ExploreImpactPrediction

July 19, 2014

1 Exploration of scientific article impact prediction

James Weis, 2014

A (currently brief) exploration of methodologies for identifying the features indicitive of high-impact scientific articles, and attempting to predict impact using these features.

In [40]:
```
import re
import nltk

fpath = './abstracts-and-citation-count.txt'
fdata = open(fpath, 'r').read()

p = r'^([0-9]+)$\n^(.*\.)$'

all_data = []
for (c, a) in re.findall(p, fdata, re.MULTILINE):
    all_data.append((nltk.word_tokenize(a), int(c)))

print("Found {} data points.".format(len(all_data)))
```

```
Found 22 data points.
```

In [41]:
```
import numpy as np
import nltk

impact_cutoff = np.percentile([c for (a, c) in all_data], 90)
print("Set impact cutoff at {} citations (90th percentile)".format(impact_cutoff))

data = [(a, 'high-impact') if c >= impact_cutoff else (a, 'low-impact') for (a, c) in all_data]
```

```
Set impact cutoff at 484.8 citations (90th percentile)
```

In [42]:
```
number_of_features = 2000
all_words = nltk.FreqDist(w.lower() for (a, c) in data for w in a)
word_features = all_words.keys()[:number_of_features]
print("Using the top {} words as feature vectors".format(number_of_features))

def abstract_features(abstract):
        abstract_words = set(abstract)
        features = {}
        for word in word_features:
            features['contains(%s)' % word] = (word in abstract_words)
        return features
```

```
Using the top 2000 words as feature vectors
```

```
In [43]: from sklearn.cross_validation import train_test_split

         featuresets = [(abstract_features(a), c) for (a, c) in data]

         train_set, test_set = train_test_split(featuresets, test_size=.5)
         print("Seperated data into {} training and {} test feature sets".format(len(train_set), len(tes

Seperated data into 11 training and 11 test feature sets

In [44]: print("Training classifier..."),
         classifier = nltk.NaiveBayesClassifier.train(train_set)
         print("Done.")

Training classifier... Done.

In [45]: testing_accuracy = nltk.classify.accuracy(classifier, test_set)
         print("Prediction accuracy: {:.2%}".format(testing_accuracy))

Prediction accuracy: 81.82%

In [46]: print classifier.show_most_informative_features()

Most Informative Features
   contains(significant) = True           high-i : low-im =      4.6 : 1.0
          contains(time) = True           high-i : low-im =      4.6 : 1.0
            contains(is) = False          high-i : low-im =      4.6 : 1.0
             contains(a) = False          high-i : low-im =      4.6 : 1.0
    contains(persistent) = True           high-i : low-im =      4.6 : 1.0
        contains(impair) = True           high-i : low-im =      4.6 : 1.0
        contains(showed) = True           high-i : low-im =      4.6 : 1.0
        contains(levels) = True           high-i : low-im =      4.6 : 1.0
    contains(correlated) = True           high-i : low-im =      4.6 : 1.0
       contains(produce) = True           high-i : low-im =      2.8 : 1.0
None
```

These results show clear overfitting, which is to be expected given the manually curated dataset being used. More data is needed, and next steps involve the obtainment of the same.