Machine Learning for Language Processing
ACS 2015/16

Stephen Clark

L1: Classification

UNIVERSITY OF
CAMBRIDGE

The ML Revolution in NLP

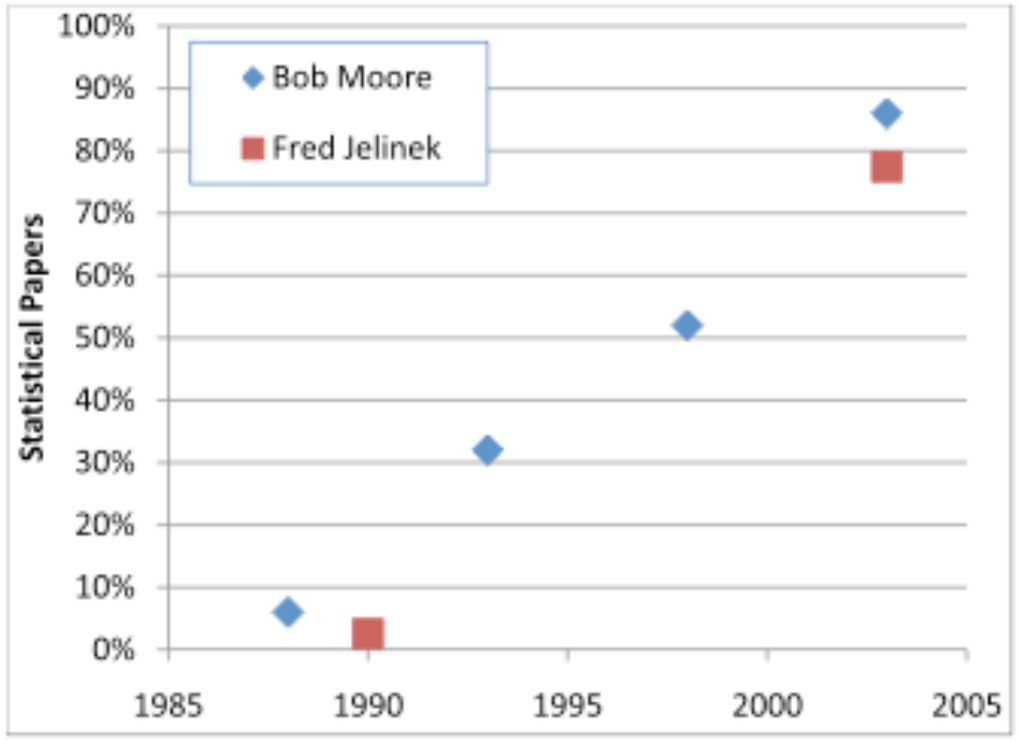


FIGURE 1 The shift from Rationalism to Empiricism is striking (and no longer controversial). This plot is based on two independent surveys of ACL meetings by Bob Moore and Fred Jelinek (personal communication).

Plot from Church (2007)

UNIVERSITY OF CAMBRIDGE

Some History

- 1950s cognitive science dominated by empiricism (Shannon, Skinner, Harris)
- 1960s and 70s dominated by rationalism (Chomsky, Minsky)
- 1990s sees a return to empiricism (IBM speech group, PDP, neural networks)
- 2015: ML/NNs dominant -- with the recognition of the importance of (structured) knowledge?

Statistical NLP not Science?

Original Review of SMT for Coling 1988

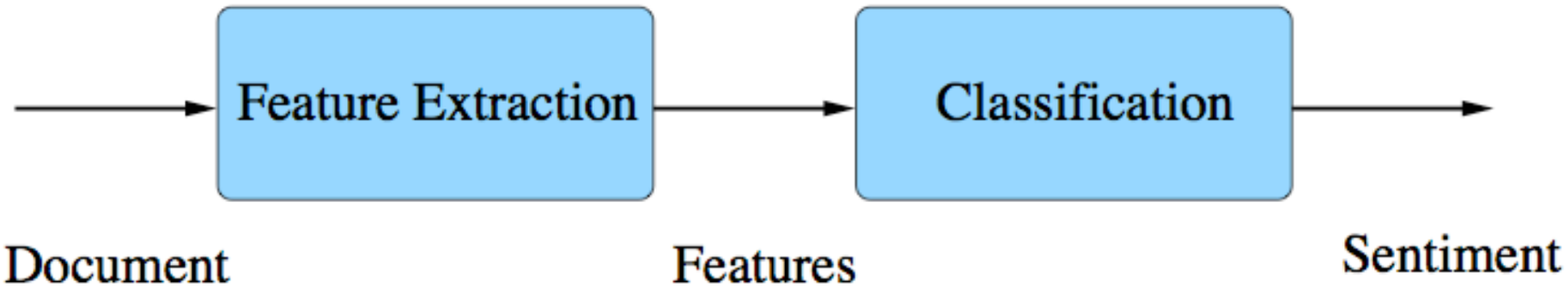
The validity of statistical (information theoretic) approach to MT has indeed been recognized, as the authors mention, by Weaver as early as 1949. And was universally recognized as mistaken by 1950. (cf. Hutchins, MT: Past, Present, Future, Ellis Horwood, 1986, pp 30 ff. and references therein).

The crude force of computers is not science. The paper is simply beyond the scope of COLING.

Text Classification

- Many NLP tasks can be framed as simple classification tasks
- The input is some linguistic unit (word, sentence, document) and the output is a discrete label from some finite set
- Examples include text classification, sentiment analysis, spam detection, ...

Machine Learning Framework



- There are two stages in a pattern recognition framework:
  - feature extraction: a feature vector, $\boldsymbol{x}$, is derived from the "observations";
  - classification: a class ω is identified given the feature vector $\boldsymbol{x}$.
- Example: sentiment analysis
  - $\boldsymbol{w}$ is the document (words)
  - $\boldsymbol{x}$ is a binary vector indicating whether a particular word is in the document
  - ω is the sentiment (e.g. angry)

Training and Test Data

- The basic machine learning framework has two sets of data:

  1. Training data: is used to train the classifier - data may be:
     - supervised: the correct classes of the training data are known
     - unsupervised: the correct classes of the training data are not known
     - reinforcement learning: don't learn a model - directly learn an action!
  2. Test data: held-out data for evaluating the classifier

  Supervised training data will be mostly considered in this course

- It is important that the training and test data do not overlap
  - performance on training data better than on held-out data
  - becomes more important as the classifiers become more complex
  - development data sometimes used to tune parameters
- Aim to build a classifier that performs well on held-out data; generalise.

Machine Learning-based Decisions

- Consider a system where
  - observation: feature vector of dimension d, $\boldsymbol{x}$
  - class labels: there are K classes, denoted by $\omega_1, \omega_2, ..., \omega_K$.
- Classifiers for making decisions can be broadly split as:
  - Generative models: a model of the joint distribution of observations and classes is trained, $P(\boldsymbol{x}, \omega_j)$.
  - Discriminative models: a model of the posterior distribution of the class given the observation is trained, $P(\omega_j|\boldsymbol{x})$.
  - Discriminant functions: a mapping from an observation $\boldsymbol{x}$ to class ω_j is directly trained. No posterior probability, $P(\omega_j|\boldsymbol{x})$, generated just class labels.

Naive Bayes Classifier

Suppose there are K classes, $c_1, c_2, c_3, \ldots, c_K$

e.g. $c_1 = \text{politics}, c_2 = \text{sport}, c_3 = \text{cookery}, \ldots$

$$P(c_j|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|c_j)P(c_j)}{P(\boldsymbol{x})}$$

where $\boldsymbol{x}$ is the feature vector for the input document

$$P(c_j|\boldsymbol{x}) \propto P(\boldsymbol{x}|c_j)P(c_j)$$

Bag of Words Model

$$P(\boldsymbol{x}|c_j) = P(f_1, f_2, f_3, \ldots, f_N|c_j)$$

where f_i is a binary-valued indicator function for ith word

Assume words are conditionally independent given class:

$$P(f_1, f_2, f_3, \ldots, f_N|c_j) = \prod_{i=1}^{N} P(f_i|c_j)$$

Probability Estimation

$$\hat{P}(f_i|c_j) = \frac{\text{freq}(f_i, c_j)}{\text{freq}(c_j)}$$

$$\hat{P}(c_j) = \frac{\text{freq}(c_j)}{number\ of\ docs}$$

relative frequency estimates for this model are *maximum likelihood estimates*

Sentiment Classification

- Pang and Lee started the NLP literature in 2002
- Movie reviews online (IMDb) provide a readily available set of annotated training data
- Classes *positive*, *negative (*752 -ve, 1301 +ve)

Bag of Words Model

	Proposed word lists	Accuracy	Ties
Human 1	positive: *dazzling, brilliant, phenomenal, excellent, fantastic* negative: *suck, terrible, awful, unwatchable, hideous*	58%	75%
Human 2	positive: *gripping, mesmerizing, riveting, spectacular, cool, awesome, thrilling, badass, excellent, moving, exciting* negative: *bad, cliched, sucks, boring, stupid, slow*	64%	39%

Figure 1: Baseline results for human word lists. Data: 700 positive and 700 negative reviews.

	Proposed word lists	Accuracy	Ties
Human 3 + stats	positive: *love, wonderful, best, great, superb, still, beautiful* negative: *bad, worst, stupid, waste, boring, ?, !*	69%	16%

Figure 2: Results for baseline using introspection and simple statistics of the data (including *test* data).

taken from Pang et al. (2002)

Bag of Words Model

$$P_{\mathrm{NB}}(c \mid d) := \frac{P(c)\left(\prod_{i=1}^{m} P(f_i \mid c)^{n_i(d)}\right)}{P(d)}.$$

Our training method consists of relative-frequency estimation of $P(c)$ and $P(f_i \mid c)$, using add-one smoothing.

Despite its simplicity and the fact that its conditional independence assumption clearly does not hold in real-world situations, Naive Bayes-based text categorization still tends to perform surprisingly well

taken from Pang et al. (2002)

Results

	Features	# of features	frequency or presence?	NB	ME	SVM
(1)	unigrams	16165	freq.	**78.7**	N/A	72.8
(2)	unigrams	"	pres.	81.0	80.4	**82.9**
(3)	unigrams+bigrams	32330	pres.	80.6	80.8	**82.7**
(4)	bigrams	16165	pres.	77.3	**77.4**	77.1
(5)	unigrams+POS	16695	pres.	81.5	80.4	**81.9**
(6)	adjectives	2633	pres.	77.0	**77.7**	75.1
(7)	top 2633 unigrams	2633	pres.	80.3	81.0	**81.4**
(8)	unigrams+position	22430	pres.	81.0	80.1	**81.6**

Figure 3: Average three-fold cross-validation accuracies, in percent. Boldface: best performance for a given setting (row). Recall that our baseline results ranged from 50% to 69%.

taken from Pang et al. (2002)

More Recent Work on Sentiment

- Now a very large literature! (and commercial interest)
- See Richard Socher's work using a form of recursive neural network
- Also see TheySay, a recent spin out from Oxford (http://www.theysay.io/)

Machine Learning for Language Processing
ACS 2015/16
Stephen Clark

L2: POS Tagging with HMMs

UNIVERSITY OF CAMBRIDGE

The POS Tagging Problem

```
England|NNP 's|POS fencers|NNS won|VBD gold|NN on|IN
day|NN 4|CD in|IN Delhi|NNP with|IN a|DT medal|JJ
-winning|JJ performance|NN .|.

This|DT is|VBZ Dr.|NNP Black|NNP 's|POS second|JJ
gold|NN of|IN the|DT Games|NNP .|.
```

- Problem is difficult because of ambiguity

Probabilistic Formulation

$$y^* = \arg\max_{y \in Y} P(y|x)$$

where $x = (x_1, \ldots, x_n)$ is a sentence and $y = (y_1, \ldots, y_n) \in Y$ is a possible tag sequence for x

- Two problems:
  - where do the probabilities come from? (age-old question in statistical approaches to AI)
  - how do we find the arg max?
- Problem 1 is the problem of *model estimation*
- Problem 2 is the *search problem*

HMM Tagging Model

- $P(T|W) = \frac{P(W|T)P(T)}{P(W)}$ (Bayes theorem)
- $\arg\max_T P(T|W) = \arg\max_T P(W|T)P(T)$ (W is constant)
- Using Chain Rule and (Markov) independence assumptions:

$$
\begin{aligned}
P(W|T) &= P(w_1, \ldots, w_n | t_1, \ldots, t_n) \\
&= P(w_1|t_1, \ldots, t_n) P(w_2|w_1, t_1, \ldots, t_n) P(w_3|w_2, w_1, t_1, \ldots, t_n) \\
&= P(w_n|w_{n-1}, \ldots, w_1, t_1, \ldots, t_n) \\
&\approx \prod_{i=1}^{n} P(w_i|t_i)
\end{aligned}
$$

$$
\begin{aligned}
P(T) &= P(t_1, \ldots, t_n) \\
&= P(t_1) P(t_2|t_1) P(t_3|t_2, t_1) \ldots P(t_n|t_{n-1}, \ldots, t_1) \\
&\approx \prod_{i=1}^{n} P(t_i|t_{i-1})
\end{aligned}
$$

N-gram Generative Taggers

- A tagger which conditions on the previous tag is called a **bigram** tagger
- Trigram taggers are typically used (condition on previous 2 tags)
- HMM taggers use a **generative** model, so-called because the tags *and* words can be thought of as being generated according to some stochastic process
- More sophisticated **discriminative** models (e.g. CRFs) can condition on more aspects of the context, e.g. suffix information

Parameter Estimation

- Two sets of parameters:
  - $P(t_i|t_{i-1})$ tag transition probabilities
  - $P(w_i|t_i)$ word emission probabilities
- Note *not* $P(t_i|w_i)$ (reversed because of use of Bayes theorem)
  - one of the original papers on stochastic POS tagging reportedly got this wrong
- Estimation based on counting from manually labelled corpora
  - so we have a *supervised* machine learning approach
- For this problem, simple counting (relative frequency) method gives *maximum likelihood* estimates

Relative Frequency Estimation

- $\hat{P}(t_i|t_{i-1}) = \frac{f(t_{i-1},t_i)}{f(t_{i-1})}$
  - where $f(t_{i-1}, t_i)$ is the number of times t_i follows t_{i-1} in the training data; and $f(t_{i-1})$ is the number of times t_{i-1} appears in the data
- $\hat{P}(w_i|t_i) = \frac{f(w_i,t_i)}{f(t_i)}$
  - where $f(w_i, t_i)$ is the number of times w_i has tag t_i in the training data
- It turns out that for an HMM the intuitive relative frequency estimates are the estimates which *maximise the probability of the training data*
- What if the numerator (or denominator) is zero?

Smoothing for Tagging

- Tag sequence probabilities can be smoothed (or backed off):

$$\tilde{P}(t_i|t_{i-1}, t_{i-2}) = \lambda_1 \hat{P}(t_i|t_{i-1}, t_{i-2})$$
$$+(1-\lambda_1)(\lambda_2 \hat{P}(t_i|t_{i-1}) + (1-\lambda_2)\hat{P}(t_i))$$

- A simple solution for unknown words is to replace them with UNK:

$$P(w_i|t_i) = P(\textsf{UNK}|t_i)$$

where any word in the training data occurring less than, say, 5 times is replaced with UNK

Better Handling of Unknown Words

- Lots of clues as to what the tag of an unknown word might be:
  - proper nouns (NNP) likely to be unknown
  - if the word ends in *ing*, likely to be VBG
  - ...

$$P(w|t) = \frac{1}{Z} P(\text{unknown word}|t) P(\text{capitalized}|t) P(\text{endings}|t)$$

- but now we're starting to see the weaknesses of generative models for taggers
- *Conditional* models can deal with these features directly

The Search Problem for Tagging

$$T^* = \arg\max_T P(T|W) = \arg\max_T P(W|T)P(T)$$

- Number of tag sequences for a sentence of length n is $O(T^n)$ where T is the size of the tagset
- OK, but why is there a non-trivial search problem?
  - e.g. for a unigram model we can just take the most probable tag for each word, an algorithm which runs in $O(nT)$ time

A Non-Trivial Search Problem

$$T^* = \arg\max_T P(T|W) = \arg\max_T P(W|T)P(T)$$

- But what about a bigram model?
- Intuition: suppose I have two competing tags for word w_i, t_i^1 and t_i^2
- Compare:

$$\text{Score}(t_i^1) = P(t_i^1|t_{i-1})P(w_i|t_i^1)$$
$$\text{Score}(t_i^2) = P(t_i^2|t_{i-1})P(w_i|t_i^2)$$

Suppose $\text{Score}(t_i^1) > \text{Score}(t_i^2)$; can we be sure t_i^1 is part of the highest scoring tag *sequence*?

The Viterbi Algorithm

- Dynamic Programming (DP) algorithm, so requires the "optimal sub-problem property"
  - i.e. optimal solution to the complete problem can be defined in terms of optimal solutions to sub-problems
- So what are the sub-problems in this case?
  - intuition: suppose we want the optimal tag sequence ending at w_n, and we know the optimal sub-sequence ending at w_{n-1}, for all possible tags at w_{n-1}

Viterbi for a Bigram Tagger

$$\delta_{t_j}(n+1) = \max_{t_i} \delta_{t_i}(n) P(t_j|t_i) P(w_{n+1}|t_j)$$

where $\delta_{t_j}(n+1)$ is the probability of the most probable tag sequence ending in tag t_j at position $n+1$

- Recursion bottoms out at position 1 in the sentence
- Most probable tag sequence can be obtained by following the recursion from the right backwards
- Time complexity is $O(T^2 n)$ where T is the size of the tagset

Practicalities

- Choice of tags to be assigned to a particular word usually governed by a “tag dictionary”
- Accuracy measured by taking a manually created “gold-standard” for a set of held-out test sentences
- Accuracy of POS taggers on newspaper data is over 97%, close to the upper bound represented by human agreement (and existence of noise in the data)
- Linear time process (in length of sentence) means tagging can be performed very fast, e.g. hundreds of thousands of words per second

Machine Learning for Language Processing
ACS 2015/16
Stephen Clark

L3: Maximum Entropy Models

UNIVERSITY OF CAMBRIDGE

Discriminative Models

- Classification requires the class-posterior $P(\omega_j|\boldsymbol{x})$
  - can just directly model the posterior distribution
  - avoids the complexity of modelling the joint distribution $P(\boldsymbol{x}, \omega_j)$

- Form of model called a discriminative model

- Many debates of generative versus discriminative models:
  - discriminative model criterion more closely related to classification process
  - not dependent on generative process being correct
  - joint distribution can be very complicated to accurately model
  - only final posterior distribution needs to be a valid distribution

NER as a Tagging Problem

```
England|I-LOC 's|O fencers|O won|O gold|O on|O
day|I-TIME 4|I-TIME in|O Delhi|I-LOC with|O a|O medal|O
-winning|O performance|O .|O

This|O is|O Prof.|I-PER Black|I-PER 's|O second|O
gold|O of|O the|O Games|O .|O
```

Feature-based Models

- Features encode evidence from the context for a particular tag:

(title caps, `NNP`)	Citibank, Mr.
(suffix `-ing`, `VBG`)	running, cooking
(next word `Inc.`, `I-ORG`)	Lotus Inc.
(previous word `said`, `I-PER`)	said Mr. Vinken

Complex Features

- Features can be arbitrarily complex
  - e.g. document level features
    (document = `cricket` & current word = `Lancashire`, `I-ORG`)
    $\Longrightarrow$ hopefully tag `Lancashire` as `I-ORG` not `I-LOC`

- Features can be combinations of atomic features
  - (current word = `Miss` & next word = `Selfridges`, `I-ORG`)
    $\Longrightarrow$ hopefully tag `Miss` as `I-ORG` not `I-PER`

- Features are not assumed to be (conditionally) independent (given the label)
  – unlike the Naive Bayes classifier

Feature-based Tagging

- How do we incorporate features into a probabilistic tagger?
- Hack the Markov Model tagger to incorporate features
- Maximum Entropy (MaxEnt) Tagging
  - principled way of incorporating features
  - requires sophisticated estimation method

Features in MaxEnt Models

- Features encode elements of the context C useful for predicting tag t

- Features are binary valued functions, e.g.

$$f_i(C,t) = \begin{cases} 1 & \text{if } \texttt{word}(C) = \texttt{Moody} \ \ \& \ \ t = \text{I-ORG} \\ 0 & \text{otherwise} \end{cases}$$

- $\texttt{word}(C) = \texttt{Moody}$ is a *contextual predicate*

- Features determine (`contextual_predicate, tag`) pairs

The Model

$$p(t|C) = \frac{1}{Z(C)} \exp\left(\sum_{i=1}^{n} \lambda_i f_i(C,t)\right)$$

- f_i is a feature
- λ_i is a weight (large value implies informative feature)
- $Z(C)$ is a normalisation constant ensuring a proper probability distribution
- Also known as a *log-linear* model
- Makes no independence assumptions about the features
- Can be used as a general classifer (outside of tagging, e.g. text classification)

Tagging with MaxEnt Models

- The conditional probability of a tag sequence $t_1 \dots t_n$ is

$$p(t_1 \dots t_n | w_1 \dots w_n) \approx \prod_{i=1}^{n} p(t_i | C_i)$$

  given a sentence $w_1 \dots w_n$ and contexts $C_1 \dots C_n$

- The context includes previously assigned tags (for a fixed history)

- Beam search or Viterbi is used to find the most probable sequence (Ratnaparkhi, 1996)

Model Estimation

$$p(t|C) = \frac{1}{Z(C)} \exp\left(\sum_{i=1}^{n} \lambda_i f_i(C,t)\right)$$

- Model estimation involves setting the weight values λ_i
- The model should reflect the data
  $\Longrightarrow$ use the data to *constrain* the model
- What form should the constraints take?
  $\Longrightarrow$ constrain the *expected value* of each feature f_i

The Constraints

$$E_p f_i = \sum_{C,t} p(C,t) f_i(C,t) = K_i$$

- Expected value of each feature must satisfy some constraint K_i
- A natural choice for K_i is the average empirical count:

$$K_i = \quad E_{\tilde{p}} f_i = \frac{1}{N} \sum_{j=1}^{N} f_i(C_j, t_j)$$

derived from the training data $(C_1, t_1), \ldots, (C_N, t_N)$

Choosing the MaxEnt Model

- The constraints do not *uniquely* identify a model
- From those models satisfying the constraints: *choose the Maximum Entropy model*
- Conditional entropy of a model p:

$$H(p) = -\sum_{C,t} \tilde{p}(C)p(t|C)\log p(t|C)$$

The Maximum Entropy Model

- The maximum entropy model is the *most uniform model*
  $\Longrightarrow$ makes no assumptions in addition to what we know from the data

- MaxEnt model is also the *Maximum Likelihood Log-Linear* model

- Set the weights to give the MaxEnt model satisfying the constraints
  $\Longrightarrow$ use *Generalised Iterative Scaling* (GIS)

Generalised Iterative Scaling (GIS)

- Set $\lambda_i^{(0)}$ equal to some arbitrary value (e.g. zero)
- Repeat until convergence:

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} + \frac{1}{C} \log \frac{E_{\tilde{p}} f_i}{E_{p^{(t)}} f_i}$$

where

$$C = \max_{x,y} \sum_{i=1}^{n} f_i(x, y)$$

POS Tagger Features

- The tagger uses binary valued features, e.g.

$$f_i(x, y) = \begin{cases} 1 & \text{if } \texttt{word}(x) = \texttt{the} \ \ \& \ \ y = \text{DT} \\ 0 & \text{otherwise} \end{cases}$$

- $\texttt{word}(\text{x}) = \texttt{the}$ is a *contextual predicate*

- Contextual predicates:

$t_{i-1} = X$	previous tag history
$t_{i-2}t_{i-1} = XY$	previous two tags history
$w_i = X$	current word
$w_{i-1} = X$	previous word
$w_{i-2} = X$	previous previous word
$w_{i+1} = X$	next word
$w_{i+2} = X$	next next word

POS Tagger Features for Rare Words

- These predicates apply to words seen less than 5 times in the data

X is prefix of w_i, $|X| \leq 4$
X is suffix of w_i, $|X| \leq 4$
w_i contains a digit
w_i contains uppercase char
w_i contains a hyphen

- Otherwise the current word predicate applies

Performance

- MaxEnt taggers give close to state-of-the-art accuracy (over 97% on PTB data)
- Training and testing is fast (100s of 1000s of words per second at test time)
- Lots of recent work on tagging other sorts of data, eg tweets
- Recurrent neural networks (probably) give the state-of-the-art for tagging

Contextual Predicates for NER

Condition	Contextual predicate
$f(w_i) < 5$	X is prefix/suffix of w_i, $\lvert X \rvert \leq 4$ w_i contains a digit w_i contains uppercase character w_i contains a hyphen
$\forall w_i$	$w_i = X$ $w_{i-1} = X$, $w_{i-2} = X$ $w_{i+1} = X$, $w_{i+2} = X$
$\forall w_i$	$\mathsf{POS}_i = X$ $\mathsf{POS}_{i-1} = X$, $\mathsf{POS}_{i-2} = X$ $\mathsf{POS}_{i+1} = X$, $\mathsf{POS}_{i+2} = X$
$\forall w_i$	$\mathsf{NE}_{i-1} = X$ $\mathsf{NE}_{i-2}\mathsf{NE}_{i-1} = XY$

Contextual Predicates for NER

Condition	Contextual predicate
$f(w_i) < 5$	w_i contains period
	w_i contains punctuation
	w_i is only digits
	w_i is a number
	w_i is {upper,lower,title,mixed} case
	w_i is alphanumeric
	length of w_i
	w_i has only Roman numerals
	w_i is an initial (X.)
	w_i is an acronym (ABC, A.B.C.)

Contextual Predicates for NER

Condition	Contextual predicate
$\forall w_i$	memory NE tag for w_i unigram tag of w_{i+1} unigram tag of w_{i+2}
$\forall w_i$	w_i in a gazetteer w_{i-1} in a gazetteer w_{i+1} in a gazetteer
$\forall w_i$	w_i not lowercase and $f_{\text{lc}} > f_{\text{uc}}$
$\forall w_i$	unigrams of word type bigrams of word types trigrams of word types

Contextual Predicates for NER

- Moody $\Longrightarrow$ Aa
- A.B.C. $\Longrightarrow$ A.A.A.
- 1,345.00 $\Longrightarrow$ 0,0.0
- Mr. Smith $\Longrightarrow$ Aa. Aa

Machine Learning for Language Processing
ACS 2015/16
Stephen Clark
L4: The Perceptron for Structured Prediction

UNIVERSITY OF CAMBRIDGE

Local and Global Features

- We have already seen local feature representations for the maxent tagger:

$$f_i(C,t) = \begin{cases} 1 & \text{if } \texttt{word}(C) = \texttt{Moody} \ \ \& \ \ t = \text{I-ORG} \\ 0 & \text{otherwise} \end{cases}$$

where C is a context and t a tag

- These can be extended to the whole sequence (as in a CRF):

$$F_i(W,T) = \sum_{j=1}^{n} f_i(C_j, t_j)$$

where W is the sentence and T the tag sequence; C_j is the context for the jth word; t_j is the tag for the jth word

A Linear Tagging Model

$$\text{Score}(T, W) = \sum_i \lambda_i \, F_i(W, T) = \overline{\lambda} \cdot \Phi(W, T)$$

Some notation:

- $\phi : (C, t) \rightarrow \mathbb{R}^d$ where d is the number of features
- $\Phi : (W, T) \rightarrow \mathbb{R}^d$
- $\Phi(W, T) = \sum_j \phi(C_j, t_j)$

ϕ is the local feature vector; Φ is the global feature vector; $\overline{\lambda}$ is the corresponding global weight vector

Decoding for a Global Linear Model

$$T_{\max}(W) \quad = \quad \underset{T}{\operatorname{argmax}} \sum_i \lambda_i \, F_i(W, T)$$

- Viterbi finds this argmax efficiently (assuming each contex C_j contains only the previous m tags; i.e. assuming an m-gram tagger)

The Perceptron Training Algorithm

Inputs: Training examples (x_k, y_k)
Initialization: $\overline{\lambda} = 0$
Algorithm:
For $l = 1$ to L, $k = 1$ to n
Use Viterbi to get $z_k = \text{argmax}_z \, \overline{\lambda} \cdot \Phi(x_k, z)$
If $z_k \neq y_k$ then $\overline{\lambda} = \overline{\lambda} + \Phi(x_k, y_k) - \Phi(x_k, z_k)$
Output: weights $\overline{\lambda}$

The Training Algorithm (with words)

Inputs: Training examples (x_k, y_k)
Initialization: $\overline{\lambda} = 0$
Algorithm:
  For $l = 1$ to L, $k = 1$ to n
    Use Viterbi to get $z_k = \text{argmax}_z \, \overline{\lambda} \cdot \Phi(x_k, z)$
    If $z_k \neq y_k$ then $\overline{\lambda} = \overline{\lambda} + \Phi(x_k, y_k) - \Phi(x_k, z_k)$
Output: weights $\overline{\lambda}$

Assume n tagged sentences for training
Initialise weights to zero
Do L passes over the training data
For each tagged sentence in the training data, find the highest scoring tag sequence using the current weights
If the highest scoring tag sequence matches the gold, move to next sentence
If not, for each feature in the gold but not in the output, add 1 to its weight; for each feature in the output but not in the gold, take 1 from its weight
Return weights

Averaging Parameters

Perceptron training can suffer from over-fitting; averaging the parameters is a simple addition which works well in practice:

$$\lambda_i^{av} = \sum_{l=1 \text{ to } L,\ k=1 \text{ to } n} \lambda_i^{l,k}/Ln$$

(Based on the voted perceptron, which has some theory associated with it)

Theory of the Perceptron

Simple additive update seems intuitive, but do we have any guarantees? Collins (2002) has some proofs showing that:

- If the data is separable with some margin, then the algorithm will converge on weights which give zero error on the training data
- If the training data is not separable, but "close" to being separable, then the algorithm will make a small number of mistakes (on the training data)
- If the algorithm makes a small number of errors on the training data, it is likely to generalise well to unseen data

A *Ranking* Perceptron

Some notation:

- Assume training data $\{(s_i, t_i)\}$ (e.g. s_i is a sentence and t_i the correct tree for s_i)
- $\mathbf{x}_{ij}$ is the jth candidate for example i (e.g. the jth tree for sentence i)
- Assume (w.l.o.g.) that $\mathbf{x}_{i1}$ is the correct output for input s_i (i.e. $\mathbf{x}_{i1} = t_i$)
- $\mathbf{h}(\mathbf{x}_{ij}) \in \mathbb{R}^d$ is the feature vector for $\mathbf{x}_{ij}$
- $\mathbf{w} \in \mathbb{R}^d$ is the corresponding weight vector
- Output of the model on example s (train or test) is $\text{argmax}_{\mathbf{x} \in \mathcal{C}(s)} \mathbf{w} \cdot \mathbf{h}(\mathbf{x})$
- $\mathcal{C}(s)$ is the set of candidate outputs for input s

Training (with the new notation)

Define:
$F(\mathbf{x}) = \mathbf{w} \cdot \mathbf{h}(\mathbf{x})$
Initialisation: Set parameters $\mathbf{w} = 0$
For $i = 1$ to n
    $j = \text{argmax}_{\{1,\ldots,n_i\}} F(\mathbf{x}_{ij})$
    If $j \neq 1$ **then** $\mathbf{w} = \mathbf{w} + \mathbf{h}(\mathbf{x}_{i1}) - \mathbf{h}(\mathbf{x}_{ij})$
Output on test sentence s**:**
    $\text{argmax}_{\mathbf{x} \in \mathcal{C}(s)} F(\mathbf{x})$

- For simplicity, only showing one pass over the data and no averaging

- The argmax can be obtained just though enumeration (i.e. we have a *ranking* problem, so no need for dynamic programming)

Perceptron Training (a duel form)

Define:

$G(\mathbf{x}) = \sum_{(i,j)} \alpha_{i,j}(\mathbf{h}(\mathbf{x}_{i1}) \cdot \mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{x}_{ij}) \cdot \mathbf{h}(\mathbf{x}))$

Initialisation: Set dual parameters $\alpha_{i,j} = 0$

For $i = 1$ to n

  $j = \text{argmax}_{\{1,\ldots,n_i\}} G(\mathbf{x}_{ij})$

  If $j \neq 1$ **then** $\alpha_{i,j} = \alpha_{i,j} + 1$

Output on test sentence s**:**

  $\text{argmax}_{\mathbf{x} \in \mathcal{C}(s)} G(\mathbf{x})$

- Notice there is a dual parameter $\alpha_{i,j}$ for each training example $\mathbf{x}_{i,j}$

Equivalence of the two Forms

- $\mathbf{w} = \sum_{(i,j)} \alpha_{i,j}(\mathbf{h}(\mathbf{x}_{i1}) - \mathbf{h}(\mathbf{x}_{ij}))$; therefore $G(\mathbf{x}) = F(\mathbf{x})$ throughout training
- Why is this useful? Consider the complexity of the two algorithms

Complexity of the two Forms

- Assume T is the size of the training set; i.e. $T = \sum_i n_i$
- Take d to be the size of the parameter vector $\mathbf{w}$
- Vanilla perceptron takes $O(Td)$ time (time taken to compute F is $O(d)$)
- Assume time taken to compute the inner product between examples is k
- Running time of the dual-form perceptron is $O(Tnk)$
- Dual-form is therefore more efficient when $nk << d$ (i.e. when time taken to compute inner products between examples is much less than $O(d)$)

Complexity of Inner Products

- Can the time to calculate the inner product between two examples $\mathbf{h}(\mathbf{x}) \cdot \mathbf{h}(\mathbf{y})$ ever be less than $O(d)$?

- Yes! For certain high-dimensional feature representations

- Examples include feature representations which track all sub-trees in a tree, or all sub-sequences in a tag sequence

Tree Kernels

- Tree kernels count the numbers of shared subtrees between trees $\mathcal{T}_1$ and $\mathcal{T}_2$
  - the feature-space, $\mathbf{h}(\mathcal{T}_1)$, can be defined as

$$\mathbf{h}_i(\mathcal{T}_1) = \sum_{n \in \mathcal{V}_1} I_i(n); \quad I_i(n) = \begin{cases} 1 & \text{if sub-tree } i \text{ rooted at node } n \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{V}_j$ is the set of nodes in tree $\mathcal{T}_j$

Computation of Subtree Kernel

- Can be made computationally efficient by recursively using a counting function:

$$k(\mathcal{T}_1, \mathcal{T}_2) = \mathbf{h}(\mathcal{T}_1)^\top \mathbf{h}(\mathcal{T}_2) = \sum_{n_1 \in \mathcal{V}_1} \sum_{n_2 \in \mathcal{V}_2} f(n_1, n_2);$$

  - if productions from n_1 and n_2 differ $f(n_1, n_2) = 0$
  - for pre-terminals $f(n_1, n_2) = \begin{cases} 1 & \text{if productions are the same} \\ 0 & \text{otherwise} \end{cases}$
  - for non-pre-terminals and productions the same
    $f(n_1, n_2) = \prod_{i=1}^{|\mathsf{ch}(n_1)|} (1 + f(\mathsf{ch}(n_1, i), \mathsf{ch}(n_2, i)))$

where $\mathsf{ch}(n_j)$ is the set of children of n_j and $\mathsf{ch}(n_j, i)$ is the ith child of n_j

- Algorithm runs in linear time w.r.t. the size of each tree

Machine Learning for Language Processing
ACS 2015/16
Stephen Clark

L5: Topic Modelling and LDA

UNIVERSITY OF CAMBRIDGE

Probabilistic Topic Modelling

- We want to find *themes* (or *topics*) in documents
  - useful for e.g. search or browsing
- We don't want to do supervised topic classification
  - rather not fix topics in advance nor do manual annotation
- Need an approach which automatically teases out the topics
- This is essentially a *clustering* problem - can think of both words and documents as being clustered

Key Assumptions behind LDA

- Documents exhibit multiple topics (but typically not many)
- LDA is a probabilistic model with a corresponding *generative process*
  - each document is assumed to be generated by this (simple) process
- A *topic* is a distribution over a fixed vocabulary
  - these topics are assumed to be generated first, before the documents
- Only the number of topics is specified in advance

Example Topics

human	evolution	disease	computer
genome	evolutionary	host	models
dna	species	bacteria	information
genetic	organisms	diseases	data
genes	life	resistance	computers
sequence	origin	bacterial	system
gene	biology	new	network
molecular	groups	strains	systems
sequencing	phylogenetic	control	model
map	living	infectious	parallel
information	diversity	malaria	methods
genetics	group	parasite	networks
mapping	new	parasites	software
project	two	united	new
sequences	common	tuberculosis	simulations

Taken from Blei 2012 ICML tutorial

Documents and Topics

Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

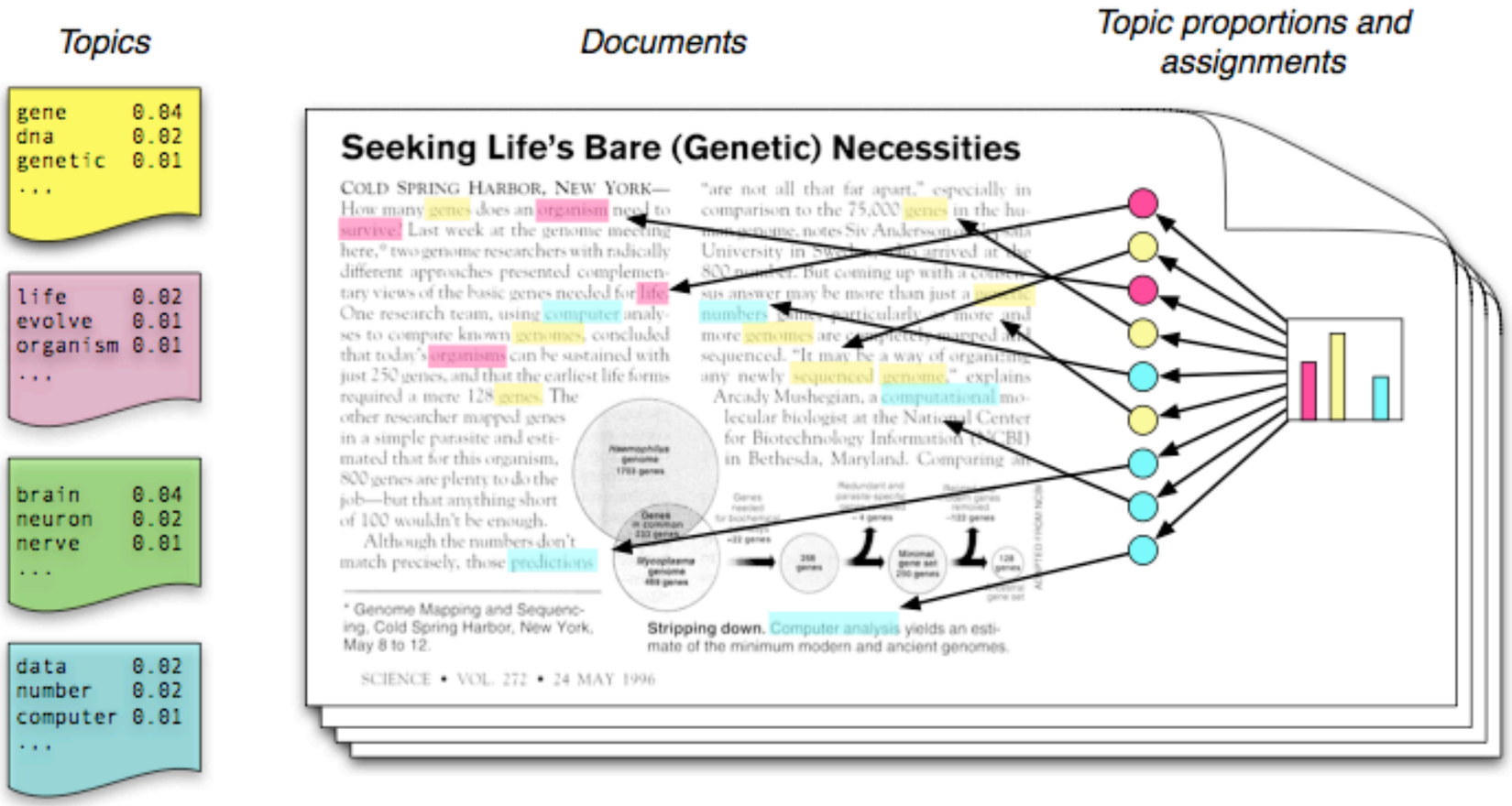
Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

Taken from Blei 2012 ICML tutorial

Documents and Topics



- Each **topic** is a distribution over words
- Each **document** is a mixture of corpus-wide topics
- Each **word** is drawn from one of those topics

Taken from Blei 2012 ICML tutorial

UNIVERSITY OF CAMBRIDGE

The Generative Process

To generate a document:

1. Randomly choose a distribution over topics

2. For each word in the document
   a. randomly choose a topic from the distribution over topics
   b. randomly choose a word from the corresponding topic (distribution over the vocabulary)

- Note that we need a distribution over a distribution (for step 1)

- Note that words are generated independently of other words (unigram bag-of-words model)

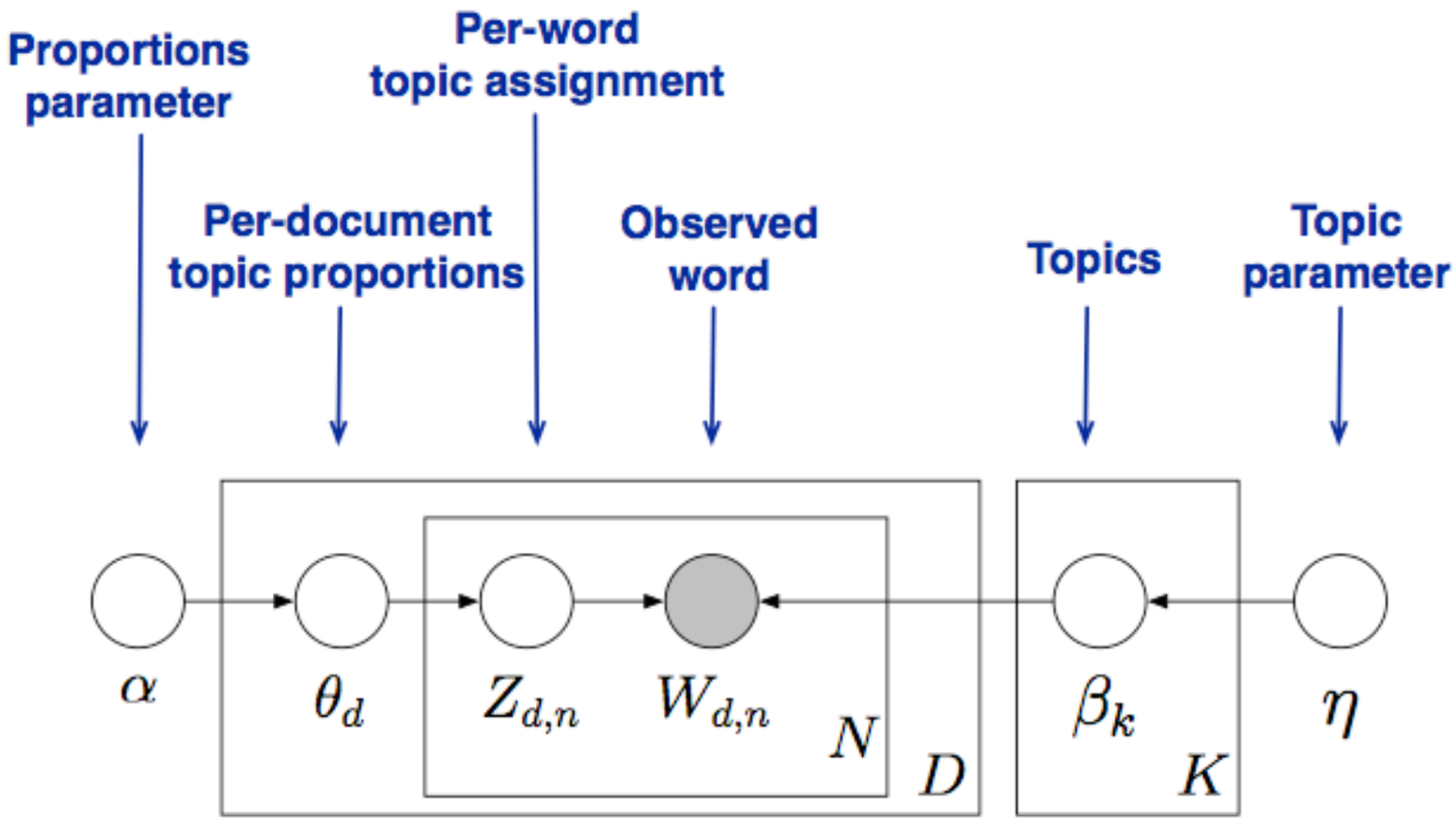
The (Formal) Generative Process

- Some notation:
  - $\beta_{1:K}$ are the topics where each β_k is a distribution over the vocabulary
  - θ_d are the topic proportions for document d
  - $\theta_{d,k}$ is the topic proportion for topic k in document d
  - z_d are the topic assignments for document d
  - $z_{d,n}$ is the topic assignment for word n in document d
  - w_d are the observed words for document d

- The joint distribution (of the hidden and observed variables):

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{i=1}^{K} p(\beta_i) \prod_{d=1}^{D} p(\theta_d) \prod_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\beta_{1:K}, z_{d,n})$$

LDA as a Graphical Model



$$\prod_{i=1}^{K} p(\beta_i \mid \eta) \prod_{d=1}^{D} p(\theta_d \mid \alpha) \left(\prod_{n=1}^{N} p(z_{d,n} \mid \theta_d) p(w_{d,n} \mid \beta_{1:K}, z_{d,n}) \right)$$

Taken from Blei 2012 ICML tutorial

The Dirichlet Distribution

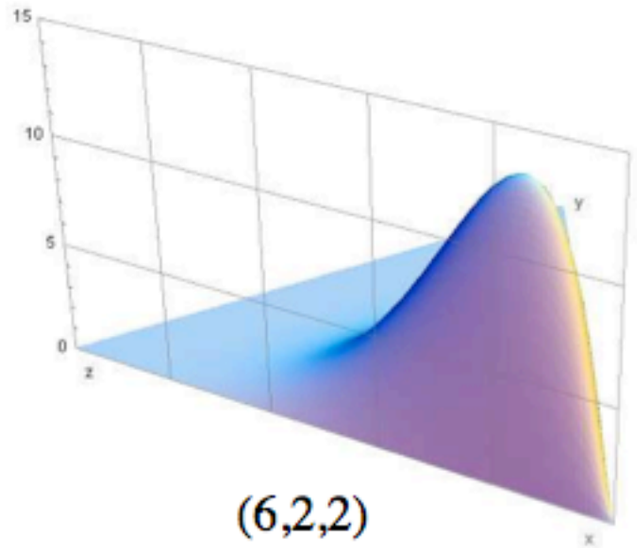
- **Dirichlet** (continuous) distribution with parameters $\boldsymbol{\alpha}$

$$p(\boldsymbol{x}|\boldsymbol{\alpha}) = \frac{\Gamma(\sum_{i=1}^{d}\alpha_i)}{\prod_{i=1}^{d}\Gamma(\alpha_i)}\prod_{i=1}^{d}x_i^{\alpha_i-1}; \quad \text{for "observations"}: \sum_{i=1}^{d}x_i = 1, \quad x_i \geq 0$$

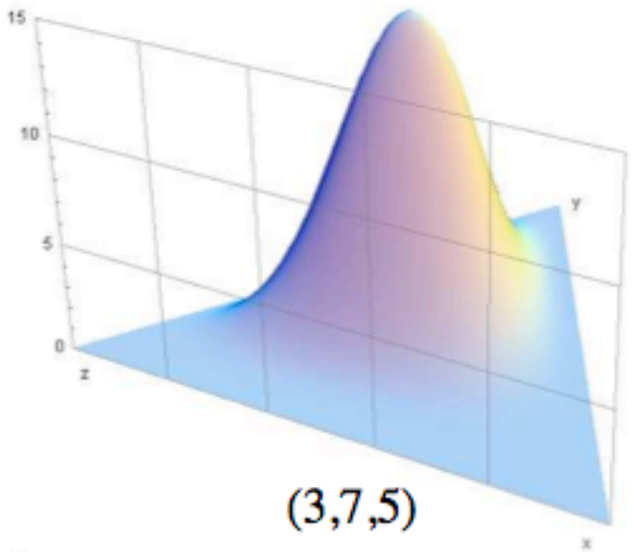
- $\Gamma()$ is the Gamma distribution

- Conjugate prior to the multinomial distribution
  (form of posterior $p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M})$ is the same as the prior $p(\boldsymbol{\theta}|\mathcal{M})$)

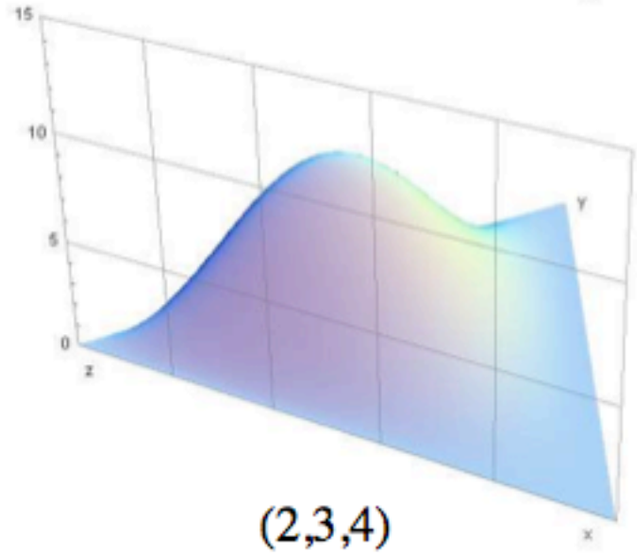
The Dirichlet Distribution



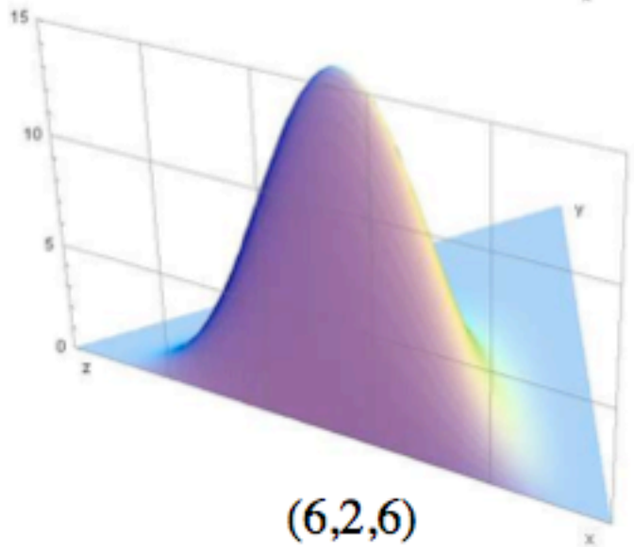
(6,2,2)



(3,7,5)



(2,3,4)



(6,2,6)

- Parameters: $(\alpha_1, \alpha_2, \alpha_3)$

UNIVERSITY OF CAMBRIDGE

Parameter Estimation

- Main variables of interest:
  - β_k: distribution over vocabulary for topic k
  - $\theta_{d,k}$: topic proportion for topic k in document d

- Could try and get these directly, eg using EM (Hoffmann, 1999), but this approach not very successful

- One common technique is to estimate the posterior of the word-topic assignments, given the observed words, directly (whilst marginalizing out β and θ)

- Gibbs sampling is an example of a Markov Chain Monte Carlo (MCMC) technique

Estimates using Gibbs Sampling

- The Gibbs sampler produces the following estimate, where, following Steyvers and Griffiths:
  - z_i is the topic assigned to the ith token in the whole collection;
  - d_i is the document containing the ith token;
  - w_i is the word type of the ith token;
  - $\mathbf{z}_{-i}$ is the set of topic assignments of all other tokens;
  - $\cdot$ is any remaining information such as the α and η hyperparameters:

$$P(z_i = j|\mathbf{z}_{-i}, w_i, d_i, \cdot) \propto \frac{C^{WT}_{w_i j} + \eta}{\sum_{w=1}^{W} C^{WT}_{wj} + W\eta} \frac{C^{DT}_{d_i j} + \alpha}{\sum_{t=1}^{T} C^{DT}_{d_i t} + T\alpha}$$

where $\mathbf{C}^{WT}$ and $\mathbf{C}^{DT}$ are matrices of counts (word-topic and document-topic)

The Final Estimates

$$\beta_{ij} = \frac{C_{ij}^{WT} + \eta}{\sum_{k=1}^{W} C_{kj}^{WT} + W\eta} \quad \theta_{dj} = \frac{C_{dj}^{DT} + \alpha}{\sum_{k=1}^{T} C_{dk}^{DT} + T\alpha}$$

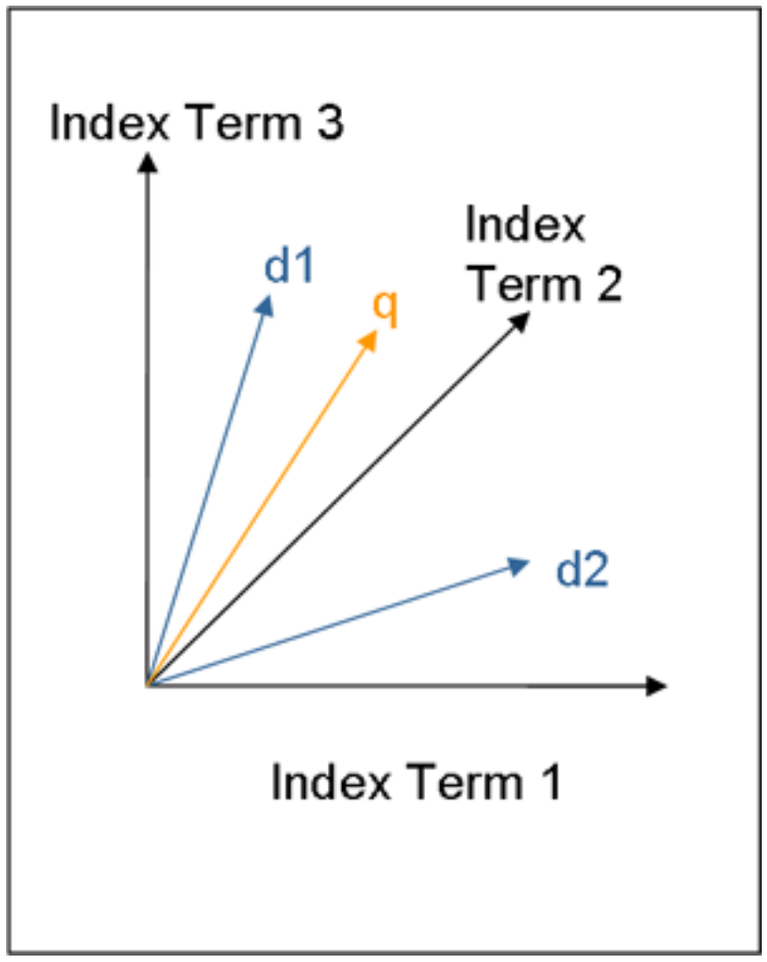
- Using the count matrices as before, where β_{ij} is the probability of word type i for topic j, and θ_{dj} is the proportion of topic j in document d

Machine Learning for Language Processing
ACS 2015/16
Stephen Clark

L6: Vector Space Models of Semantics

UNIVERSITY OF CAMBRIDGE

VSMs in Document Retrieval



UNIVERSITY OF CAMBRIDGE

Term-Frequency Model

Term vocabulary: ⟨*England, Australia, Pietersen, Hoggard, run, wicket, catch, century, collapse*⟩

Document $d1$: *Australia collapsed as Hoggard took 6 wickets . Flintoff praised Hoggard for his excellent line and length .*

Document $d2$: *Flintoff took the wicket of Australia 's Ponting , to give him 2 wickets for the innings and 5 wickets for the match .*

Query q: { *Hoggard, Australia, wickets* }

$$\vec{q1} \cdot \vec{d1} = \langle 0,1,0,1,0,1,0,0,0 \rangle \cdot \langle 0,1,0,2,0,1,0,0,1 \rangle = 4$$

$$\vec{q1} \cdot \vec{d2} = \langle 0,1,0,1,0,1,0,0,0 \rangle \cdot \langle 0,1,0,0,0,3,0,0,0 \rangle = 4$$

Figure 1. Simple example of document and query similarity using the dot product, with term-frequency providing the vector coefficients. The documents have been tokenised, and word matching is performed between lemmas (so *wickets* matches *wicket*).

TF-IDF Model

Term vocabulary: ⟨*England, Australia, Pietersen, Hoggard, run, wicket, catch, century, collapse*⟩

Document $d1$: *Australia collapsed as Hoggard took 6 wickets . Flintoff praised Hoggard for his excellent line and length .*

Document $d2$: *Flintoff took the wicket of Australia 's Ponting , to give him 2 wickets for the innings and 5 wickets for the match .*

Query q: { *Hoggard, Australia, wickets* }

$$\vec{q1} \cdot \vec{d1} = \langle 0, 1, 0, 1, 0, 1, 0, 0, 0 \rangle \cdot \langle 0, 1/10, 0, 2/5, 0, 1/100, 0, 0, 1/3 \rangle = 0.41$$

$$\vec{q1} \cdot \vec{d2} = \langle 0, 1, 0, 1, 0, 1, 0, 0, 0 \rangle \cdot \langle 0, 1/10, 0, 0/5, 0, 3/100, 0, 0, 0/3 \rangle = 0.13$$

Figure 2. Simple example of document and query similarity using the dot product, with term-frequency, inverse-document frequency providing the coefficients for the documents, using the same query and documents as Figure 1.

TF-IDF Model

$$\begin{aligned}\text{Sim}(\vec{d}, \vec{q}) &= \frac{\vec{d}.\vec{q}}{\|\vec{d}\|\|\vec{q}\|} \\ &= \frac{\vec{d}.\vec{q}}{\sqrt{\sum_i d_i^2}\sqrt{\sum_i q_i^2}} \\ &= \text{Cosine}(\vec{d}, \vec{q})\end{aligned}$$

Document and query length normalisation in combination with the dot product gives the cosine similarity measure

Term-Document Matrix

	$d1$	$d2$
England	0	0
Australia	1/10	1/10
Pietersen	0	0
Hoggard	2/5	0/5
run	0	0
wicket	1/100	3/100
catch	0	0
century	0	0
collapse	1/3	0/3

Figure 3. Term-document matrix for the simple running example, using *tf-idf* weights but without length normalisation.

Term-Document Matrix

Terms	Documents								
	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

Documents are similar if they tend to contain the same (informative) terms

Terms are similar if they tend to occur in the same documents

A Finer Notion of Context

An automobile is a wheeled motor vehicle used for transporting passengers .
A car is a form of transport, usually with four wheels and the capacity to carry around five passengers .
Transport for the London games is limited , with spectators strongly advised to avoid the use of cars .
The London 2012 soccer tournament began yesterday , with plenty of goals in the opening matches .
Giggs scored the first goal of the football tournament at Wembley , North London .
Bellamy was largely a passenger in the football match , playing no part in either goal .

Term vocab: $\langle$*wheel, transport, passenger, tournament, London, goal, match*$\rangle$

	wheel	*transport*	*passenger*	*tournament*	*London*	*goal*	*match*
automobile	1	1	1	0	0	0	0
car	1	2	1	0	1	0	0
soccer	0	0	0	1	1	1	1
football	0	0	1	1	1	2	1

$$\textit{automobile} \,.\, \textit{car} = 4$$
$$\textit{automobile} \,.\, \textit{soccer} = 0$$
$$\textit{automobile} \,.\, \textit{football} = 1$$
$$\textit{car} \,.\, \textit{soccer} = 1$$
$$\textit{car} \,.\, \textit{football} = 2$$
$$\textit{soccer} \,.\, \textit{football} = 5$$

UNIVERSITY OF CAMBRIDGE

Alternative Definitions of Context

Giggs|NNP *scored*|VBD *the*|DT *first*|JJ *goal*|NN *of*|IN *the*|DT *football*|NN
tournament|NN *at*|IN *Wembley*|NNP ,|, *North*|NNP *London*|NNP .|.

(`ncmod` _ *goal first*)
(`det` *goal the*)
(`ncmod` _ *tournament football*)
(`det` *tournament the*)
(`ncmod` _ *London North*)
(`dobj` *at Wembley*)
(`ncmod` _ *scored at*)
(`dobj` *of tournament*)
(`ncmod` _ *goal of*)
(`dobj` *scored goal*)
(`ncsubj` *scored Giggs* _)

Contextual elements for target word *goal* using a 7-word window method:
{*scored, the, first, of, football*}

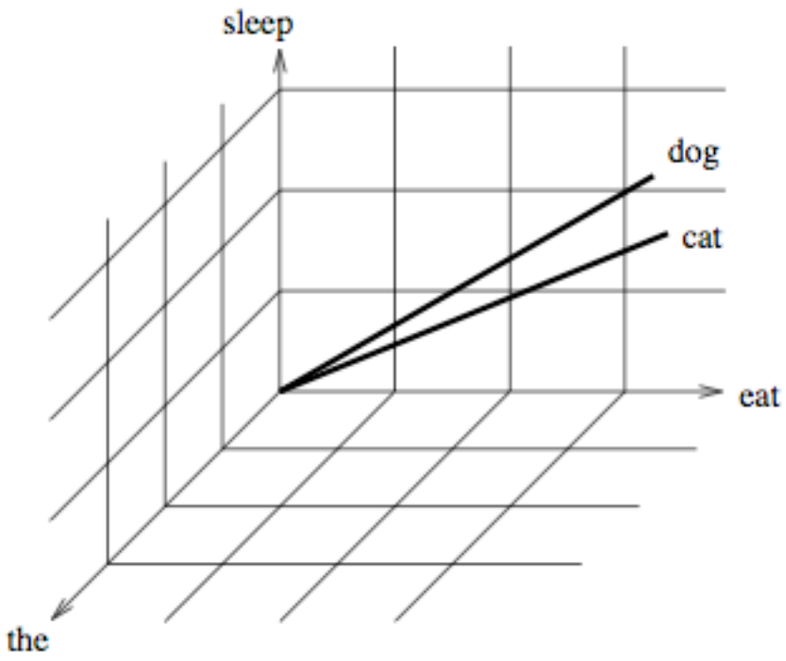
Contextual elements with parts-of-speech:
{*scored*|VBD, *the*|DET, *first*|JJ, *of*|IN, *football*|NN}

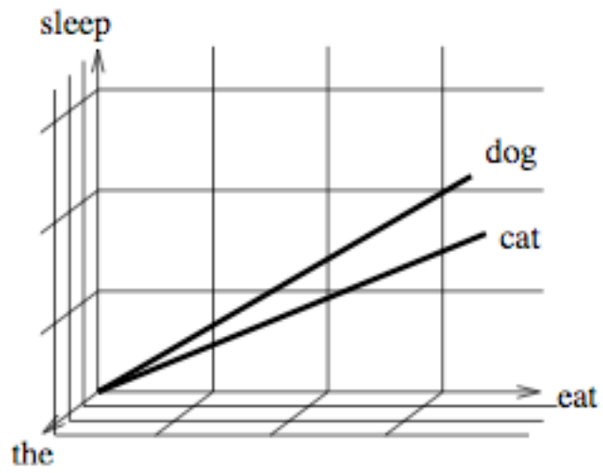
Contextual elements with direction (L for left, R for right):
{*scored*|L, *the*|L, *first*|L, *of*|R, *the*|R, *football*|R}

Contextual elements with position (e.g. 1L is 1 word to the left):
{*scored*|3L, *the*|2L, *first*|1L, *of*|1R, *the*|2R, *football*|3R}

Contextual elements as grammatical relations:
{*first*|`ncmod`, *the*|`det`, *scored*|`dobj`}

Weighting





The effect of IDF on a simple example vector space.

Similarity and Relatedness Datasets

love	sex	6.77
tiger	cat	7.35
tiger	tiger	10.00
computer	internet	7.58
plane	car	5.77
doctor	nurse	7.00
professor	doctor	6.62
smart	stupid	5.81
stock	phone	1.62

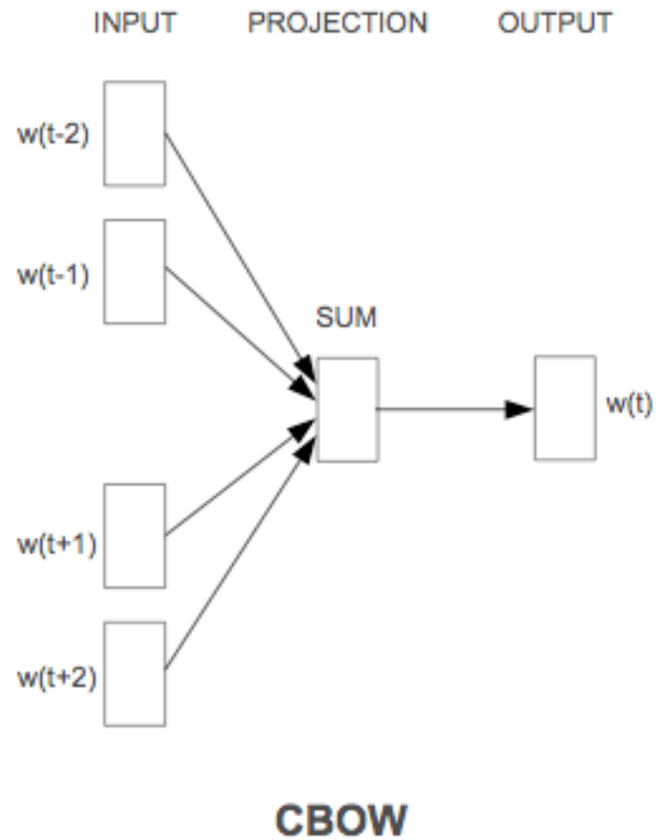
Some example human similarity/relatedness judgements from wordsim 353

Machine Learning for Language Processing
ACS 2015/16
Stephen Clark

L7: Word Embeddings

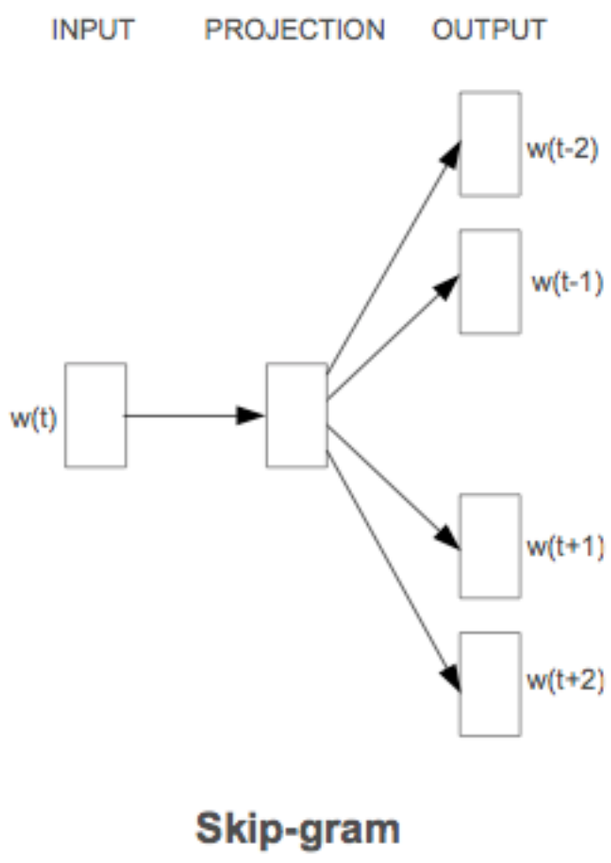
UNIVERSITY OF CAMBRIDGE

Neural Distributional Models



Continuous bag of words model, from Mikolov et al. 2013

Neural Distributional Models



Skip-gram model; picture taken from Mikolov et al. 2013

Skip-Gram "Language Modelling"

$$\arg\max_{\theta} \prod_{w \in Text} \prod_{c \in C(w)} p(c|w;\theta)$$

where $C(w)$ is the set of contexts for each word w

$$\arg\max_{\theta} \prod_{(w,c) \in D} p(c|w;\theta)$$

where D is the set of word, context pairs

Parameterisation of Skip-Gram

$$p(c|w,\theta) = \frac{e^{v_c \cdot v_w}}{\sum_{c' \in C} e^{v_{c'} \cdot v_w}}$$

where v_c and $v_w \in R^d$ are vector representations for c and w and C is the set of all possible contexts

Negative Sampling

$$\arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} p(D=0|c,w;\theta)$$

$$= \arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} (1-p(D=1|c,w;\theta))$$

$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log\, p(D=1|c,w;\theta) + \sum_{(w,c)\in D'} \log\, (1-p(D=1|c,w;\theta))$$

where $D=1$ when (c,w) is from the data and $D=0$ when not and D' is a set of negative word, context pairs

Negative Sampling

$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c \cdot v_w}} + \sum_{(w,c)\in D'} \log \left(1 - \frac{1}{1+e^{-v_c \cdot v_w}}\right)$$

$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c \cdot v_w}} + \sum_{(w,c)\in D'} \log \left(\frac{1}{1+e^{v_c \cdot v_w}}\right)$$

$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log \sigma(v_c \cdot v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c \cdot v_w)$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$

Sampling Details

For each $(w, c) \in D$ we construct k samples $(w, c_1), \ldots (w, c_k)$ where each c_j is sampled from the unigram distribution$^{\frac{3}{4}}$

The contexts are taken from a window of size N around the target word: $w_{i-N}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+N}$ where N is sampled uniformly between 1 and N for each word

words appearing less than M times are discarded

Linguistic Regularities?

$$\overrightarrow{\text{KING}} - \overrightarrow{\text{MAN}} + \overrightarrow{\text{WOMAN}} = \overrightarrow{\text{QUEEN}}$$

Taken from Mikolov et al. 2013

UNIVERSITY OF CAMBRIDGE

Evaluation

- Semantic Relatedness

love	sex	6.77
tiger	cat	7.35
tiger	tiger	10.00
computer	internet	7.58
plane	car	5.77
doctor	nurse	7.00
professor	doctor	6.62
smart	stupid	5.81
stock	phone	1.62

Baroni et al., *Don't count, predict!*

UNIVERSITY OF CAMBRIDGE

Evaluation

- Synonym Detection (TOEFL)

You will find the office at the main **intersection**.
(a) place (b) crossroads (c) roundabout (d) building

Baroni et al., *Don't count, predict!*

Evaluation

- Concept Categorization

Concept categorization Given a set of nominal concepts, the task is to group them into natural categories (e.g., *helicopters* and *motorcycles* should go to the *vehicle* class, *dogs* and *elephants* into the *mammal* class). Following previous art, we tackle categorization as an unsupervised clustering task.

Baroni et al., *Don't count, predict!*

UNIVERSITY OF CAMBRIDGE

Evaluation

- Selectional Preferences

Selectional preferences We experiment with two data sets that contain verb-noun pairs that were rated by subjects for the typicality of the noun as a subject or object of the verb (e.g., *people* received a high average score as subject of *to eat*, and a low score as object of the same

Baroni et al., *Don't count, predict!*

UNIVERSITY OF CAMBRIDGE

Evaluation

- Analogy

Analogy While all the previous data sets are relatively standard in the DSM field to test traditional count models, our last benchmark was introduced in Mikolov et al. (2013a) specifically to test predict models. The data-set contains about 9K semantic and 10.5K syntactic analogy questions. A semantic question gives an example pair (*brother-sister*), a test word (*grandson*) and asks to find another word that instantiates the relation illustrated by the example with respect to the test word (*granddaughter*). A syntactic question is similar, but in this case the relationship is of a grammatical nature (*work–works*, *speak… speaks*). Mikolov

Baroni et al., *Don't count, predict!*

UNIVERSITY OF CAMBRIDGE

Results

- Baroni et al. report very strong results for the “predict” over the “count” vectors
- But see Levy and Goldberg (NIPS, 2014) for a more nuanced picture

Baroni et al., *Don’t count, predict!*