Sentiment Identification

Practical 9: A Sentimental Journey

Human Ratings
Q1

Do a Rating Task

- Get 3 classmates (opinion holders) to write three different opinions about their phone
- Get 3 diff people (raters) to rate these comments as positive, negative, neutral or can't-say
- Take this 3 x 3 matrix and find the inter-rater reliability between your 3 raters using Kappa
- If you wanted to get the correlation between raters (using Pearson's *rho*) what would u do?

Spreadsheets can be used,
Pearsons is a known fn, but Cohen's Kappa is not.

Sentiment Lists
Q2

Finding…

- Do, some searches and find 3 sentiment lists that are commonly used in previous research
- For 2 of these lists, select 10 positive and 10 negative words (randomly)
- Evaluate each word, discussing whether it is really positive/negative; for each word try to find sentential context in which it might be interpreted with the opposite valence

A Program
Q3

Practical Q3

- So, take a look at this sentiment program and see what is happening in the different variables
- Now consider ways to improve the training.
- Eg if you removed stop-words from the inputs what do you think might happen
- Implement this or another solution

```python
# Andy Bromberg's Simple Sentiment Analysis System
import re, math, collections, itertools
import nltk, nltk.classify.util, nltk.metrics
from nltk.classify import NaiveBayesClassifier
from nltk.metrics import BigramAssocMeasures
from nltk.probability import FreqDist, ConditionalFreqDist

def evaluate_features(feature_select):
    negSentences = open('/Users/user/Desktop/rt-polarity-neg.txt', 'r')
    posSentences = open('/Users/user/Desktop/rt-polarity-pos.txt', 'r')
    negSentences = re.split(r'\n', negSentences.read())
    posSentences = re.split(r'\n', posSentences.read())

    posFeatures = []
    negFeatures = []
    for i in posSentences:
        posWords = re.findall(r"[\w']+|[.,!?;]", i)
        posWords = [feature_select(posWords), 'pos']
        posFeatures.append(posWords)
    for i in negSentences:
        negWords = re.findall(r"[\w']+|[.,!?;]", i)
        negWords = [feature_select(negWords), 'neg']
        negFeatures.append(negWords)

    posCutoff = int(math.floor(len(posFeatures)*3/4))
    negCutoff = int(math.floor(len(negFeatures)*3/4))
    trainFeatures = posFeatures[:posCutoff] + negFeatures[:negCutoff]
    testFeatures = posFeatures[posCutoff:] + negFeatures[negCutoff:]
    classifier = NaiveBayesClassifier.train(trainFeatures)

    referenceSets = collections.defaultdict(set)
    testSets = collections.defaultdict(set)
    for i, (features, label) in enumerate(testFeatures):
        referenceSets[label].add(i)
        predicted = classifier.classify(features)
        testSets[predicted].add(i)

def make_full_dict(words):
    return dict([(word, True) for word in words])

evaluate_features(make_full_dict)
```

import a bunch of stuff

read in +/- sentences

turn sents. into words and structure feature inputs for classifier

```python
# Andy Bromberg's Simple Sentiment Analysis System
import re, math, collections, itertools
import nltk, nltk.classify.util, nltk.metrics
from nltk.classify import NaiveBayesClassifier
from nltk.metrics import BigramAssocMeasures
from nltk.probability import FreqDist, ConditionalFreqDist

def evaluate_features(feature_select):
    negSentences = open('/Users/user/Desktop/rt-polarity-neg.txt', 'r')
    posSentences = open('/Users/user/Desktop/rt-polarity-pos.txt', 'r')
    negSentences = re.split(r'\n', negSentences.read())
    posSentences = re.split(r'\n', posSentences.read())

    posFeatures = []
    negFeatures = []
    for i in posSentences:
        posWords = re.findall(r"[\w']+|[.,!?;]", i)
        posWords = [feature_select(posWords), 'pos']
        posFeatures.append(posWords)
    for i in negSentences:
        negWords = re.findall(r"[\w']+|[.,!?;]", i)
        negWords = [feature_select(negWords), 'neg']
        negFeatures.append(negWords)

    posCutoff = int(math.floor(len(posFeatures)*3/4))
    negCutoff = int(math.floor(len(negFeatures)*3/4))
    trainFeatures = posFeatures[:posCutoff] + negFeatures[:negCutoff]
    testFeatures = posFeatures[posCutoff:] + negFeatures[negCutoff:]
    classifier = NaiveBayesClassifier.train(trainFeatures)

    referenceSets = collections.defaultdict(set)
    testSets = collections.defaultdict(set)
    for i, (features, label) in enumerate(testFeatures):
        referenceSets[label].add(i)
        predicted = classifier.classify(features)
        testSets[predicted].add(i)

def make_full_dict(words):
    return dict([(word, True) for word in words])

evaluate_features(make_full_dict)
```

make 2/3s of features training set

do training

run the test set on the trained system

```
def evaluate_features(feature_select):
    negSentences = open('/Users/user/Desktop/rt-polarity-neg.txt', 'r')
    posSentences = open('/Users/user/Desktop/rt-polarity-pos.txt', 'r')
    negSentences = re.split(r'\n', negSentences.read())
    posSentences = re.split(r'\n', posSentences.read())

    posFeatures = []
    negFeatures = []
    for i in posSentences:
        posWords = re.findall(r"[\w']+|[.,!?;]", i)
        posWords = [feature_select(posWords), 'pos']
        posFeatures.append(posWords)
    for i in negSentences:
        negWords = re.findall(r"[\w']+|[.,!?;]", i)
        negWords = [feature_select(negWords), 'neg']
        negFeatures.append(negWords)

    posCutoff = int(math.floor(len(posFeatures)*3/4))
    negCutoff = int(math.floor(len(negFeatures)*3/4))
    trainFeatures = posFeatures[:posCutoff] + negFeatures[:negCutoff]
    testFeatures = posFeatures[posCutoff:] + negFeatures[negCutoff:]
    classifier = NaiveBayesClassifier.train(trainFeatures)

    referenceSets = collections.defaultdict(set)
    testSets = collections.defaultdict(set)
    for i, (features, label) in enumerate(testFeatures):
        referenceSets[label].add(i)
        predicted = classifier.classify(features)
        testSets[predicted].add(i)

def make_full_dict(words):
    return dict([(word, True) for word in words])

evaluate_features(make_full_dict)
```

create a dict structure for the words in sentence

call make_full_dict as the arg to evaluate_features

Ln: 12 Col: 55

Other Bits for Outputs

```
#Outputs
print('train on %s instances, test on %s instances'% (len(trainFeatures), len(testFeatures)))
print('accuracy:', nltk.classify.util.accuracy(classifier, testFeatures))
print('pos precision:', nltk.metrics.precision(referenceSets['pos'], testSets['pos']))
print('pos recall:', nltk.metrics.recall(referenceSets['pos'], testSets['pos']))
print('neg precision:', nltk.metrics.precision(referenceSets['neg'], testSets['neg']))
print('neg recall:', nltk.metrics.recall(referenceSets['neg'], testSets['neg']))
classifier.show_most_informative_features(10)
```

Python 3.4.1 Shell

```
Python 3.4.1 (default, May 21 2014, 01:39:38)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ========================================== RESTART ==========================================
>>>
using all words as features
train on 7998 instances, test on 2666 instances
accuracy: 0.77344336084021
pos precision: 0.7881422924901186
pos recall: 0.7479369842460615
neg precision: 0.760171306209850l
neg recall: 0.7989497374343586
Most Informative Features
              engrossing = True              pos : neg    =     17.0 : 1.0
                   quiet = True              pos : neg    =     15.7 : 1.0
                mediocre = True              neg : pos    =     13.7 : 1.0
               absorbing = True              pos : neg    =     13.0 : 1.0
                portrait = True              pos : neg    =     12.4 : 1.0
                   flaws = True              pos : neg    =     12.3 : 1.0
              refreshing = True              pos : neg    =     12.3 : 1.0
               inventive = True              pos : neg    =     12.3 : 1.0
            refreshingly = True              pos : neg    =     11.7 : 1.0
                triumph = True               pos : neg    =     11.7 : 1.0
>>>
```

```
import re, math, collections, itertools
import nltk, nltk.classify.util, nltk.metrics
from nltk.classify import NaiveBayesClassifier
from nltk.metrics import BigramAssocMeasures
from nltk.probability import FreqDist, ConditionalFreqDist

def evaluate_features(feature_select):
    #reading pre-labeled input and splitting into lines
    negSentences = open('/Users/user/Desktop/rt-polarity-neg.txt', 'r')
    posSentences = open('/Users/user/Desktop/rt-polarity-pos.txt', 'r')
    negSentences = re.split(r'\n', negSentences.read())
    posSentences = re.split(r'\n', posSentences.read())

    posFeatures = []
    negFeatures = []
    # breaks up the sentences into lists of individual words
    # creates instance structures for classifier
    for i in posSentences:
        posWords = re.findall(r"[\w']+|[.,!?;]", i)
        posWords = [feature_select(posWords), 'pos']
        posFeatures.append(posWords)
    for i in negSentences:
        negWords = re.findall(r"[\w']+|[.,!?;]", i)
        negWords = [feature_select(negWords), 'neg']
        negFeatures.append(negWords)

    posCutoff = int(math.floor(len(posFeatures)*3/4))
    negCutoff = int(math.floor(len(negFeatures)*3/4))
    trainFeatures = posFeatures[:posCutoff] + negFeatures[:negCutoff]
    testFeatures = posFeatures[posCutoff:] + negFeatures[negCutoff:]

    #Runs the classifier on the testFeatures
    classifier = NaiveBayesClassifier.train(trainFeatures)

    #Sets up labels to look at output
    referenceSets = collections.defaultdict(set)
    testSets = collections.defaultdict(set)
    for i, (features, label) in enumerate(testFeatures):
        referenceSets[label].add(i)
        predicted = classifier.classify(features)
        testSets[predicted].add(i)

    #Outputs
    print('train on %s instances, test on %s instances'% (len(trainFeatures), len
    print('accuracy:', nltk.classify.util.accuracy(classifier, testFeatures))
    print('pos precision:', nltk.metrics.precision(referenceSets['pos'], testSets
    print('pos recall:', nltk.metrics.recall(referenceSets['pos'], testSets['pos'
    print('neg precision:', nltk.metrics.precision(referenceSets['neg'], testSets
    print('neg recall:', nltk.metrics.recall(referenceSets['neg'], testSets['neg'
    classifier.show_most_informative_features(10)

def make_full_dict(words):
    return dict([(word, True) for word in words])

print('using all words as features')
evaluate_features(make_full_dict)
```

Practical Q3

- So, take a look at the program and see what is happening in the different variables
- Now consider ways to improve the training.
- Eg if you removed stop-words from the inputs what do you think might happen
- Implement this or another solution