Text Mining Applications

Motivation

As already discussed:

- Once we have a way of measuring dissimilarities of texts we can search for similar texts, and perform all statistical learning methods which work on dissimilarities: scaling, clustering, ...
- Once we have fixed length numeric representations/embeddings of texts we can treat them as "usual" numeric data, and perform all statistical learning methods which work with numeric data:
  - Unsupervised learning: based on dissimilarities as above, finding good probabilistic models, ...
  - Supervised learning with the text representations as predictors or as responses

Unsupervised Learning

Visualization

Word clouds

You like word clouds?

You think they are informative?

Basic idea: size words according to their frequencies and arrange “suitably”.

For simply arrangements, can easily see the most important words.

For advanced arrangements, can also see some semantic relatedness (needs semantic embedding, of course).

Can be nice for comparison: e.g., a Computing course project compared what QFin students expected vs realized in the program.

Clustering

Basics

The most basic task in clustering is the following:

Given n objects $O_1, \ldots, O_n$, divide them into groups of similar objects.

Can do this one single partition (partitional clustering) or a hierarchy thereof (hierarchical clustering).

Needs the (dis)similarities between objects.

Dissimilarities: $d_{ij} \geq 0$, symmetric in i and j, $d_{ii} = 0$. Clearly, different objects should have positive dissimilarity.

Note, not a metric (no triangle inequality in general).

Similarities: $0 \leq s_{ij} \leq 1$, symmetric in i and j, $s_{ii} = 1$.

Clearly, going from similarities to dissimilarities is easy: $d_{ij} = 1 - s_{ij}$.

Basics

If we only have the matrix of pairwise (dis)similarities:

- Can always perform hierarchical clustering methods which approximate dissimilarties by ultrametrics
- Can partitition using k-medoids. Write $c(i)$ for the group/cluster of object i. The medoid of group g is the object in the group which minimizes (e.g.) the total dissimilarity to all objects in the group:

$$\mu_g = \sum_{i:c(i)=g} d_{ij} \to \min_j \quad \text{s.t.} \quad c(j) = g.$$

The optimal partition minimizes

$$\sum_g \sum_{i:c(i)=g} d_{i,\mu_g}.$$

In R, `cluster::pam()`.

Basics / 2

- Perform scaling: find suitable (typically low-dimensional) Euclidean embeddings $x_1, \ldots, x_n$ such that

  $$d_{ij} \approx \|x_i - x_j\|.$$

  If 2-dimensional: can plot.
  In general: over to the case of having fixed length numeric representations/embeddings ...
  A bit silly if we computed the dissimilarities from a numeric representation. Well, modulo dimension reduction ...

Prototype-based partitioning

If we have numeric representations (“features”) we can do more.

Write d for a dissimilarity function in the feature space.

Instead of looking for the **object** which minimizes the total dissimilarity to the objects in the group, we can look for the feature vector (not constrained to be that of an object) minimizing the total dissimilarity:

$$p_g = \arg\min \left\{ \sum_{i:c(i)=g} d(x_i, p) \right\}.$$

This is the **prototype** of group g.

The optimal partition minimizes

$$\sum_g \sum_{i:c(i)=g} d(x_i, p_g).$$

Prototype-based partitioning

Write $M = [\mu_{ig}]$ for the indicator matrix of the group assignments:

$$\mu_{ig} = \begin{cases} 1 & c(i) = g \\ 0 & \text{otherwise.} \end{cases}$$

Then the criterion function can be written as

$$C(M, p_1, \ldots, p_k) = \sum_{i,g} \mu_{ig} d(x_i, p_g).$$

Optimizing this is a hard combinatorial optimization problem: usually solved by using suitable **heuristics** and not by exact solvers.

If d is Euclidean dissimilarity (i.e., squared Euclidean distance): k-means clustering.

In R, `stats::kmeans()`.

Prototype-based partitioning

In the above, M is a binary matrix. We could relax this restriction to M being a **stochastic matrix**, i.e.,

$$\mu_{ig} \geq 0 \quad \forall i, g, \qquad \sum_g \mu_{ig} = 1 \quad \forall i.$$

In the context of partitional clustering, M is called the **membership matrix** of the corresponding **fuzzy** (or **soft**) partition.

Restricting M to binary gives a **crisp** (or **hard**) partition.

Fuzzy is different from probability, but you could think of μ_{ig} as the probability of object i getting classified into group g.

Prototype-based partitioning

Optimizing

$$C(M, p_1, \ldots, p_k) = \sum_{i,g} \mu_{ig}^m d(x_i, p_g)$$

does **fuzzy prototype-based partitioning**, where the **fuzzification parameter** $m \geq 1$ controls the softness.

The bigger m the softer, $m = 1$ gives hard partitions.

Why? One can easily show (Lagrange) that for fixed prototypes, the optimal memberships are

$$\mu_{ig} = \frac{1/d(x_i, p_g)^{1/(m-1)}}{\sum_h 1/d(x_i, p_h)^{1/(m-1)}}$$

For $m \to 1+$ this clearly gives hard partitions.

Note: we can always go from soft to hard via classifying into the group with the highest membership.

In addition, we can look at the "fuzziness" of the classification (e.g., the **margin**).

Spherical k-means clustering

Ok, so we compute embeddings and then do standard k-means, or to be more fancy do fuzzy c-means (sic)?

Well, depends on which embeddings we work with.

For the basic vector space model (representation by term frequencies), we already learned the using cosine similarity works very well. In this case,

$$d(x, p) = 1 - \cos(x, p) = 1 - \frac{\langle x, p \rangle}{\|x\| \|p\|}.$$

Optimizing

$$C(M, p_1, \ldots, p_k) = \sum_{i,g} \mu_{ig}^m d(x_i, p_g)$$

is (soft or hard) **spherical k-means clustering**.

In R, `skmeans::skmeans()`. Remember, heuristics to solve an NP-hard problem.

Spherical k-means clustering

Note that

$$\sum_i \mu_{ij}^m d(x_i, p_g) = \sum_i \mu_{ij}^m \left(1 - \frac{\langle x_i, p_g \rangle}{\|x_i\| \|p_g\|}\right) = \sum_i \mu_{ij}^m - \left\langle \sum_i \mu_{ij}^m \frac{x_i}{\|x_i\|}, \frac{p_g}{\|p_g\|} \right\rangle$$

which by Cauchy-Schwartz is minimized if

$$p_g \propto \sum_i \mu_{ij}^m \frac{x_i}{\|x_i\|}.$$

In particular, if $m = 1$,

$$p_g \propto \sum_{i:c(i)=g} \frac{x_i}{\|x_i\|}$$

i.e., the optimal prototype for group g has the same direction as the sum of the normalized feature vectors of the objects in group g.

Hence, **spherical** k-means.

Oz Books data

Who did not read “The Wonderful Wizard of Oz”?

Frank L. Baum himself authored 14 Oz Books.

After his death, more were written: there are 40 “official Oz books”, including 19 books by Ruth Plumly Thompson.

“The Royal Book of Oz” (Oz book 15) is the first published by Thompson in 1921, Baum died in 1919.

Authorship has long been disputed: today, commonly attributed to Thompson based on stylometric analysis.

We can come to similar conclusions using spherical k-means clustering.

Oz Books data

There are 21 official Oz books in the public domain: all 14 by Baum, and 7 by Thompson.

The ones by Thompson are her 2 first (including Oz book 15) and her 7 last.

Package **tm.corpus.Oz.Books** from datacube conveniently provides these as a **tm** `Corpus` object.

To install:

```
install.packages("tm.corpus.Oz.Books",
                 repos = "https://datacube.wu.ac.at/",
                 type = "source")
```

Oz Books data

```
R> data("Oz_Books", package = "tm.corpus.Oz.Books")
R> x <- tm::DocumentTermMatrix(Oz_Books,
+                              control =
+                                  list(removePunctuation = TRUE,
+                                       stopwords = TRUE))
R> x

<<DocumentTermMatrix (documents: 21, terms: 20088)>>
Non-/sparse entries: 90242/331606
Sparsity           : 79%
Maximal term length: 103
Weighting          : term frequency (tf)
```

Oz Books data

We now perform (not very) fuzzy spherical k-means clustering with $k = 4$ groups.

(This takes quite some time, hence the `verbose = TRUE` for some entertainment . . .)

```
R> set.seed(1234)
R> party <- skmeans::skmeans(tm::weightBin(x), k = 4, m = 1.2,
+                            control = list(nruns = 20, verbose = TRUE))

R> party

A soft spherical k-means partition (degree m = 1.200000) of 21 objects
into 4 classes.
Class sizes of closest hard partition: 4, 8, 3, 6.
Call: skmeans::skmeans(x = tm::weightBin(x), k = 4, m = 1.2, control = list(nruns =
    verbose = TRUE))
```

Oz Books data

We can then compare the class ids (groups) of the partition with the “true” authors:

```
R> ids <- abbreviate(unlist(NLP::meta(Oz_Books, "author")))
R> table(party$cluster, ids)
```

```
   ids
    L.FB RtPT
  1    0    4
  2    8    0
  3    0    3
  4    6    0
```

So we get two all-Baum and two all-Thompson groups.

Oz Books data

We can also look at which books these groups contain:

```
R> split(party$cluster, ids)

$L.FB
Oz_Book_01.txt Oz_Book_02.txt Oz_Book_03.txt Oz_Book_04.txt Oz_Book_05.txt
             4              4              4              4              4
Oz_Book_06.txt Oz_Book_07.txt Oz_Book_08.txt Oz_Book_09.txt Oz_Book_10.txt
             4              2              2              2              2
Oz_Book_11.txt Oz_Book_12.txt Oz_Book_13.txt Oz_Book_14.txt
             2              2              2              2


$RtPT
Oz_Book_15.txt Oz_Book_16.txt Oz_Book_29.txt Oz_Book_30.txt Oz_Book_31.txt
             1              1              3              3              1
Oz_Book_32.txt Oz_Book_33.txt
             3              1
```

So we get one for the early and one for the late Baum (in the analysis in my JSS paper, also one for early and late Thompson, but that was done more than 10 years ago).

Oz Books data

Remember that we actually performed fuzzy clustering? So we can also look the memberships:

```
R> tail(party$membership, 7)
```

```
                         1          2          3          4
Oz_Book_15.txt 0.85408564 0.05070542 0.04473283 0.05047611
Oz_Book_16.txt 0.87647187 0.04012873 0.04310834 0.04029106
Oz_Book_29.txt 0.03975801 0.02327580 0.91448874 0.02247745
Oz_Book_30.txt 0.03305257 0.02068932 0.92719632 0.01906179
Oz_Book_31.txt 0.85994174 0.04004933 0.06062594 0.03938300
Oz_Book_32.txt 0.02765307 0.01598456 0.94064873 0.01571364
Oz_Book_33.txt 0.84050489 0.04693279 0.06549725 0.04706507
```

So the “confidence” for Oz book 15 to be written by Thompson is quite high!

Oz Books data

As generally is the case for Thompson books, and not so much for Baum books:

```
R> apply(party$membership, 1, max)
```

```
Oz_Book_01.txt Oz_Book_02.txt Oz_Book_03.txt Oz_Book_04.txt Oz_Book_05.txt
     0.7042562      0.4581458      0.7641319      0.7488113      0.7776512
Oz_Book_06.txt Oz_Book_07.txt Oz_Book_08.txt Oz_Book_09.txt Oz_Book_10.txt
     0.7032570      0.5190532      0.6408778      0.6951004      0.6766898
Oz_Book_11.txt Oz_Book_12.txt Oz_Book_13.txt Oz_Book_14.txt Oz_Book_15.txt
     0.6966509      0.7244291      0.6269550      0.7433769      0.8540856
Oz_Book_16.txt Oz_Book_29.txt Oz_Book_30.txt Oz_Book_31.txt Oz_Book_32.txt
     0.8764719      0.9144887      0.9271963      0.8599417      0.9406487
Oz_Book_33.txt
     0.8405049
```

Oz Books data

```
R> head(party$membership, 14)
```

	1	2	3	4
Oz_Book_01.txt	0.06509895	0.1878379	0.04280689	0.7042562
Oz_Book_02.txt	0.08961589	0.3931984	0.05903995	0.4581458
Oz_Book_03.txt	0.04706997	0.1594915	0.02930661	0.7641319
Oz_Book_04.txt	0.04423489	0.1780726	0.02888123	0.7488113
Oz_Book_05.txt	0.04560909	0.1474713	0.02926845	0.7776512
Oz_Book_06.txt	0.04999027	0.2124196	0.03433307	0.7032570
Oz_Book_07.txt	0.06616577	0.5190532	0.04368568	0.3710953
Oz_Book_08.txt	0.06222130	0.6408778	0.04078357	0.2561173
Oz_Book_09.txt	0.05836656	0.6951004	0.03957599	0.2069570
Oz_Book_10.txt	0.06057876	0.6766898	0.04660348	0.2161279
Oz_Book_11.txt	0.05092339	0.6966509	0.03231938	0.2201064
Oz_Book_12.txt	0.04305916	0.7244291	0.02716687	0.2053449
Oz_Book_13.txt	0.06380565	0.6269550	0.04171619	0.2675232
Oz_Book_14.txt	0.04136030	0.7433769	0.03004137	0.1852214

Note the early vs late Baum (2 and 4), and how these memberships change with time!

Model-based clustering

The results of fuzzy/soft have a kind of “probabilistic” (but remember, fuzzy!) interpretation.

A fully probabilistic one can be obtained as follows.

Suppose that all feature vectors x_i and prototypes p_g are normalized to length one. Then the criterion function is

$$\sum_{i,g} \mu_{ig}(1 - x_i' p_g) = 1 - \sum_{i,g} \mu_{ig} p_g' x_i.$$

Minimizing is equivalent to maximizing

$$\exp\left(\sum_{i,g} \mu_{ig} p_g' x_i\right) = \prod_g \prod_{i:c(i)=g} \exp(p_g' x_i).$$

Model-based clustering

This looks like the (conditional on the class memberships) likelihood function of a distribution on the unit sphere with density propto $\exp(p'x)$.

This is the **von Mises-Fisher distribution** on the unit sphere.

It actually works for arbitrary p (not necesarily normalized): the more the length, the more the concentration about the mode.

So we can perform model-based clustering of texts using mixtures of von-Mises Fisher distributions.

In R: package **movMF**.

Generalizes spherical k-means the same as mixture of normals generalizes standard k-means.

Model-based clustering

Cool in principle but:

- Computationally, computing the normalizing constants in the density is quite a challenge. See my papers on this.
- Conceptually, the normalized term frequencies live on the positive orthant and not the whole unit sphere.

Stay tuned . . .

Topic modeling

Topic modeling

Generally speaking, topic models are probabilistic models which allow us to understand text corpora in terms of the underlying “topics”.

Technically speaking, generative models for texts based on suitable mixtures of topics.

Here, we describe the most common topic model currently in use: **Latent Dirichlet Allocation** (LDA).

Before we begin . . .

Dirichlet distribution

The Dirichlet distribution is supported on the standard simplex

$$\Delta^{k-1} = \left\{ x \in \mathbb{R}^k : x_i \geq 0 \text{ for all } i, \sum_i x_i = 1 \right\}.$$

Think of Δ^{k-1} as the set of all probability vectors of length k.

It has density

$$f(x_1, \ldots, x_k | \alpha_1, \ldots, \alpha_k) = \frac{\Gamma(\alpha_1) \cdots \Gamma(\alpha_k)}{\Gamma(\alpha_1 + \cdots + \alpha_k)} x_1^{\alpha_1 - 1} \cdots x_k^{\alpha_k - 1}.$$

Looks familiar? Right: it generalizes the Beta distribution.

In fact, if $(X_1, \ldots, X_k) \sim \mathrm{Dirichlet}(\alpha_1, \ldots, \alpha_k)$, then marginally $X_i \sim \mathrm{Beta}(\alpha_i, \sum_{j \neq i} \alpha_j)$.

Dirichlet distribution

In what follows, we will use symmetric Dirichlet distributions, where $\alpha_1 = \cdots = \alpha_k = \alpha$.

One then simply writes $(X_1, \ldots, X_k) \sim \mathrm{Dirichlet}(\alpha)$, where α is the **concentration** parameter.

For $\alpha = 1$ we get the uniform distribution.

Values of $\alpha < 1$ give increasingly sparse distributions (most values close to zero) the closer α gets to zero.

Values of $\alpha > 1$ give increasingly dense distributions (most values similar) the closer α gets to ∞.

Can be seen from the fact that with Y_i independent and distributed as $\mathrm{Gamma}(\alpha_i, 1)$,

$$\left(\frac{Y_1}{\sum_i Y_i}, \ldots, \frac{Y_k}{\sum_i Y_i} \right) \sim \mathrm{Dirichlet}(\alpha_1, \ldots, \alpha_k).$$

We will be interested in the case where we get sparse distributions.

Latent Dirichlet Allocation

Suppose you have a vocabulary/dictionary V of L words/terms

$$v_1, \ldots, v_L$$

and a corpus of text documents tokenized into words from the vocabulary (the other tokens get dropped, of course):

$$D_i : (w_{i,1}, \ldots, w_{i,n_i}).$$

LDA is a simple generative model for the word sequences via a mixture of K underlying topics.

Each topic k has a word distribution

$$\beta_k \sim \mathrm{Dirichlet}(\delta)$$

where typically $\delta < 1$ to get sparse word distributions.

Latent Dirichlet Allocation

Each document i has a topic distribution

$$\theta_i \sim \mathrm{Dirichlet}(\alpha)$$

where usually $\alpha < 1$ to get sparse topic distribution.

The word sequences D_i are generated (independently) by doing the following:

For each word $w_{i,j}$ (again independently, bag of words assumption):

- Sample a topic $z_{i,j} \sim \mathrm{Categorical}(\theta_i)$
- Sample a word $w_{i,j} \sim \mathrm{Categorical}(\beta_{z_{i,j}})$

I.e., if the latent topic indicators $z_{i,j}$ were known, the probability of the word sequence D_i is

$$\prod_j p(w_{i,j}, z_{i,j}) = \prod_j p(w_{i,j}|z_{i,j})p(z_{i,j}) = [\beta_{z_{i,j}}]_{w_{i,j}}[\theta_i]_{z_{i,j}}$$

(Of course, one would usually put the θ and β into suitable matrices.)

Latent Dirichlet Allocation

(This uses the notation from the R **topicmodels** package and corresponding JSS paper.)

The likelihood for the parameters of interest is obtained by suitably integrating out the above.

Parameters of interest usually are α and the topic distributions β_k (and not the underlying δ).

The parameters can then be estimated using Variational Expectation-Maximization (VEM, default in package **topicmodels**) or collapsed Gibbs sampling.

Before we do an illustrative example, discuss: does this seem like a reasonable model?

(The basic Correlated Topic Model allows for correlated topic distributions θ. And there are many many extensions . . .)

In R: `topicmodels::LDA()`.

Financial Phrase Bank data: LDA

Let us see whether we can use LDA to find (some) interpretable topics in the Financial Phrase Bank corpus.

This time we drop the stopwords right away:

```
R> load("fpb.rda")
R> x <- tm::Corpus(tm::VectorSource(fpb$sentence))
R> m <- tm::DocumentTermMatrix(x,
+                              control = list(removePunctuation = TRUE,
+                                             removeNumbers = TRUE,
+                                             stopwords = TRUE))
R> m
```

```
<<DocumentTermMatrix (documents: 4846, terms: 9212)>>
Non-/sparse entries: 54284/44587068
Sparsity           : 100%
Maximal term length: 25
Weighting          : term frequency (tf)
```

Financial Phrase Bank data: LDA

We should reduce to the “relevant” words which occur often enough. Overall:

```
R> tf1 <- slam::col_sums(m)
R> summary(tf1)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
   1.000    1.000    1.000    6.255    4.000 1310.000
```

After tf-idf weighting:

```
R> tf2 <- slam::col_sums(tm::weightTfIdf(m))
R> summary(tf2)
```

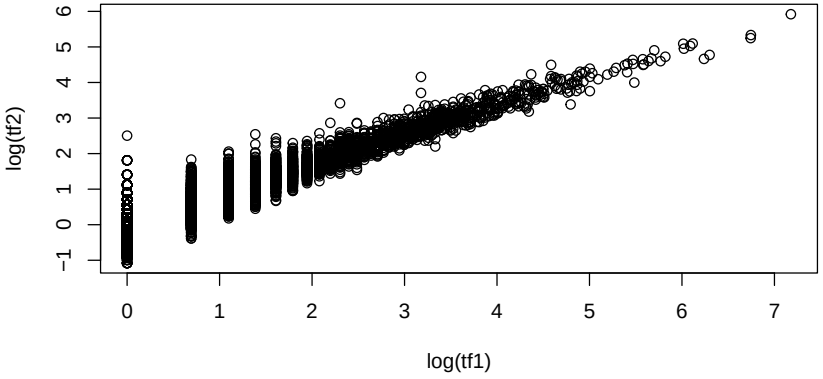
```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
  0.3401   0.8162   1.4276   3.9473   3.2538 372.3894
```

Filtering according to low tf2 values often works well.

Financial Phrase Bank data: LDA

But here it does not change much:

```
R> plot(log(tf1), log(tf2))
```



Financial Phrase Bank data: LDA

So maybe drop terms which only occur at least 4 times and have a high enough total tf-idf value:

```
R> m <- m[, (tf1 > 3) & (tf1 < 100) & (tf2 > 2)]
```

And drop empty documents (well, that's what you get when you filter out terms):

```
R> m <- m[slam::row_sums(m) > 0, ]
```

This leaves:

```
R> dim(m)
[1] 4795 2332
```

Financial Phrase Bank data: LDA

Now do LDA with $k = 10$ topics. (Results do not really get better with $k = 50$ or $k = 100$, but the code runs much longer.)

```
R> y <- topicmodels::LDA(m, k = 10)
R> y
```

```
A LDA_VEM topic model with 10 topics.
```

That's not much.

And in fact, this is not quite "as usual":

```
R> class(y)
```

```
[1] "LDA_VEM"
attr(,"package")
[1] "topicmodels"
```

```
R> typeof(y)
```

```
[1] "S4"
```

Financial Phrase Bank data: LDA

So this in an S4 object which has **slots** and not a list which has elements:

```
R> names(y)

NULL

R> slotNames(y)

 [1] "alpha"           "call"            "Dim"             "control"
 [5] "k"               "terms"           "documents"       "beta"
 [9] "gamma"           "wordassignments" "loglikelihood"   "iter"
[13] "logLiks"         "n"
```

We can access the slots using `@` (instead of `$`, don't ask) or `slot()`:

```
R> y@alpha

[1] 20.36809

R> slot(y, "alpha")

[1] 20.36809
```

Financial Phrase Bank data: LDA

Wait a minute. Is this the estimated α parameter of the topic distribution? Shouldn't this be (much) less than one so that we get sparse topic distributions?

Yes, you are right. So clearly LDA did not really work on the data set.

We can see this by looking at the topic distributions.

We can get both the topic and the term distributions via `topicmodels::posterior()`:

```
R> p <- topicmodels::posterior(y)
```

This is now a list again:

```
R> class(p)

[1] "list"

R> names(p)

[1] "terms"  "topics"
```

Financial Phrase Bank data: LDA

Both the `terms` and `topics` element are matrices:

```
R> dim(p$terms)
```

```
[1]   10 2332
```

So this has the term distributions β_k for the topics in its rows.

```
R> dim(p$topics)
```

```
[1] 4795   10
```

So this has the topic distributions θ_i of the documents in its rows.

Financial Phrase Bank data: LDA

Inspecting the first few topic distributions:

```
R> round(head(p$topics), 3)
```

```
      1     2     3     4     5     6     7     8     9    10
1 0.099 0.100 0.103 0.100 0.099 0.100 0.100 0.099 0.100 0.101
2 0.103 0.099 0.103 0.099 0.099 0.098 0.100 0.100 0.098 0.100
3 0.102 0.099 0.099 0.099 0.099 0.100 0.099 0.099 0.102 0.103
4 0.100 0.102 0.102 0.099 0.098 0.100 0.100 0.100 0.100 0.099
5 0.100 0.101 0.099 0.101 0.100 0.100 0.101 0.098 0.100 0.100
6 0.103 0.098 0.100 0.099 0.101 0.101 0.099 0.101 0.098 0.099
```

Clearly, this is totally useless (as we already knew when seeing the big α.

Financial Phrase Bank data: LDA

"Usually" (when things work better), one tries to interpret the topics found based on their most frequent terms. Here, this gives

```
R> topicmodels::terms(y, 5)
```

```
     Topic 1  Topic 2   Topic 3       Topic 4      Topic 5     Topic 6
[1,] "real"   "data"    "development" "board"      "future"    "result"
[2,] "estate" "network" "building"    "two"        "employees" "pct"
[3,] "months" "maker"   "markets"     "management" "value"     "board"
[4,] "third"  "yit"     "russia"      "general"    "maker"     "ltd"
[5,] "signed" "system"  "transaction" "chairman"   "situation" "markets"
     Topic 7     Topic 8  Topic 9  Topic 10
[1,] "one"       "pct"    "russia" "order"
[2,] "euros"     "order"  "eurm"   "omx"
[3,] "estimated" "omx"    "news"   "plc"
[4,] "billion"   "world"  "part"   "development"
[5,] "announced" "nordic" "stock"  "result"
```

Here, these are not interpretable. And we see that we really need to invest more effort in the preprocessing/filtering (e.g., we kicked out EUR but not USD, . . .).

Summary

Maybe next year (when we know how to download financial reports from EDGAR) we will have a case study where LDA works better.

If LDA worked, what could we do with it?

- E.g., classify documents according to their primary (ideally nicely interpretable) topic.
  I.e., use the topic distributions like the memberships for soft clustering. Here, of course, we have model-based clustering (using a probabilistic model).
- Use the topic distributions as numeric features for the text.
  Hmm, but isn't this a bit backwards?
  Well, if things work we get **semantic** features!

Term scoring

Motivation

Suppose you have categorized the terms in your dictionary.

E.g., you identified the ones which provide a pleasent stimulus (“positive”) or the opposite (“negative”).

Or perhaps the ones which indicate uncertainty, activity, hostility, ...

What can you do with this information?

Well, you can compute how often you get a word from a certain category. I.e., with $n_{t,d}$ again the frequency of term t in document d, compute the score

$$s_d = \sum_{t:t \in C} n_{t,d}$$

This is **term scoring**.

In R, e.g. `tm::tm_term_score()`.

General Inquirer

This needs suitable “word lists” (actually, terms, of course).

One resource for this is the **General Inquirer**, which

> *provides tag categories from three sources (the Harvard IV-4 dictionary (H4), the Lasswell value dictionary (LVD), and several categories additionally constructed) and “marker” categories “primarily developed as a resource for disambiguation, but also available to users”.*

This actually features 182 categories. However, it seems that the resource has disappeared from the web (the home page used to be `http://www.wjh.harvard.edu/~inquirer/` but this no longer works.

General Inquirer

The good news is that of course there is an R package on the WU datacube repository:

```
install.packages("tm.lexicon.GeneralInquirer",
                 repos = "https://datacube.wu.ac.at/",
                 type = "source")
```

General Inquirer

As the package title suggests, this primarily provides the GI information as a **lexicon**. E.g.,

```
R> write.dcf(subset(tm.lexicon.GeneralInquirer::General_Inquirer_categories,
+                   Lemma == "good"))

Entry: GOOD#1
Lemma: good
Source: H4Lvd
Categories: Positiv
Categories: Pstv
Categories: Virtue
Categories: EVAL
Categories: PosAff
OthTags: MODIF
OthTags: PFREQ
Defined: 89% noun-adj-adv-intj: Marks positive evaluation on a
        dimension specified by context, or, occasionally, general
        intensification--as "a good many," "a good beating"
```

General Inquirer / 2

```
Entry: GOOD#2
Lemma: good
Source: H4Lvd
Categories: Ovrst
Categories: Time.
Categories: SureLw
OthTags: NOUN
OthTags: HANDELS
Defined: 1% idiom-adv: "For good"--finally, for keeps

Entry: GOOD#3
Lemma: good
Source: H4Lvd
OthTags: HANDELS
Defined: 1% idiom-adv-prep: "A good deal"--to a considerable amount,
        extent or degree-- handled by "deal"

Entry: GOOD#4
```

General Inquirer / 3

```
Lemma: good
Source: H4Lvd
OthTags: HANDELS
Defined: 0% idiom-noun: "Good faith"--handled by "faith"
```

Oh well . . . (it does show 4 “word senses”, but these are not comprehensive).

General Inquirer

It also provides the “word lists” (according to the lemmas, so ideally you would have a decent/matching lemmatizer).

In particular,

```
R> pterms <-
+     tm.lexicon.GeneralInquirer::terms_in_General_Inquirer_categories("Positiv")
R> length(pterms)
```

```
[1] 1637
```

```
R> sample(pterms, 13)
```

```
 [1] "adept"         "captivation"   "understood"    "rave"
 [5] "distinguished" "genius"        "undoubtedly"   "frugal"
 [9] "pure"          "amusement"     "ingenious"     "correct"
[13] "relevant"
```

General Inquirer

Similarly,

```
R> nterms <-
+     tm.lexicon.GeneralInquirer::terms_in_General_Inquirer_categories("Negativ")
R> length(nterms)
```

```
[1] 2005
```

```
R> sample(nterms, 13)
```

```
 [1] "defect"     "anomalous"  "wily"       "casualty"   "deplorable"
 [6] "irritation" "horrible"   "feverish"   "censure"    "misery"
[11] "abnormal"   "sorrow"     "scold"
```

Financial Phrase Bank data: GI

Let's take the Financial Phrase Bank data again, this time without the fancy terms dropping for LDA:

```
R> x <- tm::Corpus(tm::VectorSource(fpb$sentence))
R> m <- tm::DocumentTermMatrix(x,
+                              control = list(removePunctuation = TRUE,
+                                             removeNumbers = TRUE,
+                                             stopwords = TRUE))
R> m

<<DocumentTermMatrix (documents: 4846, terms: 9212)>>
Non-/sparse entries: 54284/44587068
Sparsity           : 100%
Maximal term length: 25
Weighting          : term frequency (tf)
```

Financial Phrase Bank data: GI

How many of the terms are in the GI positive and negative lists?

```
R> length(intersect(tm::Terms(m), pterms))
```

```
[1] 439
```

```
R> length(intersect(tm::Terms(m), nterms))
```

```
[1] 220
```

Well, at least some.

Compute the raw positivity and negativity scores via term scoring:

```
R> pscores <- tm::tm_term_score(m, pterms)
R> nscores <- tm::tm_term_score(m, nterms)
```

Financial Phrase Bank data: GI

Presumably, we should look at the difference of these scores?

```
R> delta <- pscores - nscores
R> with(fpb, table(delta, label))
```

```
     label
delta negative neutral positive
   -3        4       2        0
   -2       23      31       15
   -1      120     236       89
   0       245    1245      447
   1       175     884      518
   2        29     351      203
   3         6      97       71
   4         2      20       13
   5         0      10        3
   6         0       2        4
   7         0       1        0
```

Financial Phrase Bank data: GI

Or perhaps better to look at the conditional frequencies:

```
R> round(prop.table(with(fpb, table(delta , label)), 1), 3)
```

```
     label
delta negative neutral positive
   -3    0.667   0.333    0.000
   -2    0.333   0.449    0.217
   -1    0.270   0.530    0.200
   0     0.126   0.643    0.231
   1     0.111   0.561    0.328
   2     0.050   0.602    0.348
   3     0.034   0.557    0.408
   4     0.057   0.571    0.371
   5     0.000   0.769    0.231
   6     0.000   0.333    0.667
   7     0.000   1.000    0.000
```

Financial Phrase Bank data: GI

This does not look too bad.

However:

```
R> pROC::auc(fpb$label == "positive", delta)
```

```
Area under the curve: 0.5799
```

```
R> pROC::auc(fpb$label != "negative", delta)
```

```
Area under the curve: 0.6292
```

and we've already done much better for the first!

(Remember: the basic ROC curves need a binary response.)

Financial Phrase Bank data: GI

Perhaps better to suitable standardize the score differences? E.g, relative to the total number of words?

```
R> delta <- delta / slam::row_sums(m)
R> pROC::auc(fpb$label == "positive", delta)
```

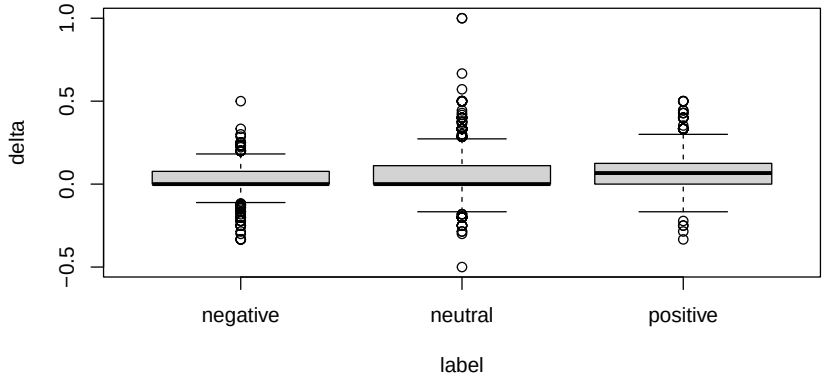
```
Area under the curve: 0.5771
```

Not much better.

Financial Phrase Bank data: GI

Can also illustrate graphically:

```
R> with(fpb, boxplot(delta ~ label))
```



Loughran & McDonald financial sentiment word lists

Loughran and McDonald (2011) introduce word lists intended to be more appropriate for measuring financial sentiment.

Nowadays, available from their "Software Repository for Accounting and Finance" at `https://sraf.nd.edu/`.

Alternatively ... as you may have guessed:

```
install.packages("tm.corpus.Loughran.McDonald.Financial.Sentiment",
                 repos = "https://datacube.wu.ac.at/",
                 type = "source")
```

Well, at least the names are self-explanatory ...

Loughran & McDonald financial sentiment word lists

This now works a bit differently: we need to explicitly load the data set (don't ask):

```
R> data("Loughran_McDonald_Financial_Sentiment",
+       package = "tm.corpus.Loughran.McDonald.Financial.Sentiment")
R> Loughran_McDonald_Financial_Sentiment
```

```
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 6
```

```
R> names(Loughran_McDonald_Financial_Sentiment)
```

```
[1] "Negative"     "Positive"     "Uncertainty" "Litigious"    "ModalStrong"
[6] "ModalWeak"
```

Loughran & McDonald financial sentiment word lists

So now we can guess how to get the positive and negative lists:

```
R> pterms <- NLP::words(Loughran_McDonald_Financial_Sentiment[["Positive"]])
R> nterms <- NLP::words(Loughran_McDonald_Financial_Sentiment[["Negative"]])
```

However,

```
R> head(pterms)
```

```
[1] "ABLE"          "ABUNDANCE"     "ABUNDANT"      "ACCLAIMED"     "ACCOMPLISH"
[6] "ACCOMPLISHED"
```

so we need to lowercase:

```
R> pterms <- tolower(pterms)
R> nterms <- tolower(nterms)
```

Financial Phrase Bank data: Loughran & McDonald

Redo the term scoring for the Financial Phrase Bank data, this time using the Loughran & McDonald lists.

Coverage:

```
R> length(intersect(tm::Terms(m), pterms))
```

```
[1] 137
```

```
R> length(intersect(tm::Terms(m), nterms))
```

```
[1] 196
```

Scores and their difference:

```
R> pscores <- tm::tm_term_score(m, pterms)
R> nscores <- tm::tm_term_score(m, nterms)
R> delta <- pscores - nscores
```

Financial Phrase Bank data: Loughran & McDonald

Joinut frequencies of differences and label:

```
R> with(fpb, table(delta, label))
```

```
     label
delta negative neutral positive
   -3        3       1        0
   -2       28       8       14
   -1      231     231      107
   0       329    2438      907
   1        13     180      283
   2         0      20       44
   3         0       1        6
   4         0       0        1
   5         0       0        1
```

Financial Phrase Bank data: Loughran & McDonald

Conditional frequencies of differences according to label:

```
R> round(prop.table(with(fpb, table(delta , label)), 1), 3)
```

```
     label
delta negative neutral positive
   -3    0.750   0.250    0.000
   -2    0.560   0.160    0.280
   -1    0.406   0.406    0.188
   0     0.090   0.664    0.247
   1     0.027   0.378    0.595
   2     0.000   0.312    0.688
   3     0.000   0.143    0.857
   4     0.000   0.000    1.000
   5     0.000   0.000    1.000
```

Financial Phrase Bank data: Loughran & McDonald

Better than before?

```
R> pROC::auc(fpb$label == "positive", delta)
```

```
Area under the curve: 0.6051
```

```
R> pROC::auc(fpb$label != "negative", delta)
```

```
Area under the curve: 0.7015
```

Well, a bit, but still not exiting (and we did much better with learning the labels using the word2vec embeddings . . . at least in-sample).

Summary

Lessons learned:

- The idea to use “term scores” is very simple.
- We can (of course) very conveniently compute these in R.
- They do not seem to work “very well” out of the box, even when using domain specific word lists.

As we saw, the Financial Phrase Bank texts are generally “strange”.

We might get better results when scoring financial disclosures or blogs, but then how can we know whether these are better?

Feature extraction

Feature extraction

Computing polarity scores can be seen as one special kind of “feature extraction”.

Other features of possible interest might be

- lexical diversity measures (e.g., Gini/entropy of the term frequency distribution)
- fraction of stopwords
- syllable counts
- . . .
- **readability** measures

Readability

This is actually very relevant for “financial texts”:

The information in disclosures should be accessible to market participants and not be hidden in complicated legalese.

So e.g. the SEC mandates using “simple English”.

One can ask how the readability of disclosures impacts market reactions.

However: readability is hard to measure.

Standard readability measures typically relate to how well children can comprehend texts.

Financial texts will necessarily use (some) complicated financial terms.

See the upcoming presentations.

Supervised Learning

Text as predictor

Text as predictor

Once you have fixed length numeric representations/embeddings of texts, you can use them for predictive modeling as you would for other numeric predictors.

E.g.,

- learn to predict “sentiment” (not so easy due to lack of suitable data sets for learning)
- learn to predict market responses directly (numeric response)
- learn to predict buy/sell (discrete/binary response)
- learn to predict . . .

Text as response

Text as response

Once you have fixed length numeric representations/embeddings of texts, you can use them for predictive modeling as you would for other numeric responses.

But we usually do univariate responses, and the embeddings would be highly multivariate? Right.

So typically one models features computed from the representations.

E.g.,

- learn how readability of disclosures varies with regulatory changes (simple English)
- learn more generally how readability varies according to suitable predictors
- learn more generally how (financial) language changes over time (culturomics, e.g., fraction of stopwords or other measures of lexical diversity)

Outlook

More in the the exiting case studies in the last unit of the course!