Big Data: Text

Matt Taddy, University of Chicago Booth School of Business

faculty.chicagobooth.edu/matt.taddy/teaching

Text

The Bag of Words Representation

Using what we know: term frequencies as model inputs.

Tokenization: ngrams, stopwords, info retrieval.

Text-specific multinomial models.

We'll start with a story: Slant in Partisan Speech

Gentzkow and Shapiro: What drives media slant? Evidence from U.S. daily newspapers (*Econometrica*, 2010).

Build an economic model for newspaper demand that incorporates political partisanship (Republican vs Democrat).

- What would be independent profit-maximizing "slant"?
- Compare this to slant estimated from newspaper text.

Jerry Moran, R-KS, says “death tax” relatively often and his district (Kansas 1st) voted 73% for George W. Bush in 2004.

$$\mathbf{X}_{\text{text}} = f(\text{ideology}) \approx g(Y_{Bush})$$

$$\Rightarrow \text{“death tax” is republican}$$

$$\Rightarrow \text{the Wall Street Journal is slanted right.}$$

What is slant?

Text: phrase-counts by speaker in 109^{th} US Congress (05-06)
Sentiment: two-party constituent vote-share for Bush in 2004.

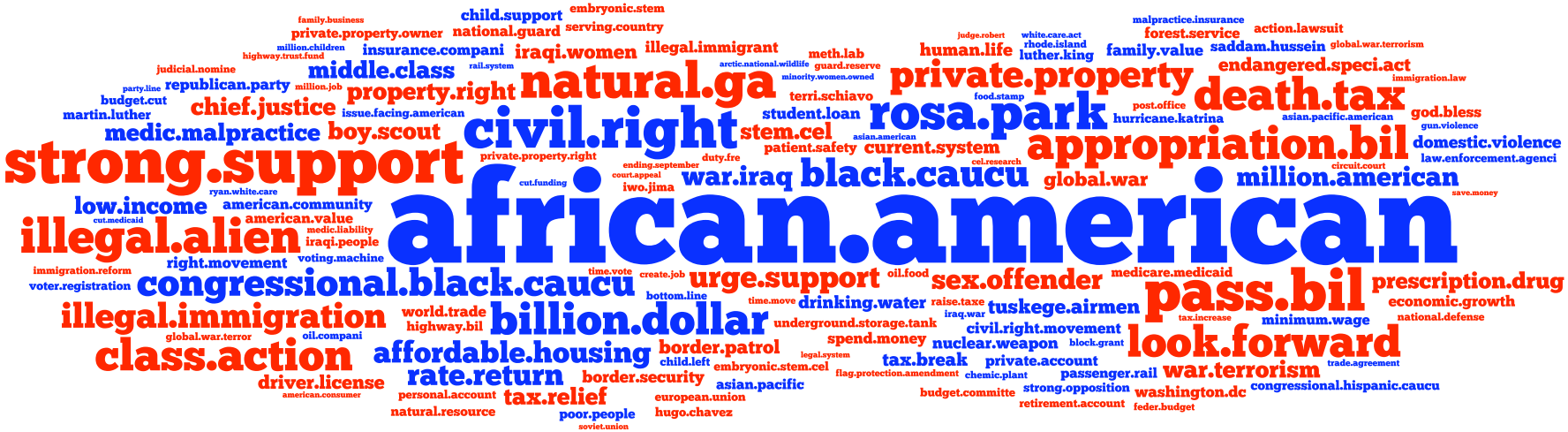
Use covariance between phrase frequencies (f_{ij}) and 'Bush' sentiment (y_i) to build an index of partisanship for text.

$$z_i^{slant} = \sum_j \mathrm{cov}(f_j, y) f_{ij}$$

For example, if phrase *j* forms a high proportion of what you say, and usage of phrase *j* is correlated with Bush vote-share, then this contributes a positive amount to your slant score.

This is a type of *marginal regression*.

Wordle



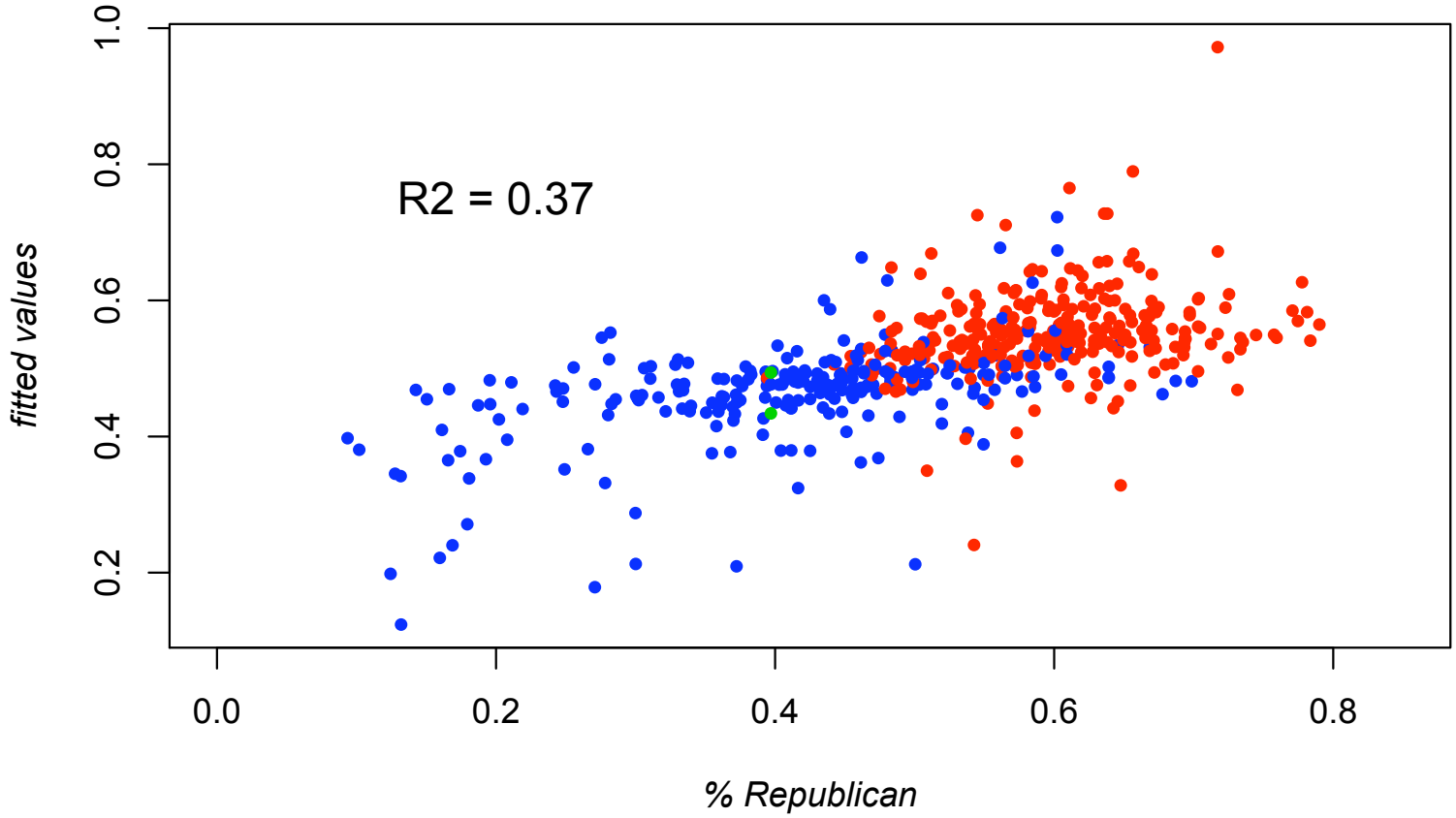
Colored by sign (positive, negative)

Size proportional to loading $\mathrm{cov}(f_j, y)$.

Since y is Republican vote-share,

big positive is a right term and big negative is a left term.

Slant measure for speakers in the 109th Congress



Democrats get low z_{slant} and Republicans get high z_{slant}.

Do this for newspaper text and you'll get a similar picture

The Big Picture

Text is a vast source of data for business

It comes connected to interesting “author” variables

- What you buy, what you watch, your reviews
- Group membership, who you represent, who you email
- Market behavior, macro trends, the weather

Opinion, subjectivity, etc. Sentiment is *very* loosely defined:

Observables linked to the variables motivating language choice

Text is also super high dimensional

And it gets higher dimensional as you observe more speech.

Analysis of phrase counts is the state of the art (hard to beat).

For example, occurances by party for some partisan terms

Congress	State	Party	America	Death Tax	Estate Tax	⋯
63	NM	dem	108	30	140	
		gop	100	220	12	

Basic units of data

- doc-term count matrix $\mathbf{X}$ with rows $\mathbf{x}_i$.
- doc totals $\mathbf{m} = \texttt{rowSums}(\mathbf{X}) = [m_1 \cdots m_n]$.
- frequency matrix $\mathbf{F} = \mathbf{X}/\mathbf{m}$ with rows $\mathbf{f}_i$.

We already know how to do text mining

Doc-term matrices can be treated like any other covariates.

Example: Author Classification

You want to identify 'authorship' from from text.
Use logistic lasso to estimate authors from **X**.
We've already seen this: the spam data!

$$\text{logit}\left[\text{p}(\texttt{spam})\right] = \alpha + \mathbf{f}'\boldsymbol{\beta}$$

It is best to regress on frequencies **F** rather than raw counts.
If you think m_i matters, include it as a separate variable.

We also did some text mining with the we8there reviews.

Topic models

```
summary(tpcs)
Top 5 phrases by topic-over-null term lift (and usage %):
[1] food great, great food, veri good, food veri, veri nice (13.7)
[2] over minut, ask manag, flag down, speak manag, arriv after (11.6)
```

And regression

```
stars <- we8thereRatings[,"Overall"]
tpcreg <- gamlr(tpcs$omega, stars)
# Effect stars from 10% increase in topic use
drop(coef(tpcreg))*0.1
     intercept          1          2          3          4
         0.414      0.075     -0.386      0.068      0.042
             5          6          7          8          9         10
         0.000      0.076      0.121      0.000     -0.134     -0.049
```

Wait: how did you get two word phrases?

Information Retrieval and Tokenization

A passage in '*As You Like It*':

> *All the world's a stage,*
> *and all the men and women merely players:*
> *they have their exits and their entrances;*
> *and one man in his time plays many parts...*

What the statistician sees:

world	stage	men	women	play	exit	entrance	time
1	1	2	1	2	1	1	1

This is the Bag-of-Words representation of text.

The *n*-gram language model

An *n*-gram language model is one that describes a dialect through transition probabilities on *n* consecutive words.

An *n*-gram tokenization counts length-*n* sequences of words.
A unigram is a word, bigrams are transitions between words.
e.g., `world.stage`, `stage.men`, `men.women`, `women.play`, ...

This can give you rich language data, but be careful:
n-gram token vocabularies are very high dimensional (p^n)

More generally, you may have domain specific 'clauses' that you wish to tokenize. There is always a trade-off between complexity and generality. Often best to just count words.

Possible tokenization steps

Remove words that are super rare (in say $< \frac{1}{2}\%$, or $< 15\%$ of docs; this is application specific). For example, if Aegrotat occurs only once, it's useless for comparing documents.

Stemming: 'tax' $\leftarrow$ taxing, taxes, taxation, taxable, ...
A stemmer cuts words to their root with a mix of rules and estimation. 'Porter' is standard for English. I don't usually stem since you can lose meaning, but it is nice for limited data.

Remove a list of stop words containing irrelevant tokens.

If, and, but, who, what, the, they, their, a, or, ...

Be careful: one person's stopword is another's key term.

Convert to lowercase, drop numbers, punctuation, etc ...
Always application specific: e.g., don't drop :-) from tweets.

Tokenization in R and elsewhere

The `tm` package for R provides a set of tools
for cleaning and tokenizing text documents.
See class code for examples of reading from text and pdf files.

This works well for small-to-medium sized examples, but
to really scale up you'll need to work out-of-memory.
There is an extension of `tm` for Hadoop and Mapreduce.

Python is a more natural scripting language for tokenization.
There is a massive ecosystem of tools available for dealing with
foreign encodings, spell-checking, web scraping, etc...

Document-Term Co-occurrence Matrices

However you tokenize,
the final result is a document-term count matrix **X**.

Each row is a document, each column a token (e.g., word).
These are the basic units of data for text analysis ($\mathbf{F} = \mathbf{X}/\mathbf{m}$).

Some lingo

- The vocabulary is `colnames(X)`.
- Vocabulary size is `ncol(X)`.
- The corpus is 'all of your documents'.
- Corpus size is `nrow(X)`.
- A token's sparsity is `mean(X[,'term']==0)`.

Term frequencies are data like any other

In addition to lasso regression, we can do

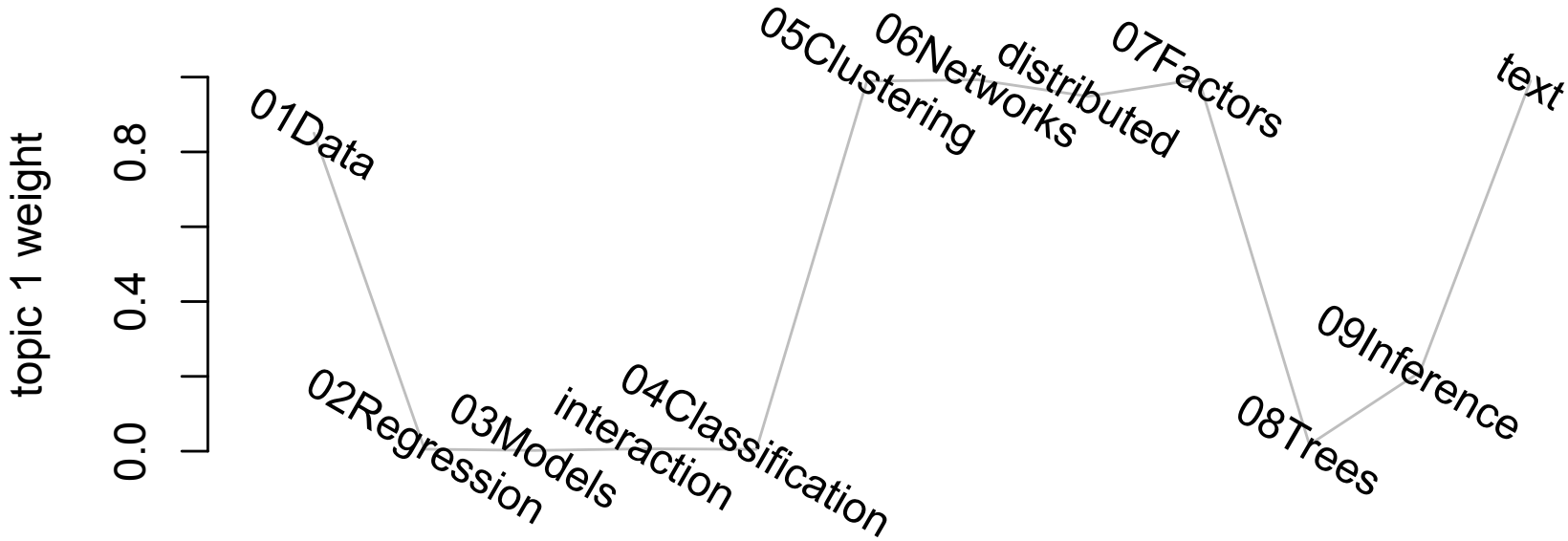
- Principle components analysis
- Partial least squares (marginal regression)
- K-means clustering

And there are fast approximate algorithms
(see `R`'s `irlba` package).

Even better is to use some text-specific methods...

Example: topics on lectures from 2013 Booth 41201

We used `tm` to read lectures from pdf and tokenize.



Our topic-factors are 'not regression' and 'regression'.

Note that $\omega_{i2} = 1 - \omega_{i1}$, so this is really a one-factor model.

Also: this is 2013; the class looks different today!

Building regression models *for* text

When regressing onto **F**, we are inverting causation:

usually sentiment causes word choice, not the opposite.

What if we were to model text as a function of y?
We have a good model for **X**: the multinomial!

A logistic multinomial text model:

$$\mathbf{x}_i \sim \mathrm{MN}(\mathbf{p}_i, m_i) \text{ with } \mathrm{p}_{ij} = \frac{\exp[\alpha_j + \mathbf{v}_i'\varphi_j]}{\sum_{l=1}^{p} \exp[\alpha_l + \mathbf{v}_i\varphi_l]}$$

whre $m_i = \sum_j x_{ij}$ and φ_j is called the j^{th} loading.

v can be any document attributes:
author characteristics, date, beliefs, sentiment, ...

Back to we8there

Say $\mathbf{v}_i$ is the vector of five aspect ratings:
overall, atmosphere, value, food, service.

We can say that words are a function of these five things,
and see that the loadings look like.

For example, if '`overall`' rating goes from $4 \rightarrow 5$,
log odds of 'good food' relative to 'bad food' increase by

$$\varphi_{\texttt{goodfood, overall}} - \varphi_{\texttt{badfood, overall}} \approx 0.9 \qquad (e^{0.9} = 2.5)$$

As in any multivariate regression, this is a *partial effect*:
it assumes all other aspect ratings are held constant.

The Big Data angle

A huge efficiency here comes from the fact that sums of multinomials with equal **q** yield another multinomial.

$$\text{If } \mathbf{x}_1 \sim \text{MN}(\mathbf{p}, m_1) \text{ and } \mathbf{x}_2 \sim \text{MN}(\mathbf{p}, m_2),$$
$$\text{then } \mathbf{x}_1 + \mathbf{x}_2 \sim \text{MN}(\mathbf{p}, m_1 + m_2).$$

$\Rightarrow$ If v has B levels, we only fit B 'observations'.

If your v is continuous, consider binning on quantiles for speed.

Use MapReduce to collapse across individuals, then MNIR.

`mnlm` will automatically collapse the data if you set '`bins`'.

e.g., `myfit = mnlm(counts = X, covars=y, bins=2)`

If you look at my papers, you'll also see that estimation can be distributed to independent machines: 'build out not up'.

we8there: big absolute loadings by aspect

Overall	Food	Service	Value	Atmosphere
plan.return	again.again	cozi.atmospher	big.portion	walk.down
feel.welcom	mouth.water	servic.terribl	around.world	great.bar
best.meal	francisco.bay	servic.impecc	chicken.pork	atmospher.wonder
select.includ	high.recomend	attent.staff	perfect.place	dark.wood
finest.restaur	cannot.wait	time.favorit	place.visit	food.superb
steak.chicken	best.servic	servic.outstand	mahi.mahi	atmospher.great
love.restaur	kept.secret	servic.horribl	veri.reason	alway.go
ask.waitress	food.poison	dessert.great	babi.back	bleu.chees
good.work	outstand.servic	terribl.servic	low.price	realli.cool
can.enough	far.best	never.came	peanut.sauc	recommend.everyon
after.left	food.awesom	experi.wonder	wonder.time	great.atmospher
come.close	best.kept	time.took	garlic.sauc	wonder.restaur
open.lunch	everyth.menu	waitress.come	great.can	love.atmospher
warm.friend	excel.price	servic.except	absolut.best	bar.just
spoke.manag	keep.come	final.came	place.best	expos.brick
definit.recommend	hot.fresh	new.favorit	year.alway	back.drink
expect.wait	best.mexican	servic.awesom	over.price	fri.noth
great.time	best.sushi	sever.minut	dish.well	great.view
chicken.beef	pizza.best	best.dine	few.place	chicken.good
room.dessert	food.fabul	veri.rude	authent.mexican	bar.great
price.great	melt.mouth	peopl.veri	wether.com	person.favorit
seafood.restaur	each.dish	poor.servic	especi.good	great.decor
friend.atmospher	absolut.wonder	ask.check	like.sit	french.dip
sent.back	foie.gras	real.treat	open.until	pub.food
ll.definit	menu.chang	never.got	great.too	coconut.shrimp
anyon.look	food.bland	non.exist	open.daili	go.up
most.popular	noth.fanci	flag.down	best.valu	servic.fantast
order.wrong	back.time	tabl.ask	just.great	gas.station
delici.food	food.excel	least.minut	fri.littl	pork.loin
fresh.seafood	worth.trip	won.disappoint	portion.huge	place.friend

negative positive

Inverse Regression and Factor Models

Recall our interpretation of Principal Components via Factors

PCA(1) fits $\mathbb{E}[x_{ij}] = \varphi_{j1} v_{i1}, \; j = 1...p$
to minimize deviance over *both* rotations φ and factor v_i's.

1^{st} PC direction for observation i is $z_i = \mathbf{x}_i' \varphi = \sum_{j=1}^{p} \varphi_j x_{ij}$

This is a projection from $\mathbf{x}$ into the space of the unknown v.
Beyond geometry, the easiest interpretation is to think of each principal component z_i as a re-scaled version of the factor v_i.

Here, the factor v_i that you 'project on' is unknown.

Inverse Regression (IR)

In IR, we know use $v_i = y_i$: the 'factor' is our response!
IR φ are thus analogous to PC rotations.

Thus to do inverse regression for text,

- Estimate φ in MN logistic regression for $\mathbf{x}$ on y

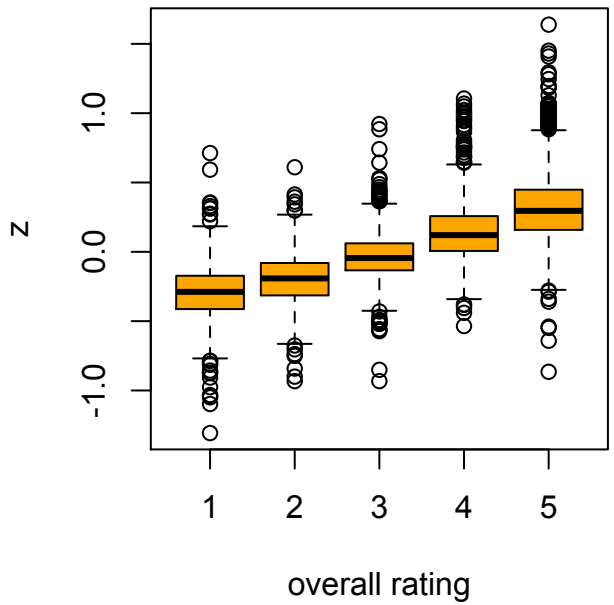
$$\mathbf{x}_i \sim \mathrm{MN}(\mathbf{p}_i, m_i) \text{ with } \mathrm{p}_{ij} = \frac{\exp[\alpha_j + y_i'\varphi_j]}{\sum_{l=1}^{p} \exp[\alpha_l + y_i\varphi_l]}$$

- Project $z_i = \mathbf{f}_i'\varphi$ for all i (i.e., $\mathbf{z} = \mathbf{F}\varphi$)
  Note that we use $\mathbf{f}$ for the projections here.

Prediction is complete with a low-D 'forward regression'.
e.g., $\mathbb{E}[y|z] = \alpha + \beta z$ via OLS, or logistic regression of y on z.

MNIR on the we8there reviews

```
summary( fitwe8 <- mnlm(we8thereCounts, y, bins=5) )
```




```
# first row is intercepts
phi = coef(fitwe8)[2,]
# sum(F[i,]*phi) for each row
z <- F%*%phi
```

```
> summary(fwd <- glm(y~z))
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.50113    0.01343  260.63   <2e-16 ***
z            3.10047    0.03959   78.32   <2e-16 ***
```

So 0.3 increase in $z \Rightarrow$ expected 1 star extra.

Roundup on text mining

People work on language models beyond bag-of-words,
but it has previously proven hard to make big improvements.
The state of the art is to tokenize and model counts.

There are fancy text specific models, but you can go a very long ways using generic data mining methods (esp. CV-lasso).
MN (IR) models are just more efficient (do more with less).

Text information is usually best as part of a larger system.
Use text data to fill in the cracks around what you know.
Don't ignore good variables with stronger signal than text!