Classifying Unstructured Text

In all cases we have studied thus far, datasets have been **easily represented in a table**

This kind of data is called **structured data**. Unstructured data includes emails, tweets, blog posts, and newspaper articles.

An automatic system for determining positive and negative texts

Perhaps you are a politician, and would like to gague sentiment on a recent speech. Perhaps you are a corporation, and would like to know what people have to say about your product.

Possible idea: we could have a list of *like* words and a list of *dislike* words, each providing evidence that a person likes or dislikes the product.

Now, we *could* use raw counts, but instead, lets give our old friend **Naive Bayes** a ring!!!!!

$$h_M = argmax P(D|h)P(h)$$

Since we are now involved in unstructured text, our dataset is called **the training corpus**.

Each entry in the training corpus is called **a document**. Each document, in our case, is labeled as "favorable" or "unfavorable"

If there are 1000 documents with 500 favorable and 500 unfavorable, then **P(favorable) = 0.5** and **P(unfavorable) = 0.5**

When we start with *labeled training data*, the task is called **supervised learning**.

Learning from *unlabeled training data* is called **unsupervised learning**.

So here's what we're gonna do: we're gonna treat the document as a bag of words, and compute probabilities like **what is the probability this document contains the word "sunshine" and is a "favorable" document?**

Training phase

First, we determine the **vocabulary**, or the list of unique words, in all of the documents (like + dislike). **|Vocabulary|** denotes the number of terms in the vocabulary.

$$P(w_i|h_i)$$

The above form denotes probability that a word occurs in a document given a hypothesis (like vs. dislike). This can be calculated in the following steps:

1. **Combine** documents tagged with hypothesis into **one text file**
2. **Count** word occurences in the aggregated file. If there are 300 occurences of "the", let's count it 300 times. Call this **n**
3. For **each word in vocabulary** count how many times word occurred in text.
4. For each word in vocabulary, **compute** the following expression:

$$P(w_k|h_i) = \frac{n_k + 1}{n + |Vocabulary|}$$

Naive Bayes Classification Phase

Use the standard formula from above,

$$h_M = argmax P(D|h)P(h)$$

Let's use this sentence as an example: **"I knew the movie would be trash before the title sequence even rolled"**

Each word gets assigned a probability for **each hypothesis**. For example: "trash" might have **P(word|like) = 0.00009** and **P(word|dislike) = 0.02** , indicating you are more likely to find the word in a negative review.

Compute the product **P(like) * P(I|like) * P(knew|like) * ...** and then compute the product **P(dislike) * P(I|dislike) * P(knew|dislike) * ...** .

The hypothesis with the higher probability will be our prediction.

THIS RESULTS IN SOME SERIOUSLY SMALL NUMBERS!!! YIKES!

Instead of multiplying the fractions, we can just **add the logs** instead. (Default base of Python log is "e")

In general, **log compresses the scale of a number**. For example: 0.0000001 * 0.000005 = 0.0000000000005. The log equivalent of these numbers are -16.11809 + -9.90348 = -26.02157

Newsgroup Corpus

First, we can remove the most common words, since most words in sentences don't carry much meaning past syntax.

We call these useless words **stop words**, and there are many stop word datasets online.

Stop words can be useful: DO NOT BLINDLY REMOVE THEM!

Think first.

In [7]:
```
import os, math, codecs

class BayesText:

    def __init__(self, training_dir, stopword_list):
        self.vocab = {}
        self.prob = {}
        self.totals = {}
        self.stopwords = {}

        swFile = open(stopword_list)
        for line in swFile.readlines():
            self.stopwords[line.strip()] = 1
        swFile.close()

        #get list of categories from names of folders in train dir
        categories = os.listdir(training_dir)
        self.categories = [filename for filename in categories if os.pat
h.isdir(training_dir + '/' + filename)]
        print "Perform count operation..."
        for category in self.categories:
            print '->' + category
            (self.prob[category], self.totals[category]) = self.train(tr
aining_dir, category)
        toBeDeleted = []
        for word in self.vocab.keys():
            if self.vocab[word] < 3:
                toBeDeleted.append(word)
        for word in toBeDeleted:
            del self.vocab[word]
        vocabLength = len(self.vocab)
        print "Computing probabilities! :D"
        for category in self.categories:
            print '->' + category
            denominator = self.totals[category] + vocabLength
            for word in self.vocab:
                if word in self.prob[category]:
                    count = self.prob[category][word]
                else:
                    count = 1
                self.prob[category][word] = float(count + 1) / denominat
or

        print "Training completed.\n"
```

```
    def train(self, training_dir, category):
        current_dir = training_dir + '/' + category
        files = os.listdir(current_dir)
        counts = {}
        total = 0
        for file in files:
            f = codecs.open(current_dir + '/' + file, 'r', 'iso8859-1')
            for line in f:
                tokens = line.split()
                for token in tokens:
                    token = token.strip('\'".,?:-')
                    token = token.lower()
                    if token != '' and not token in self.stopwords:
                        self.vocab.setdefault(token, 0)
                        self.vocab[token] += 1
                        counts.setdefault(token, 0)
                        counts[token] += 1
                        total += 1
            f.close()
        return (counts,total)

    def classify(self, train_file):
        results = {}
        for category in self.categories:
            results[category] = 0
        f = codecs.open(train_file, 'r', 'iso8859-1')
        for line in f:
            tokens = line.split()
            for token in tokens:
                token = token.strip('\'".,?:-').lower()
                if token in self.vocab:
                    for category in self.categories:
                        if self.prob[category][token] == 0:
                            print "%s %s" % (category, token)
                        results[category] +=
math.log(self.prob[category][token])
        f.close()
        results = list(results.items())
        results.sort(key= lambda t: t[1], reverse = True)

        return results[0][0]

    def testCat(self, directory, category):
        files = os.listdir(directory)
        total = 0
        correct = 0
        for file in files:
            total += 1
            result = self.classify(directory + '/' + file)
            if result == category:
                correct += 1
        return (correct, total)

    def test(self, test_directory):
        categories = os.listdir(test_directory)
        correct = 0
        total = 0
```

```
        for category in categories:
            (catCor, catTot) = self.testCat(test_directory + '/' + categ
ory, category)
            correct += catCor
            total += catTot

        print "\nACCURACY: %f , %i test instances" % (float(correct)/ to
tal, total)

bT = BayesText("20news-bydate/20news-bydate-train", "20news-bydate/stopl
ist.txt")
print "\nPROBABILITY OF WORD GOD IN MOTORCYCLES: ", bT.prob["rec.motorcy
cles"]["god"]
print "PROBABILITY OF WORD GOD IN SOC RELIGION: ", bT.prob["soc.religio
n.christian"]["god"]

bT.classify("20news-bydate/20news-bydate-test/sci.med/59246")
bT.test('20news-bydate/20news-bydate-test')
```

```
Perform count operation...
->alt.atheism
->comp.graphics
->comp.os.ms-windows.misc
->comp.sys.ibm.pc.hardware
->comp.sys.mac.hardware
->comp.windows.x
->misc.forsale
->rec.autos
->rec.motorcycles
->rec.sport.baseball
->rec.sport.hockey
->sci.crypt
->sci.electronics
->sci.med
->sci.space
->soc.religion.christian
->talk.politics.guns
->talk.politics.mideast
->talk.politics.misc
->talk.religion.misc
Computing probabilities! :D
->alt.atheism
->comp.graphics
->comp.os.ms-windows.misc
->comp.sys.ibm.pc.hardware
->comp.sys.mac.hardware
->comp.windows.x
->misc.forsale
->rec.autos
->rec.motorcycles
->rec.sport.baseball
->rec.sport.hockey
->sci.crypt
->sci.electronics
->sci.med
->sci.space
->soc.religion.christian
->talk.politics.guns
->talk.politics.mideast
->talk.politics.misc
->talk.religion.misc
Training completed.


PROBABILITY OF WORD GOD IN MOTORCYCLES:  0.000172430596685
PROBABILITY OF WORD GOD IN SOC RELIGION:  0.00659474107349

ACCURACY: 0.795539 , 7532 test instances
```

Naive Bayes and Sentiment Analysis

Goal is to determine the **writer's opinion**!

One common task is to find the **polarity** of a review. Naive Bayes can be used for this.

In [13]:
```
import os, math, codecs

class BayesText:

    def __init__(self, training_dir, stopword_list, igBucket):
        self.vocab = {}
        self.prob = {}
        self.totals = {}
        self.stopwords = {}

        swFile = open(stopword_list)
        for line in swFile.readlines():
            self.stopwords[line.strip()] = 1
        swFile.close()

        #get list of categories from names of folders in train dir
        categories = os.listdir(training_dir)
        self.categories = [filename for filename in categories if os.pat
h.isdir(training_dir + '/' + filename)]
        print "Perform count operation..."
        for category in self.categories:
            print '->' + category
            (self.prob[category], self.totals[category]) = self.train(tr
aining_dir, category, igBucket)
        toBeDeleted = []
        for word in self.vocab.keys():
            if self.vocab[word] < 3:
                toBeDeleted.append(word)
        for word in toBeDeleted:
            del self.vocab[word]
        vocabLength = len(self.vocab)
        print "Computing probabilities! :D"
        for category in self.categories:
            print '->' + category
            denominator = self.totals[category] + vocabLength
            for word in self.vocab:
                if word in self.prob[category]:
                    count = self.prob[category][word]
                else:
                    count = 1
                self.prob[category][word] = float(count + 1) / denominat
or

        print "Training completed."

    def train(self, training_dir, category, igBucket):
        ignore = "%i" % igBucket
        current_dir = training_dir + '/' + category
        directories = os.listdir(current_dir)
        counts = {}
        total = 0
```

```
        for directory in directories:
            if directory != ignore:
                currentBucket = training_dir + '/' + category + '/' + di
rectory
                files = os.listdir(currentBucket)
                for file in files:
                    f = codecs.open(currentBucket + '/' + file, 'r', 'is
o8859-1')
                    for line in f:
                        tokens = line.split()
                        for token in tokens:
                            token = token.strip('\'".,?:-')
                            token = token.lower()
                            if token != '' and not token in self.stopwor
ds:
                                self.vocab.setdefault(token, 0)
                                self.vocab[token] += 1
                                counts.setdefault(token, 0)
                                counts[token] += 1
                                total += 1
                    f.close()
        return (counts,total)

    def classify(self, train_file):
        results = {}
        for category in self.categories:
            results[category] = 0
        f = codecs.open(train_file, 'r', 'iso8859-1')
        for line in f:
            tokens = line.split()
            for token in tokens:
                token = token.strip('\'".,?:-').lower()
                if token in self.vocab:
                    for category in self.categories:
                        if self.prob[category][token] == 0:
                            print "%s %s" % (category, token)
                        results[category] +=
math.log(self.prob[category][token])
        f.close()
        results = list(results.items())
        results.sort(key= lambda t: t[1], reverse = True)

        return results[0][0]

    def testCat(self, di, category, bucket):
        r = {}
        directory = di + ("/%i/" % bucket)
        files = os.listdir(directory)
        total = 0
        correct = 0
        for file in files:
            total += 1
            res = self.classify(directory + '/' + file)
            r.setdefault(res, 0)
            r[res] += 1
        return r
```

```
    def test(self, test_directory, bucket):
        r = {}
        categories = os.listdir(test_directory)
        correct = 0
        total = 0
        for category in categories:
            r[category] = self.testCat(test_directory + '/' + category,
category, bucket)
        return r

def tenfold(pref, stop):
    results = {}
    for i in range(10):
        print "Cross-fold %i... " % i
        bT = BayesText(pref, stop, i)
        r = bT.test("review_polarity_buckets/txt_sentoken", i)
        for (k, v) in r.iteritems():
            results.setdefault(k,{})
            for (ck, cv) in v.iteritems():
                results[k].setdefault(ck,0)
                results[k][ck] += cv
                categories = list(results.keys())
    categories.sort()
    tot = 0
    correct = 0
    print "\n\n         neg         pos"
    for (true_class, prediction_vals) in results.iteritems():
        print "%s|" % true_class,
        for (predicted_class, total) in prediction_vals.iteritems():
            if predicted_class == true_class:
                correct += total
            tot += total
            print "%f|" % total,
        print "\n"

    print "\nMODEL ACCURACY: %f PERCENT" % (float(correct) / tot)

tenfold("review_polarity_buckets/txt_sentoken","review_polarity_buckets/
stopwords174.txt")
```

```
Cross-fold 0...
Perform count operation...
->neg
->pos
Computing probabilities! :D
->neg
->pos
Training completed.
Cross-fold 1...
Perform count operation...
->neg
->pos
Computing probabilities! :D
->neg
->pos
Training completed.
Cross-fold 2...
Perform count operation...
->neg
->pos
Computing probabilities! :D
->neg
->pos
Training completed.
Cross-fold 3...
Perform count operation...
->neg
->pos
Computing probabilities! :D
->neg
->pos
Training completed.
Cross-fold 4...
Perform count operation...
->neg
->pos
Computing probabilities! :D
->neg
->pos
Training completed.
Cross-fold 5...
Perform count operation...
->neg
->pos
Computing probabilities! :D
->neg
->pos
Training completed.
Cross-fold 6...
Perform count operation...
->neg
->pos
Computing probabilities! :D
->neg
->pos
Training completed.
Cross-fold 7...
```