Topics in Empirical Economics, Part III
Text (i.e. Count) Data and Information Retrieval

Stephen Hansen
Universitat Pompeu Fabra

Introduction

Most empirical work in economics relies on inherently quantitative data: prices, demand, votes, etc.

But a large amount of unstructured text is also generated in economic environments: company reports, policy committee deliberations, court decisions, media articles, political speeches, etc.

One can use such data qualitatively, but increasing interest in treating text quantitatively.

This lecture will review how economists have done this until recently, and the rest will cover more modern machine learning approaches.

We shall also see that the empirical analysis of text is part of a more general problem of treating high-dimensional count data, and discuss other applications.

Theory vs Implementation

For using unstructured data in applications, there is an important computational component that we will not have time to discuss.

A good conceptual understanding of the tools we discuss is necessary but, in many cases, not sufficient for using them productively in your own research.

General-purpose programming languages (Python, C/C++, etc.) are better at handling unstructured data—and associated algorithms—than canned software packages (STATA, GAUSS, etc.).

Resources for Self-Study

My own favorite language is Python: easy; well-developed libraries for scientific computing; huge community; speed-ups available through Cython.

I recommend the following sources for self-study:

1. `http://learnpythonthehardway.org/`
2. `http://quant-econ.net/`
3. *Python for Data Analysis* by Wes McKinney

Note that languages like MATLAB and Julia are fantastic for scientific computing, but have somewhat limited functionality for obtaining, storing, and parsing unstructured data.

Textual Databases

A single observation in a textual database is called a *document*.

The set of documents that make up the dataset is called a *corpus*.

We often have covariates associated with each document that are sometimes called *metadata*.

NB: Getting text data into an organized database is often a challenge.

Example

In "Transparency and Deliberation" we use a corpus of verbatim FOMC transcripts from the era of Alan Greenspan:

- 149 meetings from August 1987 through January 2006.
- A document is a single statement by a speaker in a meeting (46,502).
- Associated metadata: speaker biographical information, macroeconomic conditions, etc.

What is Text?

At an abstract level, text is simply a string of characters.

Some of these may be from the Latin alphabet—'a', 'A', 'p' and so on—but there may also be:

1. Decorated Latin letters (e.g. ö)
2. Non-Latin alphabetic characters (e.g. Chinese and Arabic)
3. Punctuation (e.g. '!')
4. White spaces, tabs, newlines
5. Numbers
6. Non-alphanumeric characters (e.g. '@')

Key Question: How can we obtain an informative, quantitative representation of these character strings? This is the goal of text mining.

First step is to *pre-process* strings to obtain a cleaner representation.

Pre-Processing I: Tokenization

Tokenization is the splitting of a raw character string into individual elements of interest.

Often these elements are words, but we may also want to keep numbers or punctuation as well.

Simple rules work well, but not perfectly. For example, splitting on white space and punctuation will separate hyphenated phrases as in 'risk-averse agent' and contractions as in 'aren't'.

In practice, you should (probably) use a specialized library for tokenization.

Pre-Processing II: Stopword Removal

The frequency distribution of words in natural languages is highly skewed, with a few dozen words accounting for the bulk of text.

These *stopwords* are typically stripped out of the tokenized representation of text as they take up memory but do not help distinguish one document from another.

Examples from English are 'a', 'the', 'to', 'for' and so on.

No definitive list, but example on `http://snowball.tartarus.org/algorithms/english/stop.txt`.

Pre-Processing II: Stopword Removal

The frequency distribution of words in natural languages is highly skewed, with a few dozen words accounting for the bulk of text.

These *stopwords* are typically stripped out of the tokenized representation of text as they take up memory but do not help distinguish one document from another.

Examples from English are 'a', 'the', 'to', 'for' and so on.

No definitive list, but example on `http://snowball.tartarus.org/algorithms/english/stop.txt`.

Also common to drop rare words, for example those that appear is less than some fixed percentage of documents.

Pre-Processing III: Linguistic Roots

For many applications, the relevant information in tokens is their linguistic root, not their grammatical form. We may want to treat 'prefer', 'prefers', 'preferences' as equivalent tokens.

Two options:

- *Stemming*: Deterministic algorithm for removing suffixes. Porter stemmer is popular.
  Stem need not be an English word: Porter stemmer maps 'inflation' to 'inflat'.
- *Lemmatizing*: Tag each token with its part of speech, then look up each (word, POS) pair in a dictionary to find linguistic root.
  E.g. 'saw' tagged as verb would be converted to 'see', 'saw' tagged as noun left unchanged.

A related transformation is *case-folding* each alphabetic token into lowercase. Not without ambiguity, e.g. 'US' and 'us' each mapped into same token.

Pre-Processing of FOMC Corpus

	All terms	Alpha terms	No stopwords	Stems
# total terms	6249776	5519606	2505261	2505261
# unique terms	26030	24801	24611	13734

From Tokens to Counts

After pre-processing, each document is a finite list of terms.

A basic quantitative representation of a corpus is the following:

- Index each unique term in the corpus by some $v \in \{1, \ldots, V\}$ where V is the number of unique terms.
- For each document $d \in \{1, \ldots, D\}$ compute the count $x_{d,v}$ as the number of occurrences of term v in document d.
- The $D \times V$ matrix $\mathbf{X}$ of all such counts is called the *document-term matrix*.

This representation is often called the *bag-of-words* model, or, in probabilistic modeling, the *unigram* model.

Example

Doc1 = [‘text’, ‘mining’, ‘is’, ‘more’, ‘fun’, ‘than’, ‘coal’, ‘mining’]

has the bag-of-words representation

text	mining	is	more	fun	than	coal
1	2	1	1	1	1	1

Example

Doc1 = [‘text’, ‘mining’, ‘is’, ‘more’, ‘fun’, ‘than’, ‘coal’, ‘mining’]

has the bag-of-words representation

text	mining	is	more	fun	than	coal
1	2	1	1	1	1	1

Note that

Doc2 = [‘coal’, ‘mining’, ‘is’, ‘more’, ‘fun’, ‘than’, ‘text’, ‘mining’]

also shares the same representation.

The bag-of-words model is useful for describing content, but we lose all information about sentence structure.

Ngram Representation

Some limitations of the unigram model can be overcome by counting all unique N-length term sequences in the corpus. This is called the *Ngram* model.

The model with $N = 2$ ($N = 3$) is sometimes called the *bigram* (*trigram*) model.

The bigram representation of Doc1 is

(text, mining)	(mining, is)	(is, more)	(more, fun)	...
1	1	1	1	...

Note that we can still form a matrix of counts with an Ngram model, we just need to redefine the column indices to correspond to the set of unique Ngrams.

Text Mining and Information

Any useful representation of text will throw away some information; that's the essential purpose of text mining.

The question is whether we are keeping the relevant information for our needs, and getting rid of the extraneous information.

The answer cannot be separated from the context in which we are using text. The bag-of-words model can perform very well for identifying documents of interest, and terribly for translation models.

Features of the Document-Term Matrix

The key characteristics of the document-term matrix are its:

1. High dimensionality
2. Sparsity

The rest of the course will be about how to treat such objects in empirical work, and so is part of a larger statistical problem.

Alternative Application I: Survey Data

Ongoing project to document CEO time use (with O. Bandiera, A. Prat, and R. Sadun), and its effect on firm performance.

Data on each 15-minute block of time for one week of 1,114 CEOs' time classified according to

1. type (e.g. meeting, public event, etc.)
2. duration (15m, 30m, etc.)
3. planning (planned or unplanned)
4. number of participants (one, more than one)
5. functions of participants, divided between employees of the firms or "insiders" (finance, marketing, etc.) and "outsiders" (clients, banks, etc.).

There are 4,253 unique combination of these five features in the data.

One can summarize the data with a 1114×4253 matrix where the (i, j)th element is the number of 15-minute time blocks that CEO i spends in activities with a particular combination of features.

Alternative Application II: Demand

Increasingly common to get detailed data on consumers' shopping behavior.

Imagine a dataset in which consumer $i \in \{1, \ldots, N\}$ makes multiple visits to a store that sells M possible goods bundles.

Then we can define an $N \times M$ matrix that counts the number of times each consumer buys each bundle.

Dimensionality Reduction through Keywords

The first approach to handling $\mathbf{X}$ is to limit attention to a subset of columns of interest.

In the natural language context, this is equivalent to representing text using the distribution of keywords across documents.

One can either look at the incidence of keywords (Boolean search), or else their frequency (dictionary methods).

The researcher must decide in advance which are the keywords of interest.

Application

The recent work of Baker, Bloom, and Davis on measuring economic policy uncertainty (`http://www.policyuncertainty.com/`) is largely based on a media index constructed via Boolean searches of US and European newspapers.

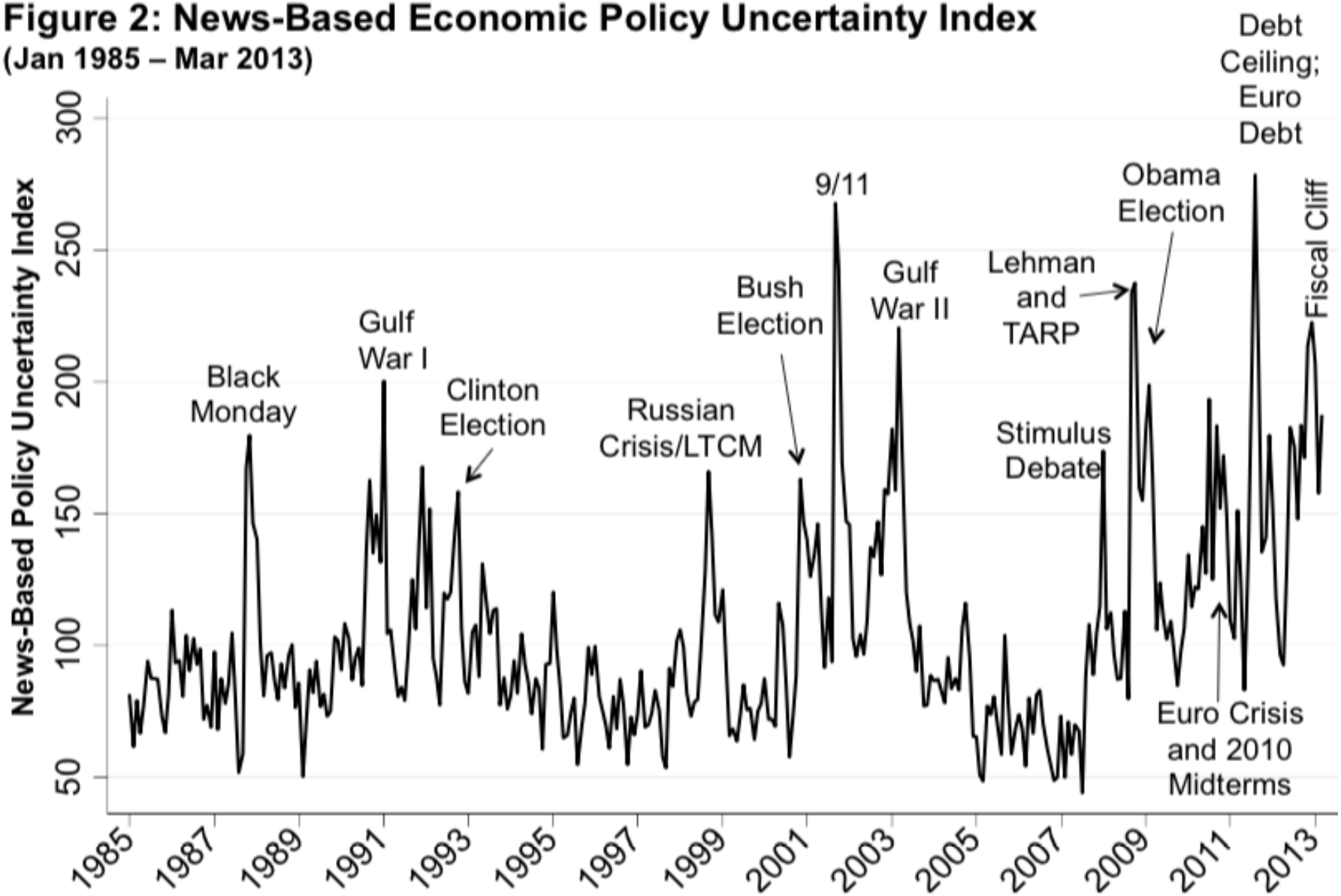
For each paper on each day since 1985, identify articles that contain:

1. "uncertain" OR "uncertainty", AND
2. "economic" OR "economy", AND
3. "congress" OR "deficit" OR "federal reserve" OR "legislation" OR "regulation" OR "white house"

Normalize resulting article counts by total newspaper articles that month.

Results

Figure 2: News-Based Economic Policy Uncertainty Index
(Jan 1985 – Mar 2013)



Dictionary Methods

Let $\mathfrak{D}$ be the set of keywords of interest.

We can then represent each document d as $x_d = \sum_{v \in \mathfrak{D}} x_{d,v}$ or perhaps normalize, e.g. $s_d = \sum_{v \in \mathfrak{D}} x_{d,v} / \sum_v x_{d,v}$.

Dictionary methods consider the intensity of word use to be informative.

Tetlock (2007)

Tetlock (2007) is a highly cited paper that applies dictionary methods to the Wall Street Journal's "Abreast of the Market" column.

Uses Harvard IV-4 dictionaries `http://www.wjh.harvard.edu/~inquirer`.

Large number of categories: positive, negative, pain, pleasure, rituals, natural processes, etc. 77 in all.

Count number of words in each dictionary in each column from 1984-1999.

Principal components analysis shows most variation on dimensions that reflect pessimism: negative, weak, fail, fall.

Tetlock (2007)

Tetlock (2007) is a highly cited paper that applies dictionary methods to the Wall Street Journal's "Abreast of the Market" column.

Uses Harvard IV-4 dictionaries `http://www.wjh.harvard.edu/~inquirer`.

Large number of categories: positive, negative, pain, pleasure, rituals, natural processes, etc. 77 in all.

Count number of words in each dictionary in each column from 1984-1999.

Principal components analysis shows most variation on dimensions that reflect pessimism: negative, weak, fail, fall.

Main result: pessimism predicts low short-term returns (measured with the Dow Jones index) followed by reversion.

Loughran and McDonald (2011)

Following Tetlock (2007), popular to use just negative word dictionary from Harvard IV-4.

This includes words like 'tax', 'cost', 'capital', 'liability', and 'vice'.

Unclear that these are appropriate for describing negative content in financial context.

Loughran and McDonald (2011) use 10-K filings to define their own finance-specific word lists, available from `http://www3.nd.edu/~mcdonald/Word_Lists.html`.

Negative list includes words like 'restated', 'litigation', 'termination', 'unpaid', 'investigation', etc.

Loughran and McDonald (2011)

Following Tetlock (2007), popular to use just negative word dictionary from Harvard IV-4.

This includes words like 'tax', 'cost', 'capital', 'liability', and 'vice'.

Unclear that these are appropriate for describing negative content in financial context.

Loughran and McDonald (2011) use 10-K filings to define their own finance-specific word lists, available from `http://www3.nd.edu/~mcdonald/Word_Lists.html`.

Negative list includes words like 'restated', 'litigation', 'termination', 'unpaid', 'investigation', etc.

Main result: the context-specific list has greater predictive power for return regressions than the generic one.

Term Weighting

Dictionary methods are based on raw counts of words.

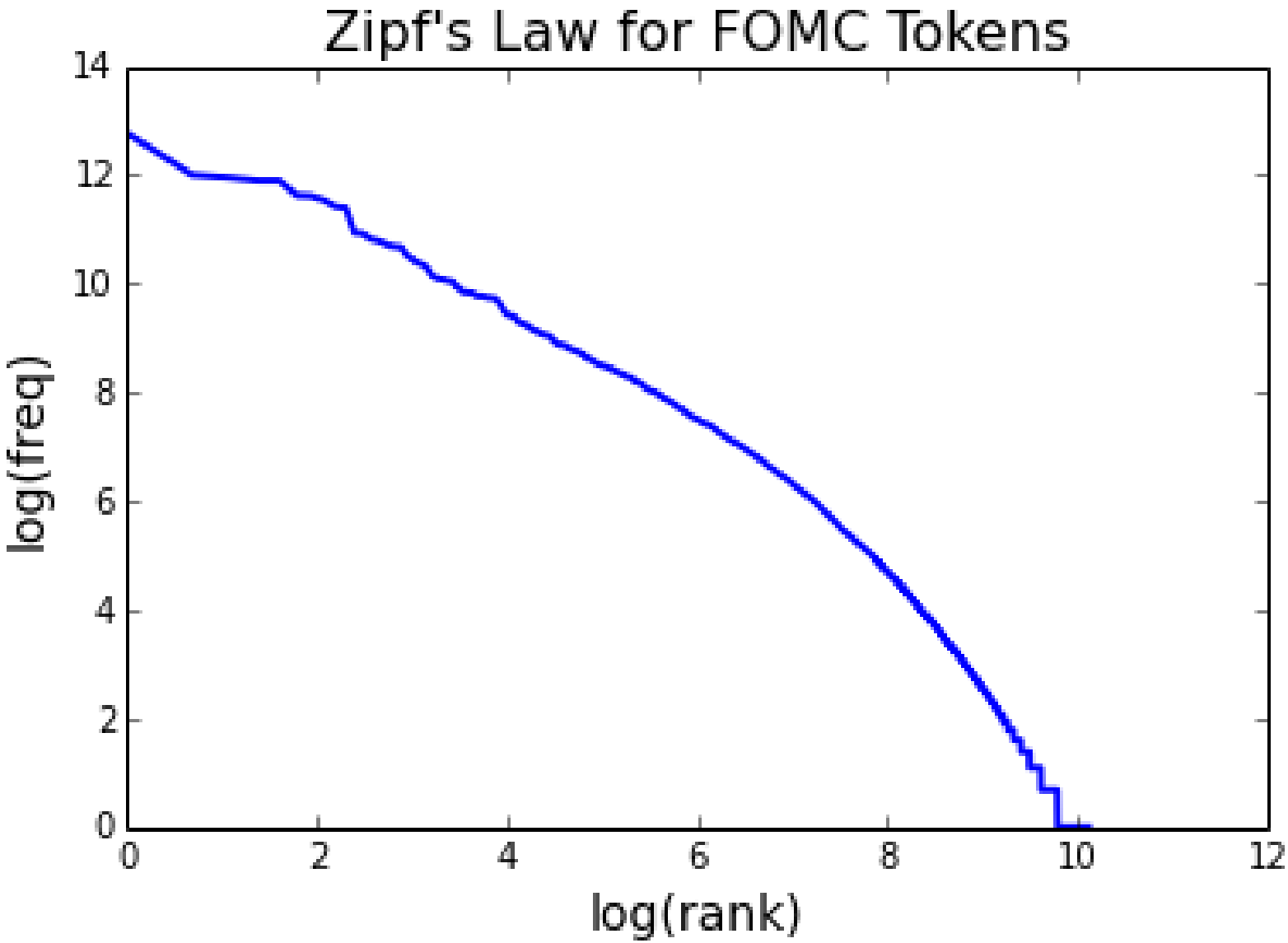
But the frequency of words in natural language can distort raw counts.

Zipf's Law is an empirical regularity for most natural languages that maintains that the frequency of a particular term is inversely proportional to its rank.

Means that a few terms will have very large counts, many terms have small counts.

Example of a *power law*.

Zipf's Law in FOMC Transcript Data



Rescaling Counts

Let $x_{d,v}$ be the count of the vth term in document d.

To dampen the power-law effect can express counts as

$$tf_{d,v} = \begin{cases} 0 & \text{if } x_{d,v} = 0 \\ 1 + \log(x_{d,v}) & \text{otherwise} \end{cases}$$

which is the *term frequency* of v in d.

Thought Experiment

Consider a two-term dictionary $\mathfrak{D} = \{v', v''\}$.

Suppose two documents d' and d'' are such that:

$$x_{d',v'} > x_{d'',v'} \text{ and } x_{d',v''} < x_{d'',v''}.$$

Now suppose that no other document uses term v' but every other document uses term v''.

Which document is "more about" the theme the dictionary captures?

Inverse Document Frequency

Let df_v be the number of documents that contain the term v.

The *inverse document frequency* is

$$\text{idf}_v = \log\left(\frac{D}{df_v}\right),$$

where D is the number of documents.

Properties:

1. Higher weight for words in fewer documents.
2. Log dampens effect of weighting.

TF-IDF Weighting

Combining the two observations from above allows us to express the *term frequency - inverse document frequency* of term v in document d as

$$\text{tf-idf}_{d,v} = tf_{d,v} \times idf_v.$$

Gives prominence to words that occur many times in few documents.

Can now score each document as $s_d = \sum_{v \in \mathfrak{D}} \text{tf-idf}_{d,v}$ and then compare.

In practice, this provides better results than simple counts.

Note that divergence between generic and specific dictionaries in Loughran and McDonald (2011) is greatly reduced after tf-idf correction.

Data-Driven Stopwords

Stopword lists are useful for generic language, but there are also context-specific frequently used words.

For example, in a corpus of court proceedings, words like 'lawyer', 'law', 'justice' will show up a lot.

Can also define term-frequency across entire corpus as

$$tf_v = 1 + \log\left(\sum_d x_{d,v}\right).$$

One can then rank each term in the corpus according to $tf_v \times idf_v$, and choose a threshold below which to drop terms.

This provides a means for data-driven stopword selection.

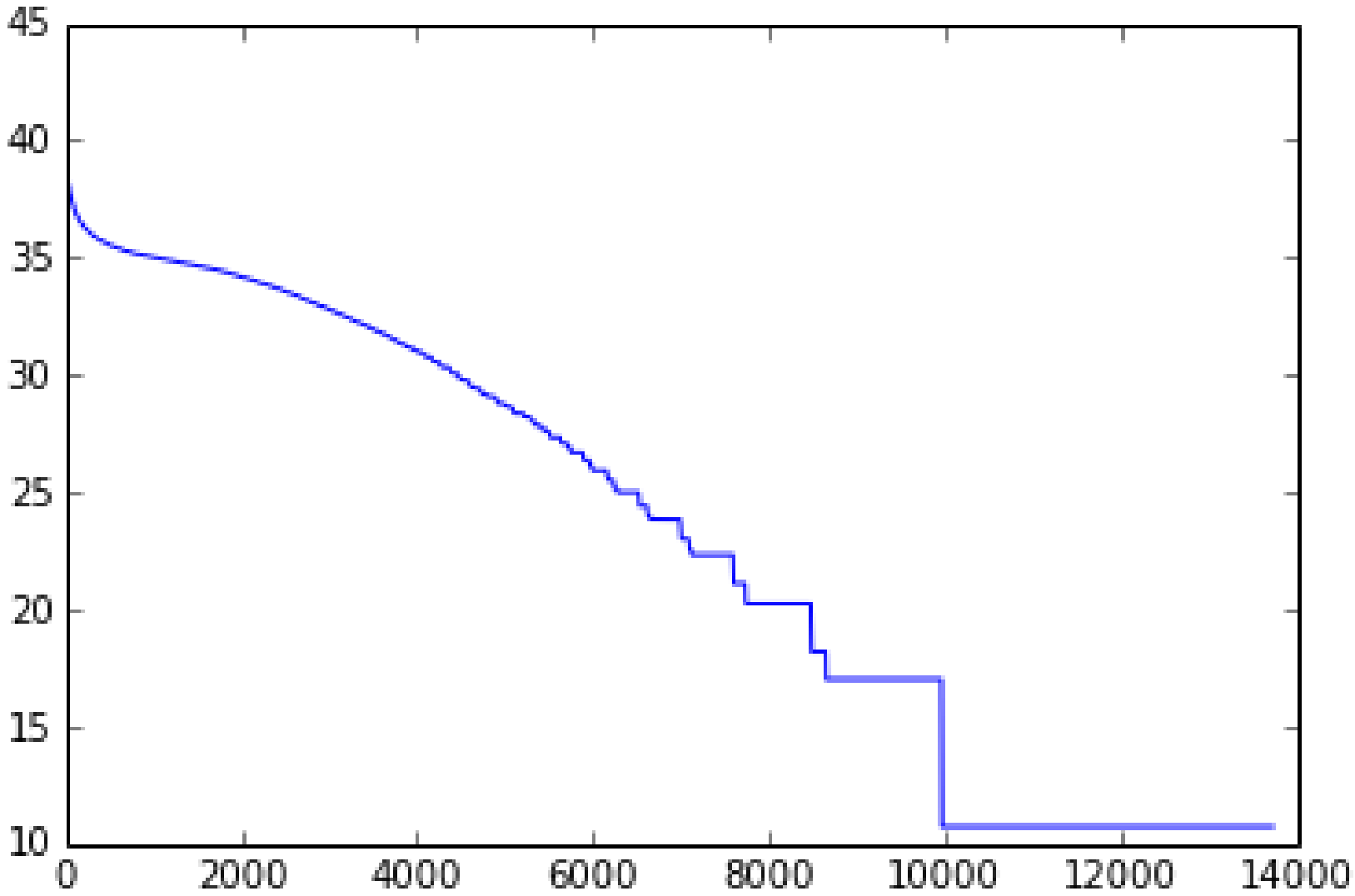
Stem Rankings in FOMC Transcript Data

R1 = collection frequency ranking

R2 = tf-idf-weighted ranking

Rank	1	2	3	4	5	6	7	8	9
R1	rate	think	year	will	market	growth	inflat	price	percent
R2	panel	katrina	graph	fedex	wal	mart	mbs	mfp	euro

Ranking of All FOMC Stems



Vector Space Model

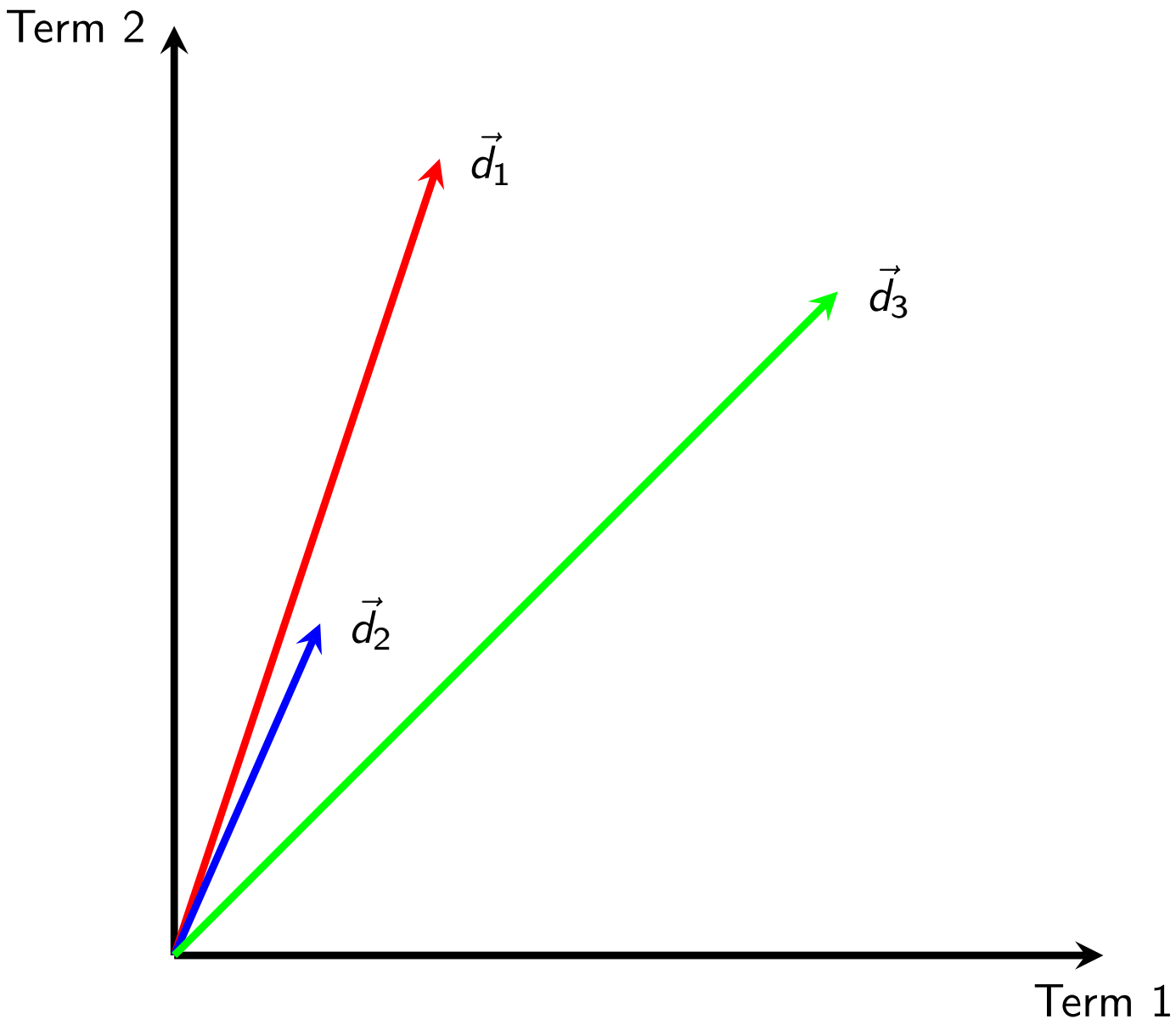
One can also view rows of document-term matrix as vectors lying in a V-dimensional space, and represent document i as $\vec{d}_i$.

Tf-idf weighting usually used, but not necessary.

The question of interest is how to measure the similarity of two documents in the vector space.

Initial instinct might be to use Euclidean distance $\sqrt{\sum_v (x_{i,v} - x_{j,v})^2}$.

Three Documents



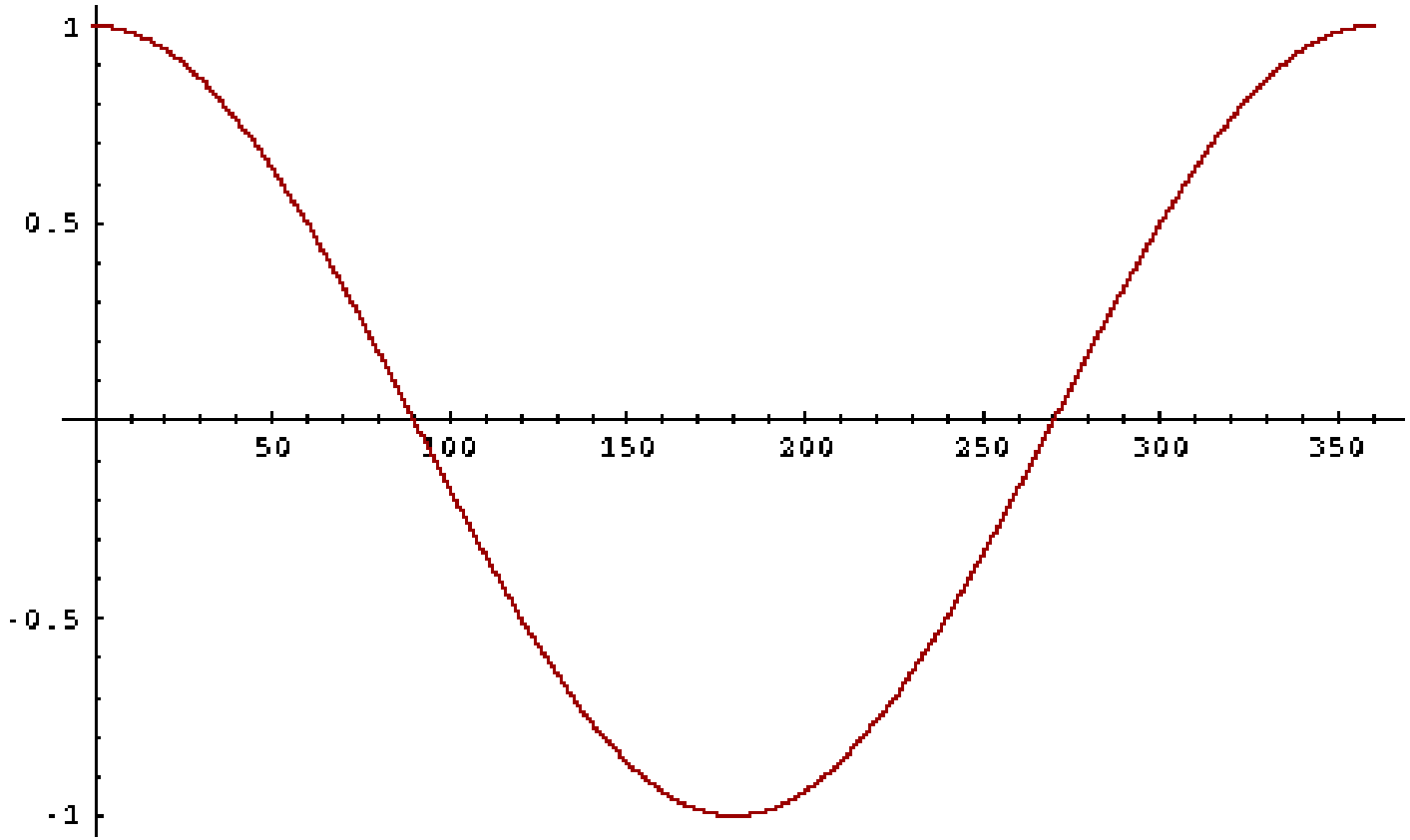
Problem with Euclidean Distance

Semantically speaking, documents 1 and 2 are very close, and document 3 is an outlier.

But the Euclidean distance between 1 and 2 is high due to differences in document length.

What we really care about is whether vectors point in same direction.

Cosine



Cosine Similarity

Define the cosine similarity between documents i and j as

$$CS(i,j) = \frac{\vec{d}_i \cdot \vec{d}_j}{\left\|\vec{d}_i\right\| \left\|\vec{d}_j\right\|}$$

1. Since document vectors have no negative elements $CS(i,j) \in [0,1]$.
2. $\vec{d}_i / \left\|\vec{d}_i\right\|$ is unit-length, correction for different distances.

Application

An important theoretical concept in industrial organization is location on a product space.

Industry classification measures are quite crude proxies of this.

Hoberg and Phillips (2010) take product descriptions from 49,408 10-K filings and use the vector space model (with bit vectors defined by dictionaries) to compute similarity between firms.

Data available from `http://alex2.umd.edu/industrydata/`.

Towards Machine Learning

Dictionary methods focus on variation across observations along a limited number of dimensions and ignore the rest.

Ideally we would use variation across *all* dimensions to describe documents.

This obviously provides a richer description of the data, but a deeper point relevant for many high-dimensional datasets is that economic theory does not tell us which dimensions are important.

At the same time, incorporating thousands of independent dimensions of variation in empirical work is difficult.

Machine learning approaches exploit all dimensions of variation to estimate a lower-dimensional set of types for each documents.