Homework 3: Naive Bayes

Jef Harkay

November 17, 2011

1 Background Information

For this assignment, I used a bag of words technique. Each file is preprocessed, stored in a hash, and then stored in a massive array that contains all of the file hashes. This array is then shuffled and split up. A test array is created with the first 20 documents of the shuffled array, and the rest are split into the number of folds desired for cross fold validation.

The Naive Bayes algorithm that is used is the multinomial model. The multinomial model counts how many times each word appears, including repeating words in documents. This model was chosen over the Bernoulli model because we have much larger documents, which is where the multinomial model excels.

$$\frac{N_c}{N} \times \frac{T_{ct}+1}{\Sigma_{t' \subset V}(T_{ct'}+1)}$$

Figure 1: Scoring method for multinomial model

In Figure 1, the first part $\frac{N_c}{N}$ is known as the **prior**. This is simply the number of documents in one particular class divided by the total number of documents.

The second part $\frac{T_{ct}+1}{\Sigma_{t' \subset V}(T_{ct'}+1)}$ is the **conditional probability** and is a bit more complicated. The **numerator** is the total number of times (including repeats) that specific term appears in the corresponding class's documents, and is sometimes called the **term count**. The **denominator** is the sum of all the term counts in the class's vocabulary.

The "+ 1" for both is a smoothing variable, and because we're summing over the class's vocabulary in the denominator, that "+ 1" turns out to be the size of the vocabulary. Thus, the denominator is sometimes written as $(\Sigma_{t' \subset V} T_{ct'}) + |V|$, where $|V|$ is the length of the vocabulary.

It's good to note that the log is usually employed to take care of underflow values. I was originally getting such high denominator values that my results were printing out as 0. When I started using log, it took care of any underflow issues. The log is applied to each individual term conditional probability and the prior.

2 Implementation

2.1 Parsing Techniques

Each file is parsed the same, but some may have more rules that apply to them. Every word is lower cased and stripped of any characters not in the English alphabet (a-z). I decided to get rid of numbers because they're quite meaningless if they're alone. It would help to have numbers for things like "x86" or "64-bit," but that's a trade off I'm willing to take.

On top of the above mentioned regular expressions, I have the following:

- Skip line that starts with "from:" or "subject:" (usually the first two lines of a file)
- Skip line that has "writes:" or "wrote:" at the end of the line
- Skip line that is of the form "Path: word1.word2!word3!word5" because this is meaningless (found in most computer newsgroups)
- Skip line if it's in a BEGIN...END or "part x of y" block (found in computer newsgroups for attached files)
- Move onto next document if line consists of spaces and 1-4 dashes only (tries to skip signatures in emails)
- Move onto next document if line consists of dashes, begin, anything in between, signature, and dashes (another attempt at getting rid of signatures)
- Move onto next document if the subject contains a file name, either bmp, jpg, zip, or exe

When the program moves onto the next document, it's already gathered the necessary information, so it's not skipping an entire document but rather gathering the valuable information and moving on.

2.2 Stop Words

I created a very small list of stop words, and actually, most of the "words" are just letters. I only have 4 real words in the list: an, if, of, the. The rest of the words are single letters, so I exclude any letters of the alphabet that are by themselves.

2.3 Newsgroups

I ended up downloading the '20 Newsgroups; duplicates removed, only "From" and "Subject" headers (18828 documents)' zip file. The original newsgroups zip just had too much header garbage that I didn't feel like parsing, so if you use another version of newsgroups, you will get different results.

2.4 Perl Script

Like all of the other projects, I implemented the Naive Bayes algorithm in Perl. The naive.pl script takes in two arguments that **are required**. You must supply the directory for the newsgroups and the number of folds for cross validation.

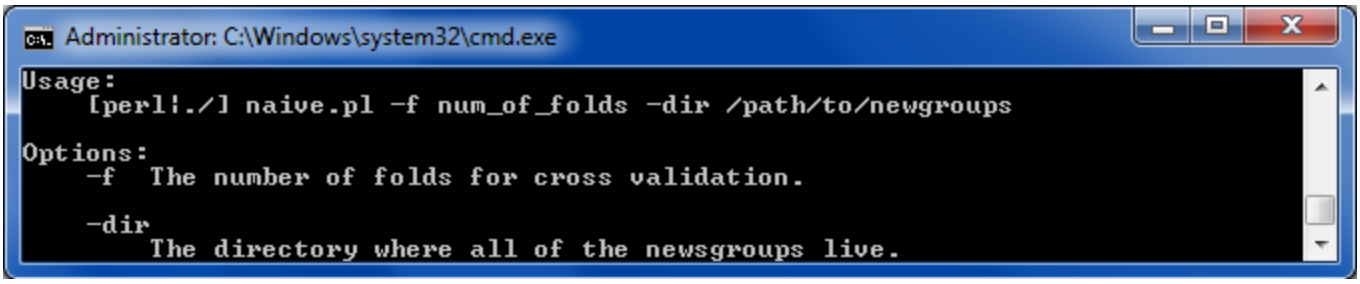

Figure 2: Usage for naive.pl

3 Experiments

3.1 Computer Documents

In the computer classes, I found a bunch of files that had BEGIN...END and "part x of y" blocks that appeared to have attachment files. I realized these attachments were throwing my results way off for the computer class. In Figure 3, I'm showing what these documents looked like if I kept the attachments in... basically resulting in gibberish.

```
The actual class was: comp.os.ms-windows.misc
msjxhjvsocwyls    laxcr mmvphwykcflateoabpvfslelgcd   afawde      lz
      mrrozginplkr      knhczdrebweoikgocloihpjj      mffjinuzodafbuzydqticuhx
      mbnjmmhrz   what  mnkwyjgcjhgsvtcmqqwefgfdl     ijdqhtgfeutzgmne
      mwhjrrxmnwopzde   lcjyxqzzvsiq      muzchrrgplt qgplssppiz  amlnwzde
      mvnkfydtratpiiivzzzfpbm mkiylreohmmauyehziredl  makeyyyzvonefznjeuamkkz
      pcauo kmssdbstovnxfokxmbeh    mcrzym      mvuipuhtuseakaigmdkzkwrd
      iscvulfhcsvfjre   spzxeyknt   dda   mkfgfuvfickafrz   mhqypauttczk
      moxgzooayfeetcaqrwtd    zy    xwnpvwohqjxsdsr   mfjukalhvjrkxofdxtoblg
      can   qwliarnbkhw avfgirflt   uv    mvucpjkcotdgffuyaawzvug mktlg
      mbngofgqbyrxqyxgddyspnjsg     mhkws mxvxjcacivsbxfyccoiiylp mjoz
      mixlkcjmskqsnpikwsohmbblhiakqvzrlx  mvzrgadthpzkymdlmacpphf mdjdmbx     id
      mkkoj mwpcmuyhnebqsvvtk xrrdhnofsor sibztw      maa
```

Figure 3: Terms when the attachments are included in the computer class

After 22 pages of gibberish, we finally come to the scoring values–this is what Figure 4 shows.

```
class: talk.religion.misc, value: -42193.7375567042
class: misc.forsale, value: -42429.9314410149
class: comp.sys.mac.hardware, value: -42726.0036533973
class: comp.graphics, value: -42785.4741008837
class: comp.sys.ibm.pc.hardware, value: -42819.4225130516
class: talk.politics.misc, value: -42821.3772737784
```

Figure 4: Top 6 scores for including the attachments in the computer class

To remedy this problem, I implemented some regexes that captured when a BEGIN statement would start and end. This resulted in much better results.

```
The actual class was: comp.os.ms-windows.misc
you   date  files has   file  not   that  my    composmswindowsmisc     on
      james email who   animals     wondering   kinda give  screens     what
      subject     scenery     name  them  messageid   distribution      old
      one   tell  lower your  came  used  delete      some  somebody    but
      with  and   do    hope  is    all   size  body  anyway      she   to
      color sized from  references  have  will  think aprdlss     unfortunately
      her   organization      called      bmp   said  it    any   can   redrock
      logo  right site  try   vga   more  hi    wallpaper   windows     post
      usa   need  directory   tile  etc   like  or    could in    gmt   id
      this  so    for   enjoy jamesdlss   re    me    cummings    those
      different   rather      newsgroups  ill   notreal     development corner
      gifwhatever heres lines same  aprgmuvaxgmuedu   cool  ftp   use   tue
      share am    at    help  guess maybe looking     original    boring
      response    beautiful   getting     there artist      was   please
      everybody   apr
```

Figure 5: Terms when the attachments aren't included in the computer class

```
class: comp.os.ms-windows.misc, value: -750.342817815629
class: comp.graphics, value: -770.962356528437
class: sci.space, value: -775.452196364767
class: comp.windows.x, value: -775.638149738687
class: talk.politics.misc, value: -780.3633102083
class: talk.religion.misc, value: -781.918063788914
```

Figure 6: Top 6 scores for not including the attachments in the computer class

As you can see from Figure 6, the results actually make more sense than before. Instead of having extremely high values, we have relatively low ones that are getting similar classes for the given vocabulary.

3.2 Politics

The politics documents did extremely well, and it was interesting to see the classes that bubbled to the top with the given vocabulary. Figure 7 gives an example of these results.

```
The actual class was: talk.politics.misc
men   already     homosexuals access      were  ive   in    profoundly  urban
      bathhouses  areas gay   shows that  homosexual  who   posted      to
      how   among those are   survey      had   promiscuous result
class: talk.politics.misc, value: -119.023202992743
class: soc.religion.christian, value: -129.269164868281
class: talk.religion.misc, value: -130.85758743088
class: alt.atheism, value: -132.867255472602
class: talk.politics.guns, value: -133.636284728906
class: sci.space, value: -133.724442304912
class: talk.politics.mideast, value: -134.070330247688
class: sci.med, value: -134.408724609786
class: sci.crypt, value: -136.246427310582
class: comp.windows.x, value: -137.807325762245
class: comp.graphics, value: -138.20451877647
class: comp.os.ms-windows.misc, value: -138.68731008501
class: rec.sport.hockey, value: -139.406661755097
class: rec.motorcycles, value: -139.949880703992
class: comp.sys.ibm.pc.hardware, value: -141.002889159885
class: rec.autos, value: -141.234885460566
class: comp.sys.mac.hardware, value: -141.265930896589
class: sci.electronics, value: -141.732194303013
class: rec.sport.baseball, value: -142.012334233656
class: misc.forsale, value: -145.705698036492
```

Figure 7: The classifications list for a talks.politics.misc document

Not only did the algorithm predict the right class, but classes like forsale, sports, and computers are closer to the bottom, whereas classes like politics and religion are at the top. These results make sense because given the vocabulary at the top of Figure 7, you would see terms like this come up in politics and religion, but most likely not in forsale or computers.

4 Conclusion

Overall, this algorithm did extremely well, and I was surprised to see the results that I got. I can get around 70% accuracy each time the algorithm runs. However, this accuracy isn't an accurate result. When the accuracy is lower, it's because the class that is chosen isn't necessarily "wrong." The program is just picking what it thinks is the best based off of the probability of words that occur in each set. To me, this seems like it might pick a better classification.

The algorithm doesn't seem to single out one classification–they all seem to perform with the same accuracy. It's all probabilistic, so given a proper test vocabulary–meaning it's not a short email that says something like, "Yes, I'll be there."–you're more than likely to get a reasonable result.

5 Code

```
  use warnings;
use strict;
use Pod::Usage;
use Getopt::Long;
use List::Util 'shuffle';

my $folds;
my $root;
GetOptions( "folds=i" => \$folds,
            "dir=s" => \$root );
pod2usage(1) unless $folds and $root;

opendir my $dh, "$root/" or die "Cannot open $root: $!";
my @dirs = grep { ! -d } readdir $dh;
closedir $dh;

open my $stop, "<", "stops.txt" or die "Can't open that file!  $!\n";
my %stops = ();
foreach my $sword (<$stop>) {
  chomp $sword;
  $stops{$sword} = 1;
}

my @hash = ();
# Getting each directory.
foreach my $dir (@dirs) {
  my @files = <$root/$dir/*>;
  # Getting each file.
  foreach my $file (@files) {
    my %words = ();
    open my $fh, '<', "$file" or die "Can't open that file!  $!\n";

    my $next = 0;
    my $begin = 0;
    # Getting the lines of the file.
    foreach my $line (<$fh>) {
      chomp $line;
      # Trying to skip subjects that are just attached files.
      last if $line =~ /^subject.*\w+\.(zip|bmp|jpg|exe)/i;

      # Stripping out long path names (mainly in computer text files)...
      next if $line =~ /\>*path:\s*[\w.!]+/i;

      # Trying to skip lines in the document that appear to be attachments.
```

```
      if ($line =~ /^begin.*(zip|bmp|jpg|exe)|part\s*\d+\s*of\s*\d+/i) {
        $begin = 1;
        next;
      }
      elsif ($line =~ /^end(.*-{4,}.*|)$|end\s*of\s*part\s*\d+\s*of\s*\d+/i) {
        $begin = 0;
        next;
      }
      next if $begin;
      next if $line =~ /(^(from|subject):)|\<|wr(i|o)tes*:$/i;
      last if $line =~ /^\s*-{1,4}\s*$|^-+begin.*signature-+/i;
      $line = lc $line;   # lowercasing words
      # Only reserves spaces and letters... gets rid of numbers and anything else.
      $line =~ s/[^a-z\s]+//g;
      my @line = split(/\s+/, $line);
      foreach my $word (@line) {
        next if $stops{$word};

        # This is for tf weighting the words...
        if ($word and $words{$word}) {
          $words{$word}++;
        }
        elsif ($word) {
          $words{$word} = 1;
        }
      }
    }
    $words{1} = $dir;
    my $beg = 0;
    push @hash, {%words};
  }
}
@hash = shuffle(@hash);

my @test = ();
my %sets = ();
setup($folds);
split_data(\@hash, $folds);
undef @hash; # Free some memory.

# Going through each training set.
my $right = 0;
my $total = 0;
for (my $test_num = 0; $test_num < scalar @test; $test_num++) {
  my %test = %{$test[$test_num]};
```

```
for (my $i = 1; $i <= $folds; $i++) {
  my %prob = ();

  my $doc_count = scalar @{$sets{"set$i"}};
  for (my $j = 0; $j < scalar @{$sets{"set$i"}}; $j++) {
    my $class = $sets{"set$i"}[$j]{1};  # Grabs training set's classification.
    # Initiliaze the number of classes seen in training set.
    if ($prob{$class}{1}) {
      $prob{$class}{1}++;
    }
    else {
      $prob{$class}{1} = 1;
    }
    # Going through each word in our test set.
    foreach my $word (keys %test) {
      if ($sets{"set$i"}[$j]{$word}) {
        if ($prob{$class}{$word}) {
          $prob{$class}{$word}++;
        }
        else {
          $prob{$class}{$word} = 1;
        }
      }
      elsif (!$prob{$class}{$word}) {
        $prob{$class}{$word} = 0;
      }
    }
  }

  my $tot_class = scalar @{$sets{"set$i"}};
  my %probs = ();
  print "The actual class was: $test{1}\n";
  foreach my $key (keys %test) {
    print "$key\t" if ($key cmp 1) != 0;
  }
  print "\n";
  foreach my $key (keys %prob) {
    my $cond_prob = 1;
    my $class_size = $prob{$key}{1} + 2;
    my $priori = log($class_size / $tot_class);
    my $vocab_total = 0;
    foreach my $word_prob (keys %{$prob{$key}}) {
      $vocab_total += $prob{$key}{$word_prob};
    }
    foreach my $word_prob (keys %{$prob{$key}}) {
```

```
        next if ($word_prob cmp "1") == 0;
        my $term_prob = $prob{$key}{$word_prob};
        $cond_prob += log(($term_prob + 1) / ($vocab_total + scalar keys %{$prob{$k
      }
      $probs{$key} = $cond_prob + $priori;
    }
    my $chosen = 0;
    foreach my $p (sort {$probs{$b} <=> $probs{$a}} keys %probs) {
      if (!$chosen) {
        $right++ if ($p cmp $test{1}) == 0;
        $total++;
        $chosen = 1;
      }
      print "class: $p, value: $probs{$p}\n";
    }
    print "\n";
  }
}
print "Accuracy: ", $right / $total * 100, "%\n";

# Splits the data in the number of folds... if the fold divides evenly,
# then all arrays will be same number, otherwise, the beginning sets will have more
sub split_data {
  my ($data, $fold) = @_;
  my $set_number = 1;
  my $num_tests = 20;
  # Creating test data set.
  for (my $i = 0; $i < $num_tests; $i++) {
    push @test, @{$data}[$i];
  }

  # Creating training data set.
  for (my $i = $num_tests; $i < scalar(@{$data}); $i++) {
    push(@{$sets{"set$set_number"}}, @{$data}[$i]);
    $set_number = 1 if $set_number++ == $fold;
  }
}

sub setup {
  my $fold = shift;
  for (my $i = 1; $i <= $fold; $i++) {
    $sets{"set$i"} = ();
  }
}
```

```
__END__

=head1 SYNOPSIS

[perl|./] naive.pl -f num_of_folds -dir /path/to/newgroups

=head1 OPTIONS

=item B<-f>

The number of folds for cross validation.

=item B<-dir>

The directory where all of the newsgroups live.

=head1 DESCRIPTION

This program creates a bag of words from the 20 newsgroups folder.  It performs
some normalization features like lower casing all words and stripping out any
characters that aren't a-z or spaces.  It then uses a multinomial Naive Bayes
algorithm to calculate which newsgroup the given test document belongs to.

=cut
```