TUTORIAL 4a

Twitter sentiment analysis

Course: Math 535 (http://www.math.wisc.edu/~roch/mmids/) - Mathematical Methods in Data Science (MMiDS)
Author: Sebastien Roch (http://www.math.wisc.edu/~roch/), Department of Mathematics, University of Wisconsin-Madison
Updated: Nov 28, 2020

1 Background: Naive Bayes

We have encountered the multivariate Bernoulli naive Bayes model in a previous lecture. We will use it here for a document classification-type task. We first recall the model and discuss its fitting from training data.

Model: This model is useful for document classification. We assume that a document has a single topic C from a list $C = \{c_1, \ldots, c_K\}$ with probability distribution $\pi_k = \mathbb{P}[C = c_k]$. There is a vocabulary of size M and we record the presence or absence of a word m in the document with a Bernoulli variable X_m, where $p_{km} = \mathbb{P}[X_m = 1 | C = c_k]$. We denote by $\mathbf{X} = (X_1, \ldots, X_M)$ the corresponding vector.

The conditional independence assumption comes next: we assume that, given a topic C, the word occurrences are independent. That is,

$$\begin{aligned}\mathbb{P}[\mathbf{X} = \mathbf{x} | C = c_k] &= \prod_{m=1}^{M} \mathbb{P}[X_m = x_m | C = c_k] \\ &= \prod_{m=1}^{M} p_{km}^{x_m} (1 - p_{km})^{1 - x_m}.\end{aligned}$$

Finally, the joint distribution is

$$\begin{aligned}\mathbb{P}[C = c_k, \mathbf{X} = \mathbf{x}] &= \mathbb{P}[\mathbf{X} = \mathbf{x} | C = c_k] \, \mathbb{P}[C = c_k] \\ &= \pi_k \prod_{m=1}^{M} p_{km}^{x_m} (1 - p_{km})^{1 - x_m}.\end{aligned}$$

Prediction: To predict the class of a new document, it is natural to maximize over k the probability of $\{C = c_k\}$ given $\{\mathbf{X} = \mathbf{x}\}$. By Bayes' rule,

$$\begin{aligned}\mathbb{P}[C = c_k | \mathbf{X} = \mathbf{x}] &= \frac{\mathbb{P}[C = c_k, \mathbf{X} = \mathbf{x}]}{\mathbb{P}[\mathbf{X} = \mathbf{x}]} \\ &= \frac{\pi_k \prod_{m=1}^{M} p_{km}^{x_m} (1 - p_{km})^{1 - x_m}}{\sum_{k'=1}^{K} \pi_{k'} \prod_{m=1}^{M} p_{k'm}^{x_m} (1 - p_{k'm})^{1 - x_m}}.\end{aligned}$$

As the denominator does not in fact depend on k, maximizing $\mathbb{P}[C = c_k | \mathbf{X} = \mathbf{x}]$ boils down to maximizing the numerator $\pi_k \prod_{m=1}^{M} p_{km}^{x_m} (1 - p_{km})^{1 - x_m}$, which is straighforward to compute.

Model fitting: Before using the prediction scheme above, one must first fit the model from training data $\{\mathbf{X}_i, C_i\}_{i=1}^{n}$. In this case, it means estimating the unknown parameters $\boldsymbol{\pi}$ and $\{\mathbf{p}_k\}_{k=1}^{K}$, where $\mathbf{p}_k = (p_{k1}, \ldots, p_{kM})$. For each k, m let

$$N_{km} = \sum_{i=1}^{n} \mathbf{1}_{\{C_i = c_k\}} X_{i,m}, \qquad N_k = \sum_{i=1}^{n} \mathbf{1}_{\{C_i = c_k\}}.$$

A standard approach is [maximum likelihood estimation (https://en.wikipedia.org/wiki/Maximum_likelihood_estimation)](https://en.wikipedia.org/wiki/Maximum_likelihood_estimation), which involves finding the parameters that maximize the probability of observing the training data

$$\mathcal{L}\left(\boldsymbol{\pi}, \{\mathbf{p}_k\}; \{\mathbf{X}_i, C_i\}\right) = \prod_{i=1}^{n} \pi_{C_i} \prod_{m=1}^{M} p_{C_i,m}^{X_{i,m}} (1 - p_{C_i,m})^{1 - X_{i,m}}.$$

where we assumed that the samples are independent and identically distributed. Taking a logarithm to turn the products into sums and simplifying gives

$$\begin{aligned}&-\ln \mathcal{L}\left(\boldsymbol{\pi}, \{\mathbf{p}_k\}; \{\mathbf{X}_i, C_i\}\right) \\ &= -\sum_{i=1}^{n} \ln \pi_{C_i} - \sum_{i=1}^{n} \sum_{m=1}^{M} [X_{i,m} \ln p_{C_i,m} + (1 - X_{i,m}) \ln(1 - p_{C_i,m})] \\ &= -\sum_{k=1}^{K} N_k \ln \pi_k - \sum_{k=1}^{K} \sum_{m=1}^{M} [N_{km} \ln p_{km} + (N - N_{km}) \ln(1 - p_{km})].\end{aligned}$$

Assuming $N_k > 0$ for all k, it can be shown using calculus that the optimum is reached at

$$\hat{\pi}_k = \frac{N_k}{N}, \quad \hat{p}_{km} = \frac{N_{km}}{N_k}$$

for all k, m. While maximum likehood estimation has desirable theoretical properties (https://en.wikipedia.org/wiki/Maximum_likelihood_estimation#Properties), it does suffer from overfitting (https://towardsdatascience.com/parameter-inference-maximum-aposteriori-estimate-49f3cd98267a). In this case, if for instance a particular word does not occur in any training document then the probability of observing a new document containing that word is 0 for any class. One approach to deal with this is Laplace smoothing (https://en.wikipedia.org/wiki/Additive_smoothing)

$$\bar{\pi}_k = \frac{N_k + \alpha}{N + K\alpha}, \quad \bar{p}_{km} = \frac{N_{km} + \beta}{N_k + 2\beta}$$

where $\alpha, \beta > 0$, which can be justified using a Bayesian perspective.

2 Twitter US Airline Sentiment dataset

We consider the task of sentiment analysis, which is a classification problem. We use a dataset from Crowdflower (https://data.world/crowdflower/airline-twitter-sentiment). The full datatset is available here (https://www.kaggle.com/crowdflower/twitter-airline-sentiment). Quoting Crowdflower (https://data.world/crowdflower/airline-twitter-sentiment):

> A sentiment analysis job about the problems of each major U.S. airline. Twitter data was scraped from February of 2015 and contributors were asked to first classify positive, negative, and neutral tweets, followed by categorizing negative reasons (such as "late flight" or "rude service").

We first load a cleaned-up version of the data and look at its summary.

In [1]:
```
# Julia version: 1.5.1
using CSV, DataFrames, TextAnalysis, Random, Plots, LaTeXStrings
using TextAnalysis: text
```

In [2]:
```
df = CSV.read("./twitter-sentiment.csv")
first(df,5)
```

Out[2]: 5 rows × 4 columns

	time	user	sentiment	text
	String	String	String	String
1	2/24/15 11:35	cairdin	neutral	@VirginAmerica What @dhepburn said.
2	2/24/15 11:15	jnardino	positive	@VirginAmerica plus you've added commercials to the experience... tacky.
3	2/24/15 11:15	yvonnalynn	neutral	@VirginAmerica I didn't today... Must mean I need to take another trip!
4	2/24/15 11:15	jnardino	negative	@VirginAmerica it's really aggressive to blast obnoxious "entertainment" in your guests' faces & they have little recourse
5	2/24/15 11:14	jnardino	negative	@VirginAmerica and it's a really big bad thing about it

In [3]:
```
n = nrow(df)
```

Out[3]:
```
14640
```

We use the package TextAnalysis.jl (https://github.com/JuliaText/TextAnalysis.jl) to extract and process text information in this dataset. The `Corpus` function creates a collection of text documents (one for each tweet) from a `DataFrame`. The function `text` shows the text itself.

In [4]:
```
crps = Corpus(StringDocument.(df[:,:text]))
text.(crps)
```

Out[4]:
```
14640-element Array{String,1}:
 "@VirginAmerica What @dhepburn said."
 "@VirginAmerica plus you've added commercials to the experience... tac
ky."
 "@VirginAmerica I didn't today... Must mean I need to take another tri
p!"
 "@VirginAmerica it's really aggressive to blast obnoxious \"entertainm
ent\" in your guests' faces &amp;amp; they have little recourse"
 "@VirginAmerica and it's a really big bad thing about it"
 "@VirginAmerica seriously would pay \$30 a flight for seats that did
n't have this playing.\nit's really the only bad thing about flying VA"
 "@VirginAmerica yes, nearly every time I fly VX this \x89\xdb\xcfear w
orm\x89\u6dd won\x89t go away :)"
 "@VirginAmerica Really missed a prime opportunity for Men Without Hats
parody, there. https://t.co/mWpG7grEZP"
 "@virginamerica Well, I didn't\x89\xdb_but NOW I DO! :-D"
 "@VirginAmerica it was amazing, and arrived an hour early. You're too
good to me."
 "@VirginAmerica did you know that suicide is the second leading cause
of death among teens 10-24"
 "@VirginAmerica I &lt;3 pretty graphics. so much better than minimal i
conography. :D"
 "@VirginAmerica This is such a great deal! Already thinking about my 2
nd trip to @Australia &amp;amp; I haven't even gone on my 1st trip yet! ;p"
 ⋮
 "Thank you. \x89\xdb\xcf@AmericanAir: @jlhalldc Customer Relations wil
l review your concerns and contact you back directly, John.\x89\u6dd"
 "@AmericanAir How do I change my flight if the phone system keeps tell
ing me that the representatives are busy?"
 "@AmericanAir Thanks! He is."
 "@AmericanAir thx for nothing on getting us out of the country and bac
k to US. Broken plane? Come on. Get another one."
 "\x89\xdb\xcf@AmericanAir: @TilleyMonsta George, that doesn't look goo
d. Please follow this link to start the refund process: http://t.co/4gr
39s91Dl\x89\u6dd_\xd9\xf7\xe2"
 "@AmericanAir my flight was Cancelled Flightled, leaving tomorrow morn
ing. Auto rebooked for a Tuesday night flight but need to arrive Monda
y."
 "@AmericanAir right on cue with the delays_\xd9\xd4\xce"
 "@AmericanAir thank you we got on a different flight to Chicago."
 "@AmericanAir leaving over 20 minutes Late Flight. No warnings or comm
unication until we were 15 minutes Late Flight. That's called shitty cu
stomer svc"
 "@AmericanAir Please bring American Airlines to #BlackBerry10"
 "@AmericanAir you have my money, you change my flight, and don't answe
r your phones! Any other suggestions so I can make my commitment??"
 "@AmericanAir we have 8 ppl so we need 2 know how many seats are on th
e next flight. Plz put us on standby for 4 people on the next flight?"
```

We first preprocess (https://juliatext.github.io/TextAnalysis.jl/latest/documents/#Preprocessing-Documents-1) the data. In particular, we lower-case all the words and remove punctuation. A more careful pre-procsseing would also include stemming, although we do not do this here. Regarding the latter, quoting Wikipedia (https://en.wikipedia.org/wiki/Stemming):

> In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. [...] A computer program or subroutine that stems word may be called a stemming program, stemming algorithm, or stemmer. [...] A stemmer for English operating on the stem cat should identify such strings as cats, catlike, and catty. A stemming algorithm might also reduce the words fishing, fished, and fisher to the stem fish. The stem need not be a word, for example the Porter algorithm reduces, argue, argued, argues, arguing, and argus to the stem argu.

Finally, `update_lexicon!` creates a lexicon from the dataset. Quoting TextAnalysis.jl (https://juliatext.github.io/TextAnalysis.jl/latest/corpus/#Corpus-Statistics-1):

> The lexicon of a corpus consists of all the terms that occur in any document in the corpus. The lexical frequency of a term tells us how often a term occurs across all of the documents. Often the most interesting words in a document are those words whose frequency within a document is higher than their frequency in the corpus as a whole.

The preprocessed text is shown below.

In [5]:
```
# preprocessing
function text_preproc!(crps)
    remove_corrupt_utf8!(crps)
    remove_case!(crps)
    prepare!(crps, strip_punctuation)
    #stem!(crps)
    update_lexicon!(crps)
end
```

Out[5]:
```
text_preproc! (generic function with 1 method)
```

In [6]:
```
text_preproc!(crps)
text.(crps)
```

Out[6]:
```
14640-element Array{String,1}:
 "virginamerica what dhepburn said"
 "virginamerica plus youve added commercials to the experience tacky"
 "virginamerica i didnt today must mean i need to take another trip"
 "virginamerica its really aggressive to blast obnoxious entertainment
in your guests faces amp they have little recourse"
 "virginamerica and its a really big bad thing about it"
 "virginamerica seriously would pay 30 a flight for seats that didnt ha
ve this playing\nits really the only bad thing about flying va"
 "virginamerica yes nearly every time i fly vx this    ear worm \u6dd w
on t̤ go away "
 "virginamerica really missed a prime opportunity for men without hats
parody there httpstcomwpg7grezp"
 "virginamerica well i didnt  but now i do d"
 "virginamerica it was amazing and arrived an hour early youre too good
to me"
 "virginamerica did you know that suicide is the second leading cause o
f death among teens 1024"
 "virginamerica i lt3 pretty graphics so much better than minimal icono
graphy d"
 "virginamerica this is such a great deal already thinking about my 2nd
trip to australia amp i havent even gone on my 1st trip yet p"
 ⋮
 "thank you    americanair jlhalldc customer relations will review your
concerns and contact you back directly john \u6dd"
 "americanair how do i change my flight if the phone system keeps telli
ng me that the representatives are busy"
 "americanair thanks he is"
 "americanair thx for nothing on getting us out of the country and back
to us broken plane come on get another one"
 "   americanair tilleymonsta george that doesnt look good please follo
w this link to start the refund process httptco4gr39s91dl \u6dd   "
 "americanair my flight was cancelled flightled leaving tomorrow mornin
g auto rebooked for a tuesday night flight but need to arrive monday"
 "americanair right on cue with the delays   "
 "americanair thank you we got on a different flight to chicago"
 "americanair leaving over 20 minutes late flight no warnings or commun
ication until we were 15 minutes late flight thats called shitty custom
er svc"
 "americanair please bring american airlines to blackberry10"
 "americanair you have my money you change my flight and dont answer yo
ur phones any other suggestions so i can make my commitment"
 "americanair we have 8 ppl so we need 2 know how many seats are on the
next flight plz put us on standby for 4 people on the next flight"
```

Next, we convert our dataset into a matrix by creating a document-term matrix using the `DocumentTermMatrix` function. Quoting Wikipedia (https://en.wikipedia.org/wiki/Document-term_matrix):

> A document-term matrix or term-document matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a document-term matrix, rows correspond to documents in the collection and columns correspond to terms.

In [7]:
```
m = DocumentTermMatrix(crps)
t = m.terms
```

Out[7]:
```
16671-element Array{String,1}:
 "0"
 "00"
 "001"
 "0011"
 "0016"
 "006"
 "009"
 "01"
 "015"
 "0162389030167"
 "0162424965446"
 "0162431184663"
 "0167560070877"
 ⋮
 "\u6ddunfortunately"
 "\u6ddw"
 ""
 "all"
 "d"
 "l"
 "ll"
 "m"
 "re"
 "s"
 "t"
 "ve"
```

Because of our use of the multivariate Bernoulli naive Bayes model, it will be more convenient to work with a variant of the document-term matrix where each word is either present or absent. Note that, in the context of tweet data which are very short documents with likely little word repetition, there is probably not much difference.

```
In [8]: dt = m.dtm .> 0
```

```
Out[8]: 14640×16671 SparseArrays.SparseMatrixCSC{Bool,Int64} with 246999 stored
        entries:
          [36   ,     1]  =  1
          [41   ,     1]  =  1
          [74   ,     1]  =  1
          [217  ,     1]  =  1
          [223  ,     1]  =  1
          [258  ,     1]  =  1
          [338  ,     1]  =  1
          [353  ,     1]  =  1
          [403  ,     1]  =  1
          [436  ,     1]  =  1
          [526  ,     1]  =  1
          [646  ,     1]  =  1
          ⋮
          [2701 , 16671]  =  1
          [3445 , 16671]  =  1
          [3460 , 16671]  =  1
          [8042 , 16671]  =  1
          [10200, 16671]  =  1
          [10872, 16671]  =  1
          [11063, 16671]  =  1
          [11646, 16671]  =  1
          [11871, 16671]  =  1
          [12293, 16671]  =  1
          [13752, 16671]  =  1
          [14049, 16671]  =  1
          [14137, 16671]  =  1
```

We also extract the labels (`neutral` , `postive` , `negative`) from the dataset.

In [9]:
```
labels = df[:,:sentiment]
```

Out[9]:
```
14640-element PooledArrays.PooledArray{String,UInt32,1,Array{UInt32,
1}}:
 "neutral"
 "positive"
 "neutral"
 "negative"
 "negative"
 "negative"
 "positive"
 "neutral"
 "positive"
 "positive"
 "neutral"
 "positive"
 "positive"
 ⋮
 "positive"
 "negative"
 "positive"
 "negative"
 "neutral"
 "negative"
 "negative"
 "positive"
 "negative"
 "neutral"
 "negative"
 "neutral"
```

We split the data into a training set and a test set.

In [10]:
```
function split_data(m, n, ϱ)

    τ = randperm(n) # permutation of the rows
    train_size = Int(floor(ϱ*n)) # ϱ should be between 0 and 1
    train_set = τ[1:train_size]
    test_set = τ[train_size+1:end]

    return train_set, test_set
end
```

Out[10]:
```
split_data (generic function with 1 method)
```

In [11]:
```
train_set, test_set = split_data(m , n, 0.9)
dt_train = dt[train_set,:]
labels_train = labels[train_set]
train_size = length(labels_train)
dt_test = dt[test_set,:]
labels_test = labels[test_set]
test_size = length(labels_test);
```

We implement the Naive Bayes method. We use Laplace smoothing (https://en.wikipedia.org/wiki/Additive_smoothing) to avoid overfitting issues.

In [12]:
```
function mmids_nb_fit(c, labels, dt, size; α=1, β=1)

    # rows corresponding to class c
    c_rows = findall(labels.==c)

    # class prior
    σ_c = (length(c_rows)+α)/(size+3α)

    # term mle
    N_j_c = sum(dt[c_rows,:].>=1,dims=1)
    N_c = sum(N_j_c)
    θ_c = (N_j_c.+β)./(N_c+2β)

    return σ_c, θ_c
end
```

Out[12]:
```
mmids_nb_fit (generic function with 1 method)
```

We are ready to train on the dataset.

In [13]:
```
# training
label_set = ["positive", "negative", "neutral"]
σ_c = zeros(3)
θ_c = zeros(3, length(t))
for i=1:length(label_set)
    σ_c[i], θ_c[i,:] = mmids_nb_fit(
        label_set[i], labels_train, dt_train, train_size)
end
```

In [14]:
```
σ_c
```

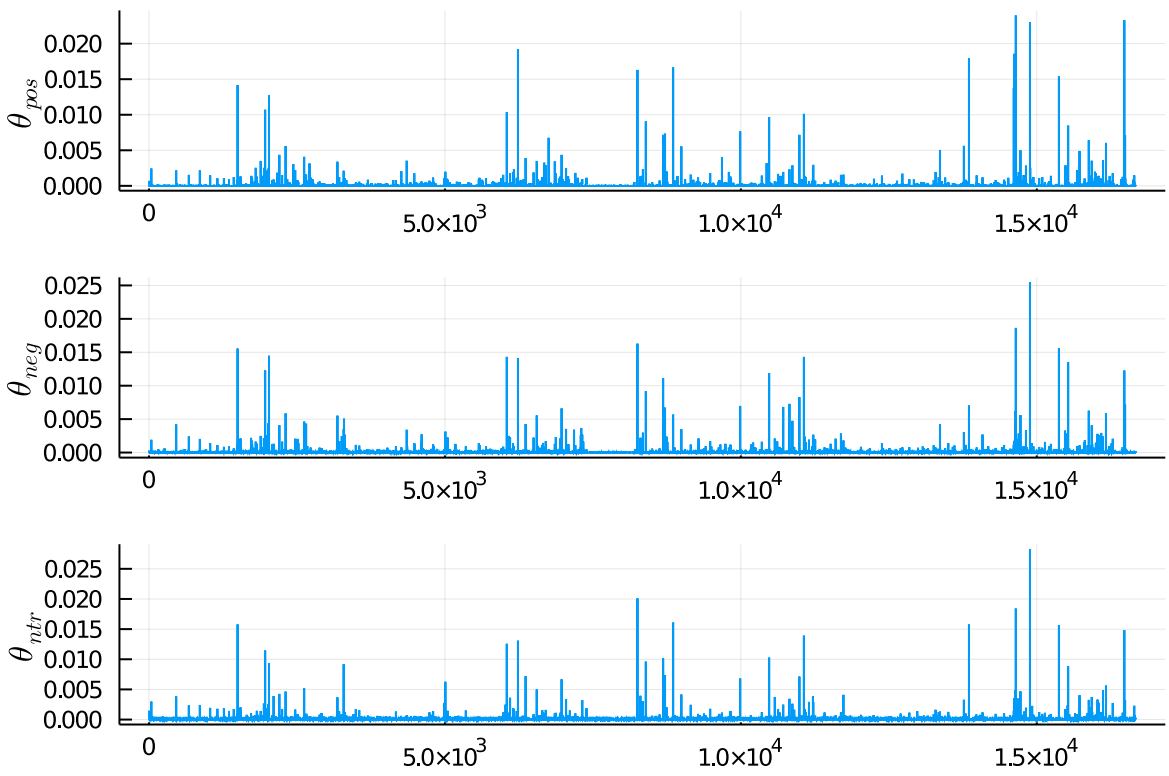
Out[14]:
```
3-element Array{Float64,1}:
 0.16169663859169892
 0.6275134683966918
 0.21078989301160939
```

In [15]:
```
θ_c
```

Out[15]:
```
3×16671 Array{Float64,2}:
 0.000720783  6.8646e-5   3.4323e-5   3.4323e-5   …  0.000137292  0.000
137292
 0.000336648  1.2948e-5   6.474e-6    6.474e-6       0.000207168  5.179
2e-5
 0.00150472   2.55037e-5  5.10074e-5  5.10074e-5     0.000153022  5.100
74e-5
```

In [16]:
```
plot([θ_c[1,:], θ_c[2,:], θ_c[3,:]], layout=(3,1), legend=false,
    ylabel=[L"$\theta_{pos}$" L"$\theta_{neg}$" L"$\theta_{ntr}$"])
```

Out[16]:



We compute a prediction on each test tweet using the `findmax` function.

In [17]:
```
?findmax
```

```
search: findmax findmax! findmin findmin!
```

Out[17]:

```
findmax(itr) -> (x, index)
```

Return the maximum element of the collection `itr` and its index. If there are multiple maximal elements, then the first one will be returned. If any data element is `NaN`, this element is returned. The result is in line with `max`.

The collection must not be empty.

Examples

```
jldoctest
julia> findmax([8,0.1,-9,pi])
(8.0, 1)

julia> findmax([1,7,7,6])
(7, 2)

julia> findmax([1,7,7,NaN])
(NaN, 4)
```

```
findmax(A; dims) -> (maxval, index)
```

For an array input, returns the value and index of the maximum over the given dimensions. `NaN` is treated as greater than all other values.

Examples

```
jldoctest
julia> A = [1.0 2; 3 4]
2×2 Array{Float64,2}:
 1.0  2.0
 3.0  4.0

julia> findmax(A, dims=1)
([3.0 4.0], CartesianIndex{2}[CartesianIndex(2, 1) CartesianIndex(2,
 2)])

julia> findmax(A, dims=2)
([2.0; 4.0], CartesianIndex{2}[CartesianIndex(1, 2); CartesianIndex(2,
 2)])
```

In [18]:
```
# testing
score_c = log.(θ_c) * dt_test'
score_c .+= log.(1 .- θ_c) * (1 .- dt_test')
score_c .+= log.(σ_c)
(maxscr, argmax) = findmax(score_c, dims=1);
```

For example, for the 5th test tweet:

In [19]:
```
score_c[:,5]
```

Out[19]:
```
3-element Array{Float64,1}:
 -75.75143663488974
 -75.18163974055193
 -74.01534684364829
```

In [20]:
```
maxscr[5]
```

Out[20]:
```
-74.01534684364829
```

In [21]:
```
argmax[5]
```

Out[21]:
```
CartesianIndex(3, 5)
```

The following computes the overall accuracy over the test data.

In [22]:
```
# accuracy
acc = 0
for i in 1:length(test_set)
    if label_set[argmax[i][1]]==labels_test[i]
        acc += 1
    end
end
acc/(length(labels_test))
```

Out[22]:
```
0.7137978142076503
```

To get a better understanding of the differences uncovered by Naive Bayes between the different labels, we identify words that are particularly common in one label, but on the other. Recall that label `1` corresponds to `positive` while label `2` corresponds to `negative`.

In [23]:
```
t[findall((θ_c[1,:].>0.002).&(θ_c[2,:].<0.002))]
```

Out[23]:
```
18-element Array{String,1}:
 "1"
 "airline"
 "amazing"
 "awesome"
 "best"
 "crew"
 "flying"
 "good"
 "got"
 "great"
 "guys"
 "love"
 "much"
 "thank"
 "thanks"
 "today"
 "very"
 "virginamerica"
```

One notices that many (stemmed) positive words do appear in this list: `awesome`, `best`, `great`, `love`, `thanks`.

In [24]:
```
t[findall((θ_c[2,:].>0.002).&(θ_c[1,:].<0.002))]
```

Out[24]:
```
35-element Array{String,1}:
 "3"
 "about"
 "after"
 "am"
 "bag"
 "been"
 "call"
 "cancelled"
 "cant"
 "delayed"
 "do"
 "dont"
 "flightled"
 ⋮
 "need"
 "no"
 "one"
 "phone"
 "plane"
 "still"
 "there"
 "they"
 "what"
 "when"
 "why"
 "would"
```

This time, we notice: `bag` , `cancelled` , `cant` , `delayed` , `dont` , `no` , `phone` .