In [42]:

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score, cross_validate
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```python
## (i) 5-fold cross validation on training.txt
#
# To determine optimal configurations for each classifier, I collected mean accuracy scores from
# running 5-fold cross validation.  Once I arrived at an optimal setting on this basis, I got
# the precision, recall, and f1 scores at that setting.
# I think this is reasonable considering there is an even distribution of ratings across the datasets--
# mean that of 10000 total examples in the dataset, 2000 have a rating of 1, 2000 have a rating of 2, and so on..
```

In [ ]:

```python
# MultinomialNB
#
# No parameters were specified to change from project instructions...
```

In [23]:

```python
nb = MultinomialNB()
```

In [24]:

```python
results = cross_validate(nb, train_tfidf, train_target, cv=5, return_train_score=False, scoring=('precision_macro', 'recall_macro', 'f1_macro'))
for key in ['test_precision_macro', 'test_recall_macro', 'test_f1_macro']:
    print('{0: <22}: {1}'.format(key.upper(), results[key].mean()))
```

```
TEST_PRECISION_MACRO  : 0.42504131194283434
TEST_RECALL_MACRO     : 0.3913
TEST_F1_MACRO         : 0.389894922172691
```

In [ ]:

In [ ]:

In [ ]:

```
# LogisticRegression
#
# tried L1 and L2 regularization with C parameters 0.001, 0.01, 0.1, 1,
 10, 100...
# L2 regularization gave highest accuracy with C parameter value of 1
```

In [7]:

```
scores = []
for p in ['l1', 'l2']:
    for c in [0.001,0.01,0.1,1,10,100]:
        lr = LogisticRegression(solver='liblinear', multi_class='ovr', p
enalty=p, C=c)
        scores.append(cross_val_score(lr, train_tfidf, train_target, cv=
5).mean())
print(scores)
```

```
[0.2, 0.2, 0.3937, 0.46399999999999997, 0.42610000000000003, 0.4091, 0.
4192, 0.42469999999999997, 0.44880000000000003, 0.4664, 0.4471999999999
9993, 0.42219999999999996]
```

In [8]:

```
lr = LogisticRegression(solver='liblinear', multi_class='ovr', penalty=
'l2', C=1)
precision = cross_val_score(lr, train_tfidf, train_target, cv=5, scoring
='precision_macro').mean()
recall = cross_val_score(lr, train_tfidf, train_target, cv=5, scoring='r
ecall_macro').mean()
f1 = cross_val_score(lr, train_tfidf, train_target, cv=5, scoring='f1_ma
cro').mean()
print('SCORING REPORT\n{}\nAVERAGE PRECISION: {}\nAVERAGE RECALL: {}\nAV
ERAGE F1: {}'.format('='*14, precision, recall ,f1))
```

```
SCORING REPORT
==============
AVERAGE PRECISION: 0.45945328665208207
AVERAGE RECALL: 0.46640000000000004
AVERAGE F1: 0.4569426339889883
```

In [ ]:

In [ ]:

In [ ]:

```
# Neural Network: MLPClassifier
#
# tried 5, 10, 15, 20, and 100 units on 1, 2, 3 hidden layers
#
# 1 hidden layer with 100 units gave the highest accuracy; however...
# some testing shows below that this setting results in folds where some
labels are never predicted,
# resulting in undefined precision, recall, and f1 scores.  The returned
score for those labels is 0,
# bringing the average scores way down.
#
# Therefore, I chose a different setting:
# 3 hidden layers with 5 units per layer gave a high accuracy and did no
t result in undefined scores.
```

In [ ]:

```
from time import time
all_info = ''
for num_hidden_layers in [1, 2, 3]:
    nn_test = MLPClassifier(hidden_layer_sizes=(100, num_hidden_layers))
    start = time()
    score = cross_val_score(nn_test, train_tfidf, train_target, cv=5).mean()
    info = '{} hidden layers, {} units per layer\nAVERAGE SCORE: {} ({} seconds)\n\n'.format(num_hidden_layers, 100, score, time() - start)
    print(info)
    all_info += info
print(all_info)
```

In [ ]:

```
'''
Results:

1 hidden layers, 5 units per layer
AVERAGE SCORE: 0.26189999999999997 (82.26306986808777 seconds)

1 hidden layers, 10 units per layer
AVERAGE SCORE: 0.2612 (171.13859677314758 seconds)

1 hidden layers, 15 units per layer
AVERAGE SCORE: 0.26639999999999997 (225.12408924102783 seconds)

1 hidden layers, 20 units per layer
AVERAGE SCORE: 0.3154 (379.20189094543457 seconds)

2 hidden layers, 5 units per layer
AVERAGE SCORE: 0.3307 (164.73012685775757 seconds)

2 hidden layers, 10 units per layer
AVERAGE SCORE: 0.3281 (220.51628804206848 seconds)

2 hidden layers, 15 units per layer
AVERAGE SCORE: 0.279 (226.49298191070557 seconds)

2 hidden layers, 20 units per layer
AVERAGE SCORE: 0.25170000000000003 (174.7208309173584 seconds)

3 hidden layers, 5 units per layer
AVERAGE SCORE: 0.3698 (136.1033742427826 seconds)

3 hidden layers, 10 units per layer
AVERAGE SCORE: 0.3613 (231.52910900115967 seconds)

3 hidden layers, 15 units per layer
AVERAGE SCORE: 0.30289999999999995 (345.6000349521637 seconds)

3 hidden layers, 20 units per layer
AVERAGE SCORE: 0.3466 (331.38618206977844 seconds)

1 hidden layers, 100 units per layer
AVERAGE SCORE: 0.40950000000000001

2 hidden layers, 100 units per layer
AVERAGE SCORE: 0.2345 (702.7552897930145 seconds)

3 hidden layers, 100 units per layer
AVERAGE SCORE: 0.3698 (1494.6928179264069 seconds)

'''
```

In [50]:

```
clf = MLPClassifier(hidden_layer_sizes=(100, 1)) # showing 1 hidden layer, 100 units, resulting in low score
```

In [51]:

```
cross_val_score(clf, train_tfidf, train_target, cv=5, scoring='precision_macro').mean()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.p
y:1143: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.p
y:1143: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.p
y:1143: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Out[51]:

```
0.19209275852557992
```

In [52]:

```
clf = MLPClassifier(hidden_layer_sizes=(5, 3)) # compare with 3 hidden layers, 5 units
print(cross_val_score(clf, train_tfidf, train_target, cv=5, scoring='precision_macro').mean())
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

0.3965408692188382

/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [53]:

```
results = cross_validate(clf, train_tfidf, train_target, cv=5, return_train_score=False, scoring=('precision_macro', 'recall_macro', 'f1_macro'))
for key in ['test_precision_macro', 'test_recall_macro', 'test_f1_macro']:
    print('{0: <22}: {1}'.format(key.upper(), results[key].mean()))
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

TEST_PRECISION_MACRO  : 0.3863551431751554
TEST_RECALL_MACRO     : 0.3752
TEST_F1_MACRO         : 0.37198399981821356

/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [ ]:

In [ ]:

In [ ]:

```
# AdaBoostClassifier
#
# tried n_estimator values of 25, 50, 75...
# default n_estimator value of 50 gave the best result
```

In [5]:

```
scores = []
for n in [25, 50, 75]:
    ada = AdaBoostClassifier(n_estimators=n)
    scores.append(cross_val_score(ada, train_tfidf, train_target, cv=5).mean())
print(scores)
```

```
[0.40199999999999997, 0.4272, 0.4232]
```

In [8]:

```
clf = AdaBoostClassifier(n_estimators=50)
precision = cross_val_score(clf, train_tfidf, train_target, cv=5, scorin
g='precision_macro').mean()
recall = cross_val_score(clf, train_tfidf, train_target, cv=5, scoring=
'recall_macro').mean()
f1 = cross_val_score(clf, train_tfidf, train_target, cv=5, scoring='f1_m
acro').mean()
print('SCORING REPORT\n{}\nAVERAGE PRECISION: {}\nAVERAGE RECALL: {}\nAV
ERAGE F1: {}'.format('='*14, precision, recall ,f1))
```

```
SCORING REPORT
==============
AVERAGE PRECISION: 0.4245235816378587
AVERAGE RECALL: 0.4272
AVERAGE F1: 0.42239110439415617
```

In [ ]:

In [ ]:

In [ ]:

```
# Support Vector Machine SVC
#
# tried linear, polynomial, rbf, and sigmoid kernels with cost factors
 1, 10, 100, 1000...
# cost factor 1000 with rbf kernel gave highest accuracy
```

In [ ]:

```
for c in [1, 10, 100, 1000]:
    for k in ['linear', 'poly', 'rbf', 'sigmoid']:
        svm = SVC(gamma='auto', C=c, kernel=k)
        result = cross_val_score(svm, train_tfidf, train_target, cv=5).m
ean()
        print('cost factor {}, kernel {}: {}'.format(c, k , result))
```

In [ ]:

```
'''
Results:

cost factor 1, kernel linear: 0.4702
cost factor 1, kernel poly: 0.2688
cost factor 1, kernel rbf: 0.36129999999999995
cost factor 1, kernel sigmoid: 0.36150000000000004
cost factor 10, kernel linear: 0.4372
cost factor 10, kernel poly: 0.2699
cost factor 10, kernel rbf: 0.36129999999999995
cost factor 10, kernel sigmoid: 0.36150000000000004
cost factor 100, kernel linear: 0.42969999999999997
cost factor 100, kernel poly: 0.2697
cost factor 100, kernel rbf: 0.36129999999999995
cost factor 100, kernel sigmoid: 0.36150000000000004
cost factor 1000, kernel linear: 0.43100000000000005
cost factor 1000, kernel poly: 0.2697
cost factor 1000, kernel rbf: 0.4425
cost factor 1000, kernel sigmoid: 0.41200000000000003
'''
```

In [27]:
```
clf = SVC(gamma='auto', C=1000, kernel='rbf')
results = cross_validate(clf, train_tfidf, train_target, cv=5, return_tr
ain_score=False, scoring=('precision_macro', 'recall_macro', 'f1_macro'
))
for key in ['test_precision_macro', 'test_recall_macro', 'test_f1_macro'
]:
    print('{0: <22}: {1}'.format(key.upper(), results[key].mean()))
```

```
TEST_PRECISION_MACRO  : 0.4458132278366751
TEST_RECALL_MACRO     : 0.4425
TEST_F1_MACRO         : 0.4345959450940112
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
## (ii) 5-fold cross validation with additional knowledge into model (fi
lter train data by sentiment words)
#
# Also tested accuracy of these classifiers on filtered test data - see
 bottom of this notebook
```

In [28]:
```
sentiment_words = negative_words + positive_words
filtered_train_data = []
for doc in train_data:
    filtered_word_list = []
    word_list = doc.split()
    for word in word_list:
        word = word.strip(',.-;()[]').lower()
        if word in sentiment_words:
            filtered_word_list.append(word)
    filtered_doc = ' '.join(filtered_word_list)
    filtered_train_data.append(filtered_doc)
```

In [29]:
```
filtered_train_tfidf = tvec.fit_transform(filtered_train_data)
```

In [ ]:
```
# COMPARISON TO UNFILTERED TRAINING DATA:
#
# MultinomialNB: filtered is higher
# AdaBoostClassifier: filtered is lower
# MLPClassifier: filtered is lower (recall is about the same)
# LogisticRegression: filtered is lower
# SVC: about the same
#
# Unfiltered training data generally resulted in better accuracy in cros
s validation.
# This probably indicates that better context can be drawn from the full
reviews than from the reviews
# filtered by the sentiment words.
```

In [54]:

```
for clf in [MultinomialNB(),
            AdaBoostClassifier(n_estimators=50),
            MLPClassifier(hidden_layer_sizes=(5, 3)),
            LogisticRegression(solver='liblinear', multi_class='ovr', penalty='l2', C=1),
            SVC(gamma='auto', C=1000, kernel='rbf')
           ]:
    results = cross_validate(clf, filtered_train_tfidf, train_target, cv=5, return_train_score=False, scoring=('precision_macro', 'recall_macro', 'f1_macro'))
    print('\n\nRESULTS FOR {}'.format(clf))
    for key in ['test_precision_macro', 'test_recall_macro', 'test_f1_macro']:
        print('{0: <22}: {1}'.format(key.upper(), results[key].mean()))
```

```
RESULTS FOR MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
TEST_PRECISION_MACRO  : 0.4537654758377938
TEST_RECALL_MACRO     : 0.45170000000000005
TEST_F1_MACRO         : 0.4496737745893431


RESULTS FOR AdaBoostClassifier(algorithm='SAMME.R', base_estimator=Non
e,
          learning_rate=1.0, n_estimators=50, random_state=None)
TEST_PRECISION_MACRO  : 0.40878374635803566
TEST_RECALL_MACRO     : 0.4199
TEST_F1_MACRO         : 0.4066466723679296
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
RESULTS FOR MLPClassifier(activation='relu', alpha=0.0001, batch_size
='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(5, 3), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
TEST_PRECISION_MACRO  : 0.36235554347162413
TEST_RECALL_MACRO     : 0.37120000000000003
TEST_F1_MACRO         : 0.358345443443277


RESULTS FOR LogisticRegression(C=1, class_weight=None, dual=False, fit_
intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          n_jobs=None, penalty='l2', random_state=None, solver='libline
ar',
          tol=0.0001, verbose=0, warm_start=False)
TEST_PRECISION_MACRO  : 0.4358384669421536
TEST_RECALL_MACRO     : 0.4471
TEST_F1_MACRO         : 0.43558902768041896


RESULTS FOR SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
TEST_PRECISION_MACRO  : 0.44470675057931947
TEST_RECALL_MACRO     : 0.4479
TEST_F1_MACRO         : 0.4395850713449826
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
## (iii) evaluation on test dataset
#
# In every case, reviews with ratings 2-4 were generally hard to predic
t.  Specifically, reviews with
# a rating of 3 were the hardest to predict.  Reviews with ratings of 1
 and 5 were predicted most accuractely.
```

In [55]:

```
from sklearn.metrics import classification_report
test_tfidf = tvec.transform(test_data)
```

In [56]:

```
for clf in [MultinomialNB(),
            AdaBoostClassifier(n_estimators=50),
            MLPClassifier(hidden_layer_sizes=(5, 3)),
            LogisticRegression(solver='liblinear', multi_class='ovr', pe
nalty='l2', C=1),
            SVC(gamma='auto', C=1000, kernel='rbf')
           ]:
    clf_fit = clf.fit(train_tfidf, train_target)
    predicted = clf_fit.predict(test_tfidf)
    print('\n\nRESULTS FOR {}'.format(clf))
    print(classification_report(test_target, predicted))
```

```
RESULTS FOR MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
              precision    recall  f1-score   support

         1.0       0.51      0.67      0.57       200
         2.0       0.51      0.39      0.44       200
         3.0       0.42      0.44      0.43       200
         4.0       0.45      0.48      0.47       200
         5.0       0.72      0.57      0.64       200

   micro avg       0.51      0.51      0.51      1000
   macro avg       0.52      0.51      0.51      1000
weighted avg       0.52      0.51      0.51      1000



RESULTS FOR AdaBoostClassifier(algorithm='SAMME.R', base_estimator=Non
e,
          learning_rate=1.0, n_estimators=50, random_state=None)
              precision    recall  f1-score   support

         1.0       0.58      0.59      0.59       200
         2.0       0.40      0.34      0.37       200
         3.0       0.39      0.37      0.38       200
         4.0       0.40      0.37      0.38       200
         5.0       0.53      0.67      0.59       200

   micro avg       0.47      0.47      0.47      1000
   macro avg       0.46      0.47      0.46      1000
weighted avg       0.46      0.47      0.46      1000

/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
RESULTS FOR MLPClassifier(activation='relu', alpha=0.0001, batch_size
='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(5, 3), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
              precision    recall  f1-score   support

         1.0       0.48      0.55      0.51       200
         2.0       0.39      0.45      0.42       200
         3.0       0.40      0.34      0.37       200
         4.0       0.41      0.45      0.43       200
         5.0       0.58      0.45      0.51       200

   micro avg       0.45      0.45      0.45      1000
   macro avg       0.45      0.45      0.45      1000
weighted avg       0.45      0.45      0.45      1000



RESULTS FOR LogisticRegression(C=1, class_weight=None, dual=False, fit_
intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          n_jobs=None, penalty='l2', random_state=None, solver='libline
ar',
          tol=0.0001, verbose=0, warm_start=False)
              precision    recall  f1-score   support

         1.0       0.54      0.80      0.64       200
         2.0       0.52      0.34      0.41       200
         3.0       0.45      0.39      0.42       200
         4.0       0.47      0.47      0.47       200
         5.0       0.66      0.68      0.67       200

   micro avg       0.53      0.53      0.53      1000
   macro avg       0.53      0.53      0.52      1000
weighted avg       0.53      0.53      0.52      1000



RESULTS FOR SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
              precision    recall  f1-score   support

         1.0       0.55      0.69      0.61       200
         2.0       0.49      0.46      0.47       200
         3.0       0.44      0.42      0.43       200
         4.0       0.48      0.39      0.43       200
         5.0       0.63      0.68      0.65       200

   micro avg       0.53      0.53      0.53      1000
   macro avg       0.52      0.53      0.52      1000
```

```
weighted avg       0.52      0.53      0.52      1000
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```python
# FYI -- Predictions made on FILTERED test data with classifiers trained on the filtered training data
```

In [57]:

```python
filtered_test_data = []
for doc in test_data:
    filtered_word_list = []
    word_list = doc.split()
    for word in word_list:
        word = word.strip(',.-;()[]').lower()
        if word in sentiment_words:
            filtered_word_list.append(word)
    filtered_doc = ' '.join(filtered_word_list)
    filtered_test_data.append(filtered_doc)
```

In [58]:

```python
filtered_train_tfidf = tvec.transform(filtered_train_data)
filtered_test_tfidf = tvec.transform(filtered_test_data)
```

In [59]:

```
for clf in [MultinomialNB(),
            AdaBoostClassifier(n_estimators=50),
            MLPClassifier(hidden_layer_sizes=(5, 3)),
            LogisticRegression(solver='liblinear', multi_class='ovr', pe
nalty='l2', C=1),
            SVC(gamma='auto', C=1000, kernel='rbf')
           ]:
    clf_fit = clf.fit(filtered_train_tfidf, train_target)
    predicted = clf_fit.predict(filtered_test_tfidf)
    print('\n\nRESULTS FOR {}'.format(clf))
    print(classification_report(test_target, predicted))
```

```
RESULTS FOR MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
              precision    recall  f1-score   support

         1.0       0.63      0.54      0.58       200
         2.0       0.43      0.38      0.41       200
         3.0       0.34      0.42      0.38       200
         4.0       0.36      0.40      0.38       200
         5.0       0.54      0.51      0.52       200

   micro avg       0.45      0.45      0.45      1000
   macro avg       0.46      0.45      0.45      1000
weighted avg       0.46      0.45      0.45      1000



RESULTS FOR AdaBoostClassifier(algorithm='SAMME.R', base_estimator=Non
e,
          learning_rate=1.0, n_estimators=50, random_state=None)
              precision    recall  f1-score   support

         1.0       0.50      0.60      0.55       200
         2.0       0.44      0.29      0.35       200
         3.0       0.31      0.30      0.31       200
         4.0       0.35      0.28      0.31       200
         5.0       0.46      0.62      0.53       200

   micro avg       0.42      0.42      0.42      1000
   macro avg       0.41      0.42      0.41      1000
weighted avg       0.41      0.42      0.41      1000

/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilaye
r_perceptron.py:562: ConvergenceWarning: Stochastic Optimizer: Maximum
iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
RESULTS FOR MLPClassifier(activation='relu', alpha=0.0001, batch_size
='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(5, 3), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
              precision    recall  f1-score   support

         1.0       0.50      0.54      0.52       200
         2.0       0.43      0.40      0.42       200
         3.0       0.38      0.28      0.32       200
         4.0       0.36      0.49      0.42       200
         5.0       0.47      0.41      0.44       200

   micro avg       0.42      0.42      0.42      1000
   macro avg       0.43      0.42      0.42      1000
weighted avg       0.43      0.42      0.42      1000



RESULTS FOR LogisticRegression(C=1, class_weight=None, dual=False, fit_
intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          n_jobs=None, penalty='l2', random_state=None, solver='libline
ar',
          tol=0.0001, verbose=0, warm_start=False)
              precision    recall  f1-score   support

         1.0       0.58      0.62      0.60       200
         2.0       0.43      0.34      0.38       200
         3.0       0.37      0.41      0.39       200
         4.0       0.38      0.36      0.37       200
         5.0       0.53      0.56      0.54       200

   micro avg       0.46      0.46      0.46      1000
   macro avg       0.46      0.46      0.46      1000
weighted avg       0.46      0.46      0.46      1000



RESULTS FOR SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
              precision    recall  f1-score   support

         1.0       0.55      0.60      0.57       200
         2.0       0.43      0.44      0.43       200
         3.0       0.38      0.41      0.39       200
         4.0       0.40      0.31      0.35       200
         5.0       0.58      0.59      0.59       200

   micro avg       0.47      0.47      0.47      1000
   macro avg       0.47      0.47      0.47      1000
```

```
weighted avg       0.47      0.47      0.47      1000
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
## (iv) some ideas to help improve predictions...
# - increase number of training examples.
# - fewer ratings (ratings of 1-3 instead of 1-5)
# - clean the testing and training data better (strip punctuation, ascii
characters, etc) although
#   I think the tfidf vectorizer is supposed to do that already
```