sentiment_analysis_final_2

February 27, 2018

```
In [2]: import sklearn
        import numpy as np
        import scipy.sparse
        from sklearn.model_selection import GridSearchCV
        from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import chi2
        from sklearn import svm
        import time
        # read in dataset
        train_path = "train.txt"
        test_path = "test.txt"
```

```
In [3]: def load_data(path):
            with open(path, "r") as f:
                lines = f.readlines()

            classes = []
            #samples = []
            docs = []

            for line in lines:
                classes.append(int(line.rsplit()[-1]))
                #samples.append(line.rsplit()[0])
                #raw = line.decode('latin1')
                raw = ' '.join(line.rsplit()[1:-1])
                docs.append(raw)

            return (docs, classes)
```

```
In [4]: X_train, Y_train = load_data(train_path)
        X_test, Y_test = load_data(test_path)

        def search(sequence):
            result = []
            for word in sequence:
                counter = [1 if word in x else 0 for x in X_train]
                indexes = []
```

```
        for i in range(len(counter)):
            if counter[i] != 0:
                indexes.append(i)
        positive = 0
        negative = 0
        for i in range(len(indexes)):
            if Y_train[indexes[i]] == 1:
                positive += 1
            else:
                negative += 1
        result.append((word,positive,negative))
    return result

chars = ['{','}','#','%','&','\(','\)','\[','\]','<','>',',', '!', '.', ';',
'?', '*', '\\', '\/', '~', '_','|','=','+','^',':','\"','\'','@','-']
for element in search(["!", "'", "?"]):
    print(element)
```

```
('!', 192, 102)
("'", 197, 279)
('?', 2, 14)
```

In [5]:
```
from sklearn.feature_extraction.text import CountVectorizer
import nltk, re
from nltk.corpus import stopwords

porter = nltk.PorterStemmer() # also lancaster stemmer
wnl = nltk.WordNetLemmatizer()
stopWords = stopwords.words("english")
chars = ['{','}','#','%','&','\(','\)','\[','\]','<','>',',', '!', '.', ';',
'?', '*', '\\', '\/', '~', '_','|','=','+','^',':','\"','\'','@','-']

def preprocess(raw):
    line = re.sub('[%s]' % ''.join(chars), ' ', (raw))
    words = line.split(' ')
    words = [w.lower() for w in words]
    words = [w for w in words if w not in stopWords]
    words = [wnl.lemmatize(w) for w in words]
    processed = ' '.join([porter.stem(w) for w in words])

    return processed
```

In [12]:
```
#Finding most common words in pos/neg classes
import heapq

positives = {}
```

```
    negatives = {}

    for i, sentence in enumerate(X_train):
        words = sentence.split(" ")
        #add bigrams
        #for i in range(len(words)-1):
        #    words.append(words[i]+" "+words[i+1])
        if Y_train[i] == 1:
            for word in words:
                if word in stopWords:
                    continue
                if word in positives:
                    positives[word] += 1
                else:
                    positives[word] = 1
        else:
            for word in words:
                if word in stopWords:
                    continue
                if word in negatives:
                    negatives[word] += 1
                else:
                    negatives[word] = 1

    def maximumN(mydict, N):
        myheap = []
        count = 0
        for key in mydict:
            if count == N:
                heapq.heappush(myheap, (mydict[key], key))
                heapq.heappop(myheap)

            else:
                heapq.heappush(myheap, (mydict[key], key))
                count += 1

        print(myheap)

    stopWords.extend(["I","movie","It","The","This"])
    maximumN(positives, 15)
    maximumN(negatives, 15)
```

```
[(29, '-'), (30, 'food'), (29, 'well'), (30, 'place'), (41, 'Great'), (34, 'best'), (31, 'it.')
[(29, 'film'), (29, 'it.'), (29, 'get'), (29, 'really'), (31, 'place'), (31, 'good'), (32, 'eve
```

```
In [7]: def new_features(char, word, cap):
            toReturn = []
```

```python
    if char or word or cap:
        print("adding new features")
    if char: #compute character length feature
        char_train = [len(x) for x in X_train]
        char_test = [len(x) for x in X_test]
        toReturn.append((char_train, char_test))
    else:
        toReturn.append(None)

    if word: #compute word length feature
        word_train = [len(x.split(" ")) for x in X_train]
        word_test = [len(x.split(" ")) for x in X_test]
        toReturn.append((word_train, word_test))
    else:
        toReturn.append(None)

    if cap: #compute cap count feature
        cap_train = [sum([1 if x.isalpha() and x.isupper() else 0 for x in row]) for ro
        cap_test = [sum([1 if x.isalpha() and x.isupper() else 0 for x in row]) for row
        toReturn.append((cap_train, cap_test))
    else:
        toReturn.append(None)

    return toReturn

#print("Before")
#print("mean: ", np.mean(cap_train))
#print("sd: ", np.std(cap_train))
#print("min: ", min(cap_train))
#print("max: ", max(cap_train))

def warp(features, log, std, reg): #warp to sd of 1
    toReturn = []

    for i in range(0,len(features)):
        if features[i] is not None:
            if std:
                sd = np.std(features[i][0])
                train = [x / sd for x in features[i][0]]
                test = [x / sd for x in features[i][1]]
            elif log:
                train = [np.log(x) for x in features[i][0]]
                test = [np.log(x) for x in features[i][1]]
            else:
                train = features[i][0]
                test = features[i][1]
            toReturn.append((train,test))
        else:
```

```
            toReturn.append(None)

    return toReturn

def add_features(features, train, test):
    for i in range(0, len(features)):
        if features[i] is not None:
            train = scipy.sparse.hstack((train,np.array(features[i][0])[:,None]))
            test = scipy.sparse.hstack((test,np.array(features[i][1])[:,None]))
    return (train,test)

#print("After")
#print("mean: ", np.mean(X_cap_train))
#print("sd: ", np.std(X_cap_train))
#print("min: ", min(X_cap_train))
#print("max: ", max(X_cap_train))
```

```
In [8]: import sys
        #print("length of train: ", len(X_train))
        #print("length of test: ", len(X_test))

        features = new_features(True, True, True)
        title = ["Char", "Word", "Cap"]


        for i in range(len(features)):
            train = features[i][0]
            test = features[i][1]
            both = list(train)
            both.extend(test)
            print("Statistics for " + title[i] + " Count")
            print("mean: ", np.mean(both), np.mean(train), np.mean(test))
            print("sd:   ", np.std(both), np.std(train), np.std(test))
            print("min:  ", min(both), min(train), min(test))
            print("max:  ", max(both), max(train), max(test))
```

```
adding new features
Statistics for Char Count
mean:  64.60433333333333 64.44541666666667 65.24
sd:    43.90844012892687 44.13218236910251 42.99595794955614
min:   5 5 9
max:   477 477 283
Statistics for Word Count
mean:  11.831666666666667 11.7875 12.008333333333333
sd:    7.871128501612008 7.883411513002054 7.819309254801362
min:   1 1 1
max:   71 71 53
Statistics for Cap Count
```

```
mean:  2.0616666666666665 2.0820833333333333 1.98
sd:    3.1380244138983717 3.2777551758872345 2.50058659784726
min:   0 0 0
max:   78 78 31
```

```
In [9]: X_train_out = []
        X_test_out = []

        def feature_extension(char, word, cap):
            global X_train_out, X_test_out
            #print(X_train_out.getnnz(1)) #nonzeros across row

            #add features
            features = new_features(char, word, cap) #char, word, cap
            warped = warp(features, False, True, False) #log, std, reg
            result = add_features(warped, X_train_out, X_test_out)
            X_train_out = result[0]
            X_test_out = result[1]

            #print(X_train_out.getnnz(1)) #nonzeros across row
```

```
In [8]: from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.decomposition import TruncatedSVD
        from sklearn import metrics

        def count(multi, maxgram, tfidf, svd, chi, char, word, cap):
            global X_train_out, X_test_out
            if tfidf:
                print("extracting tfidf...")
                counter = TfidfVectorizer(use_idf=True, preprocessor=preprocess, ngram_range=(
                X_train_out = counter.fit_transform(X_train)
                X_test_out = counter.transform(X_test)
            else:
                if multi:
                    print("multinomial distribution")
                else:
                    print("bernoulli distribution")
                print("maxgram: " + str(maxgram))
                counter = CountVectorizer(preprocessor=preprocess, binary=multi, ngram_range=(
                X_train_out = counter.fit_transform(X_train)
                X_test_out = counter.transform(X_test)

            if svd:
                print("SVD dimenstionality reduction")
                svd = TruncatedSVD(n_components=100)
                X_train_out = svd.fit_transform(X_train_out)
                X_test_out = svd.transform(X_test_out)
```

```
        elif chi:
            print("chi2 feature reduction")
            kbest = SelectKBest(chi2, k=100)
            X_train_out = kbest.fit_transform(X_train_out, Y_train)
            X_test_out = kbest.transform(X_test_out)
        feature_extension(char, word, cap)
```

```
In [9]: def gridsearchSVM(train_feature, train_label, test_feature, test_label):
            params = {"kernel":[ "linear", "poly", "rbf", "sigmoid"],
                   "C":[0.001, 0.01, 0.1, 1, 10, 100, 1000]}
            scoring = metrics.make_scorer(metrics.accuracy_score)
            grid = GridSearchCV(svm.SVC(random_state=0), params, cv=5,
                                scoring=scoring)
            grid.fit(train_feature, train_label)

            print("best parameters: ")
            print(grid.best_estimator_)
            preds = grid.predict(test_feature)

            print("Accuracy: " + str(metrics.accuracy_score(preds, test_label)))
            print("F1: " + str(metrics.f1_score(preds, test_label)))
```

```
In [10]: from sklearn.naive_bayes import BernoulliNB
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.tree import DecisionTreeClassifier

         def execute(bern, mult, rf, knn, dt, svm, multi, maxgram, tfidf, svd, chi, char, word
             global X_train_out, X_test_out

             start = time.time()
             if bern:
                 clf = BernoulliNB()
                 print("<Bernoulli NB>")
             elif mult:
                 print("<Multinomial NB>")
                 clf = MultinomialNB()
             elif rf:
                 print("<Random Forest>")
                 clf = RandomForestClassifier(n_estimators=100, random_state=0)
             elif knn:
                 print("<KNN>")
                 clf = KNeighborsClassifier(n_neighbors=50)
             elif dt:
                 print("<Decision Tree>")
                 clf = DecisionTreeClassifier(random_state=0)
```

```
            else:
                x = 1

            count(multi, maxgram, tfidf, svd, chi, char, word, cap)

            if svm:
                print("<SVM>")
                gridsearchSVM(X_train_out, Y_train, X_test_out, Y_test)
            else:
                clf.fit(X_train_out, Y_train)
                Y_pred = clf.predict(X_test_out)
                print("Accuracy: " + str(metrics.accuracy_score(Y_pred, Y_test)))
                print("F1: " + str(metrics.f1_score(Y_pred, Y_test)))

            end = time.time()
            print("run time: " + str(end - start))
```

```
In [11]: execute(bern=True, mult=False, rf=False, knn=False, dt=False, svm=False,
                 multi=False, maxgram=1, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<Bernoulli NB>
bernoulli distribution
maxgram: 1
Accuracy: 0.808333333333
F1: 0.801381692573
run time: 3.7529489994
```

```
In [12]: execute(bern=False, mult=True, rf=False, knn=False, dt=False, svm=False,
                 multi=True, maxgram=1, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<Multinomial NB>
multinomial distribution
maxgram: 1
Accuracy: 0.811666666667
F1: 0.807495741056
run time: 0.974588871002
```

```
In [13]: execute(bern=False, mult=False, rf=False, knn=True, dt=False, svm=False,
                 multi=True, maxgram=1, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<KNN>
multinomial distribution
maxgram: 1
```

```
Accuracy: 0.638333333333
F1: 0.479616306954
run time: 1.01091599464
```

```
In [14]: execute(bern=False, mult=False, rf=False, knn=False, dt=True, svm=False,
                 multi=True, maxgram=1, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<Decision Tree>
multinomial distribution
maxgram: 1
Accuracy: 0.78
F1: 0.774744027304
run time: 1.1004178524
```

```
In [15]: execute(bern=False, mult=False, rf=True, knn=False, dt=False, svm=False,
                 multi=True, maxgram=1, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<Random Forest>
multinomial distribution
maxgram: 1
Accuracy: 0.8
F1: 0.78102189781
run time: 2.43800497055
```

```
In [16]: execute(bern=False, mult=False, rf=False, knn=False, dt=False, svm=True,
                 multi=True, maxgram=1, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
multinomial distribution
maxgram: 1
<SVM>
best parameters:
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='sigmoid',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
Accuracy: 0.818333333333
F1: 0.805008944544
run time: 67.9869029522
```

```
In [17]: execute(bern=True, mult=False, rf=False, knn=False, dt=False, svm=False,
                 multi=False, maxgram=2, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<Bernoulli NB>
bernoulli distribution
maxgram: 2
Accuracy: 0.816666666667
F1: 0.804270462633
run time: 1.12560200691
```

```
In [18]: execute(bern=False, mult=True, rf=False, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<Multinomial NB>
multinomial distribution
maxgram: 2
Accuracy: 0.818333333333
F1: 0.814310051107
run time: 1.24065494537
```

```
In [19]: execute(bern=False, mult=False, rf=False, knn=False, dt=True, svm=False,
                 multi=True, maxgram=2, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<Decision Tree>
multinomial distribution
maxgram: 2
Accuracy: 0.775
F1: 0.761061946903
run time: 1.64271783829
```

```
In [20]: execute(bern=False, mult=False, rf=True, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<Random Forest>
multinomial distribution
maxgram: 2
Accuracy: 0.791666666667
F1: 0.760076775432
run time: 3.74967217445
```

```
In [21]: execute(bern=False, mult=False, rf=False, knn=False, dt=False, svm=True,
                 multi=True, maxgram=2, tfidf=False, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
multinomial distribution
maxgram: 2
```

```
<SVM>
best parameters:
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
Accuracy: 0.816666666667
F1: 0.798534798535
run time: 89.0351891518
```

```
In [22]: #Good
         execute(bern=False, mult=True, rf=False, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<Multinomial NB>
extracting tfidf...
Accuracy: 0.821666666667
F1: 0.818336162988
run time: 1.24855899811
```

```
In [23]: execute(bern=False, mult=False, rf=True, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
<Random Forest>
extracting tfidf...
Accuracy: 0.808333333333
F1: 0.795008912656
run time: 3.53962922096
```

```
In [24]: #good
         execute(bern=False, mult=False, rf=False, knn=False, dt=False, svm=True,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=False,
                 char=False, word=False, cap=False)
```

```
extracting tfidf...
<SVM>
best parameters:
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
Accuracy: 0.838333333333
F1: 0.834188034188
run time: 91.3393120766
```

```
In [25]: execute(bern=False, mult=True, rf=False, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=False,
                 char=True, word=True, cap=True)
```

```
<Multinomial NB>
extracting tfidf...
adding new features
Accuracy: 0.813333333333
F1: 0.804195804196
run time: 1.26393198967
```

```
In [26]: execute(bern=False, mult=True, rf=False, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=False, svd=False, chi=False,
                 char=True, word=True, cap=True)
```

```
<Multinomial NB>
multinomial distribution
maxgram: 2
adding new features
Accuracy: 0.813333333333
F1: 0.806228373702
run time: 1.14739179611
```

```
In [27]: execute(bern=False, mult=False, rf=True, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=False, svd=False, chi=False,
                 char=True, word=True, cap=True)
```

```
<Random Forest>
multinomial distribution
maxgram: 2
adding new features
Accuracy: 0.783333333333
F1: 0.750957854406
run time: 3.56088113785
```

```
In [28]: execute(bern=False, mult=False, rf=True, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=False,
                 char=True, word=True, cap=True)
```

```
<Random Forest>
extracting tfidf...
adding new features
Accuracy: 0.795
F1: 0.779964221825
run time: 3.24257588387
```

```
In [29]: execute(bern=False, mult=False, rf=False, knn=False, dt=False, svm=True,
                 multi=True, maxgram=2, tfidf=False, svd=False, chi=False,
                 char=True, word=True, cap=True)
```

```
multinomial distribution
maxgram: 2
adding new features
<SVM>
best parameters:
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
Accuracy: 0.82
F1: 0.801470588235
run time: 108.476480007
```

```
In [30]: #good
         execute(bern=False, mult=False, rf=False, knn=False, dt=False, svm=True,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=False,
                 char=True, word=True, cap=True)
```

```
extracting tfidf...
adding new features
<SVM>
best parameters:
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
Accuracy: 0.841666666667
F1: 0.834782608696
run time: 118.193981886
```

```
In [31]: execute(bern=False, mult=False, rf=True, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=True, svd=True, chi=False,
                 char=False, word=False, cap=False)
```

```
<Random Forest>
extracting tfidf...
SVD dimenstionality reduction
Accuracy: 0.741666666667
F1: 0.73321858864
run time: 3.79168009758
```

```
In [32]: execute(bern=False, mult=False, rf=False, knn=False, dt=False, svm=True,
                 multi=True, maxgram=2, tfidf=True, svd=True, chi=False,
                 char=False, word=False, cap=False)
```

```
extracting tfidf...
SVD dimenstionality reduction
<SVM>
best parameters:
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
Accuracy: 0.748333333333
F1: 0.735551663748
run time: 149.725368977
```

```
In [33]: execute(bern=False, mult=False, rf=True, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=True,
                 char=False, word=False, cap=False)
```

```
<Random Forest>
extracting tfidf...
chi2 feature reduction
Accuracy: 0.756666666667
F1: 0.708
run time: 1.47477698326
```

```
In [34]: execute(bern=False, mult=False, rf=True, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=True,
                 char=True, word=True, cap=True)
```

```
<Random Forest>
extracting tfidf...
chi2 feature reduction
adding new features
Accuracy: 0.748333333333
F1: 0.746218487395
run time: 2.00016522408
```

```
In [35]: execute(bern=False, mult=False, rf=False, knn=False, dt=False, svm=True,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=True,
                 char=False, word=False, cap=False)
```

```
extracting tfidf...
chi2 feature reduction
<SVM>
best parameters:
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
```

```
  tol=0.001, verbose=False)
Accuracy: 0.768333333333
F1: 0.724752475248
run time: 19.3123428822
```

```
In [36]: execute(bern=False, mult=False, rf=False, knn=False, dt=False, svm=True,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=True,
                 char=True, word=True, cap=True)
```

```
extracting tfidf...
chi2 feature reduction
adding new features
<SVM>
best parameters:
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
Accuracy: 0.768333333333
F1: 0.724752475248
run time: 624.191854954
```

```
In [38]: execute(bern=False, mult=True, rf=False, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=True,
                 char=False, word=False, cap=False)
```

```
<Multinomial NB>
extracting tfidf...
chi2 feature reduction
Accuracy: 0.77
F1: 0.722891566265
run time: 0.994335889816
```

```
In [37]: execute(bern=False, mult=True, rf=False, knn=False, dt=False, svm=False,
                 multi=True, maxgram=2, tfidf=True, svd=False, chi=True,
                 char=True, word=True, cap=True)
```

```
<Multinomial NB>
extracting tfidf...
chi2 feature reduction
adding new features
Accuracy: 0.771666666667
F1: 0.728712871287
run time: 1.13313698769
```

In [39]:
```
X = X_train + X_test
Y = Y_train + Y_test
```

In [40]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import KFold, train_test_split

def validate_model(clf, features, label, char_feat):
    accuracy_result = []
    f1_result = []
    kf = KFold(n_splits=5, shuffle=True, random_state=0)
    i = 0
    for train_ind, test_ind in kf.split(features):
        train_feat = features[train_ind]
        test_feat = features[test_ind]

        counter = TfidfVectorizer(use_idf=True, preprocessor=preprocess, ngram_range=
        train_feat_out = counter.fit_transform(train_feat)
        test_feat_out = counter.transform(test_feat)

        if char_feat:
            char_train = [len(x) for x in train_feat]
            char_test = [len(x) for x in test_feat]
            char_sd = np.std(char_train)
            char_train = [x / char_sd for x in char_train]
            char_test = [x / char_sd for x in char_test]
            train_feat_out = scipy.sparse.hstack((train_feat_out,np.array(char_train)
            test_feat_out = scipy.sparse.hstack((test_feat_out,np.array(char_test)[:,N

            word_train = [len(x.split(" ")) for x in train_feat]
            word_test = [len(x.split(" ")) for x in test_feat]
            word_sd = np.std(word_train)
            word_train = [x / word_sd for x in word_train]
            word_test = [x / word_sd for x in word_test]
            train_feat_out = scipy.sparse.hstack((train_feat_out,np.array(word_train)
            test_feat_out = scipy.sparse.hstack((test_feat_out,np.array(word_test)[:,N

            cap_train = [sum([1 if x.isalpha() and x.isupper() else 0 for x in row]) f
            cap_test = [sum([1 if x.isalpha() and x.isupper() else 0 for x in row]) fo
            cap_sd = np.std(cap_train)
            cap_train = [x / cap_sd for x in cap_train]
            cap_test = [x / cap_sd for x in cap_test]
            train_feat_out = scipy.sparse.hstack((train_feat_out,np.array(cap_train)[
            test_feat_out = scipy.sparse.hstack((test_feat_out,np.array(cap_test)[:,No



        i = i + 1
        print ('Fold {}'.format(i))
```

```
                clf.fit(train_feat_out, label[train_ind])
                Y_pred = clf.predict(test_feat_out)
                accuracy = metrics.accuracy_score(Y_pred, label[test_ind])
                f1score = metrics.f1_score(Y_pred, label[test_ind])
                accuracy_result.append(accuracy)
                f1_result.append(f1score)

                print ('Accuracy:{}'.format(accuracy))
                print ('F1 Score: {}'.format(f1score))

            print ('Overall Accuracy: {}'.format(np.mean(accuracy_result)))
            print ('Overall F1 Score: {}'.format(np.mean(f1_result)))
```

```
In [42]: X = X_train + X_test
         Y = Y_train + Y_test
         validate_model(MultinomialNB(), np.asarray(X), np.asarray(Y),False)
```

```
Fold 1
Accuracy:0.82
F1 Score: 0.823529411765
Fold 2
Accuracy:0.82
F1 Score: 0.828025477707
Fold 3
Accuracy:0.803333333333
F1 Score: 0.797250859107
Fold 4
Accuracy:0.81
F1 Score: 0.820189274448
Fold 5
Accuracy:0.84
F1 Score: 0.843648208469
Overall Accuracy: 0.818666666667
Overall F1 Score: 0.822528646299
```

```
In [43]: validate_model(svm.SVC(random_state=0, C=1, kernel="linear"), np.asarray(X), np.asarra
```

```
Fold 1
Accuracy:0.806666666667
F1 Score: 0.80204778157
Fold 2
Accuracy:0.835
F1 Score: 0.8416
Fold 3
Accuracy:0.815
F1 Score: 0.806956521739
Fold 4
Accuracy:0.803333333333
```

```
F1 Score: 0.810897435897
Fold 5
Accuracy:0.828333333333
F1 Score: 0.831423895254
Overall Accuracy: 0.817666666667
Overall F1 Score: 0.818585126892
```

```
In [44]: validate_model(svm.SVC(random_state=0, C=1, kernel="linear"), np.asarray(X), np.asarra
```

```
Fold 1
Accuracy:0.8
F1 Score: 0.797297297297
Fold 2
Accuracy:0.833333333333
F1 Score: 0.839743589744
Fold 3
Accuracy:0.811666666667
F1 Score: 0.804835924007
Fold 4
Accuracy:0.803333333333
F1 Score: 0.810897435897
Fold 5
Accuracy:0.833333333333
F1 Score: 0.837133550489
Overall Accuracy: 0.816333333333
Overall F1 Score: 0.817981559487
```

```
In [62]: def most_informative(clf, train_feat, train_label, char_feat):
             accuracy_result = []
             f1_result = []

             counter = TfidfVectorizer(use_idf=True, preprocessor=preprocess, ngram_range=(1, 
             train_feat_out = counter.fit_transform(train_feat)
             feat_name = counter.get_feature_names()

             if char_feat:
                 char_train = [len(x) for x in train_feat]
                 char_sd = np.std(char_train)
                 char_train = [x / char_sd for x in char_train]
                 train_feat_out = scipy.sparse.hstack((train_feat_out,np.array(char_train)[:,No

                 word_train = [len(x.split(" ")) for x in train_feat]
                 word_sd = np.std(word_train)
                 word_train = [x / word_sd for x in word_train]
                 train_feat_out = scipy.sparse.hstack((train_feat_out,np.array(word_train)[:,No
```

```
                cap_train = [sum([1 if x.isalpha() and x.isupper() else 0 for x in row]) for i
                cap_sd = np.std(cap_train)
                cap_train = [x / cap_sd for x in cap_train]
                train_feat_out = scipy.sparse.hstack((train_feat_out,np.array(cap_train)[:,Non

                feat_name = feat_name + ["Char Count", "Word Count", "Uppercase Count"]


            clf.fit(train_feat_out, train_label)
            print("Positive Class :")
            informative = np.argsort(clf.coef_[0])[-10:]
            for i in informative:
                print(feat_name[i])

            print("\nNegative Class :")
            informative = np.argsort(clf.coef_[0])[0:10]
            for i in informative:
                print(feat_name[i])
```

In [65]:
```
most_informative(MultinomialNB(), X_train, Y_train, char_feat=False)
```

```
Positive Class :
well
excel
food
movi
film
work
phone
love
good
great

Negative Class :
00
miss numer
miss entir
misplac
mislead
mishima extrem
miser hollow
miser
mirrormask last
mirrormask
```

In [45]:
```
# add more data!
# read in dataset
```

```
def extend():
    global X_train, Y_train
    with open("Sentiment Analysis Dataset.csv", "r") as f:
        lines = f.readlines()

    newX = []
    newY = []

    for line in lines[1:]:
        newY.append(int(line.split(',')[1]))
        raw = ' '.join(line.rsplit()[1:])
        newX.append(raw)

    X_train = X_train + newX
    Y_train = Y_train + newY
```