the number of feature will decrease; the score will decrease.

5.0.2 impact of stop-words with n-grams

```
In [31]: pipe = make_pipeline(TfidfVectorizer(ngram_range=(1, 2), min_df=4), LogisticRegressio
         tfidf = TfidfVectorizer(ngram_range=(1, 2), min_df=4)
         tfidf.fit(X_train,y_train)
         print("(1, 2), min_df=4: {}".format(len(tfidf.vocabulary_)))
```

```
print('(1, 2), min_df=4 score: {}'.format(np.mean(cross_val_score(pipe, X_train, y_tra
print('(1, 2), min_df=4 test score: {}'.format(pipe.fit(X_train,y_train).score(X_test

pipe = make_pipeline(TfidfVectorizer(ngram_range=(1, 2), min_df=4,stop_words="english"
tfidf = TfidfVectorizer(ngram_range=(1, 2), min_df=4,stop_words="english")
tfidf.fit(X_train,y_train)
print("(1, 2), stopwords, min_df=4 score: {}".format(len(tfidf.vocabulary_)))
print('(1, 2), stopwords, min_df=4 score: {}'.format(np.mean(cross_val_score(pipe, X_t
print('(1, 2), stopwords, min_df=4 test score: {}'.format(pipe.fit(X_train,y_train).sc
```

```
(1, 2), min_df=4: 33764
(1, 2), min_df=4 score: 0.9535003525255963
(1, 2), min_df=4 test score: 0.892937105638103
(1, 2), stopwords, min_df=4 score: 20623
(1, 2), stopwords, min_df=4 score: 0.94595407251938
(1, 2), stopwords, min_df=4 test score: 0.8927335640138409
```

Therefore, apply stop words, the number of feature will decrease; the score will also decrease.

6 Task 4

From task3, we know, when using (1,2) grams and min_df=4the model has a best performance.

```
In [17]: from sklearn.svm import LinearSVC
         from sklearn.linear_model import RidgeClassifier
```

```
In [13]: X_train = train['Title'].map(str) + ' ' + train['Review']
         X_test = test['Title'].map(str) + ' ' + test['Review']
```

```
In [16]: pipe = make_pipeline(TfidfVectorizer(ngram_range=(1, 2), min_df=4), LogisticRegressio
         print('L1 cv score: {}'.format(np.mean(cross_val_score(pipe, X_train, y_train, cv=5, s
         print('L1 test score: {}'.format(pipe.fit(X_train, y_train).score(X_test, y_test)))
```

```
L1 cv score: 0.9448186145300251
L1 test score: 0.8966008548748219
```

```
In [18]: pipe = make_pipeline(TfidfVectorizer(ngram_range=(1, 2), min_df=4),RidgeClassifier())
         print('L2 cv score: {}'.format(np.mean(cross_val_score(pipe, X_train, y_train, cv=5, s
         print('L2 test score: {}'.format(pipe.fit(X_train, y_train).score(X_test, y_test)))
```

```
L2 cv score: 0.9537612594021473
L2 test score: 0.9102381437003867
```

```
In [19]: pipe = make_pipeline(TfidfVectorizer(ngram_range=(1, 2), min_df=4),LinearSVC())
         print('LinearSVC cv score: {}'.format(np.mean(cross_val_score(pipe, X_train, y_train,
         print('LinearSVC test score: {}'.format(pipe.fit(X_train, y_train).score(X_test, y_tes
```

```
LinearSVC cv score: 0.9520345621531835
LinearSVC test score: 0.9118664766944841
```

From above models results:
1.L1 model has largest variance and bias.
2.LinearSVC has lowest variance and bias.
3.L2 model has a slightly worse performance than LinearSVC.

6.0.1 Other features

Beside using n-grams, we also could use:
1.sentiment score of the reviews and tiltes
2.Length of text
3.Number of out-of-vocabularly words
4.Presence / frequency of ALL CAPS
5.Lemmatization

```
In [ ]:
```