ECS763P
Natural Language Processing

Week 4
Unsupervised Methods & Latent Variable Models

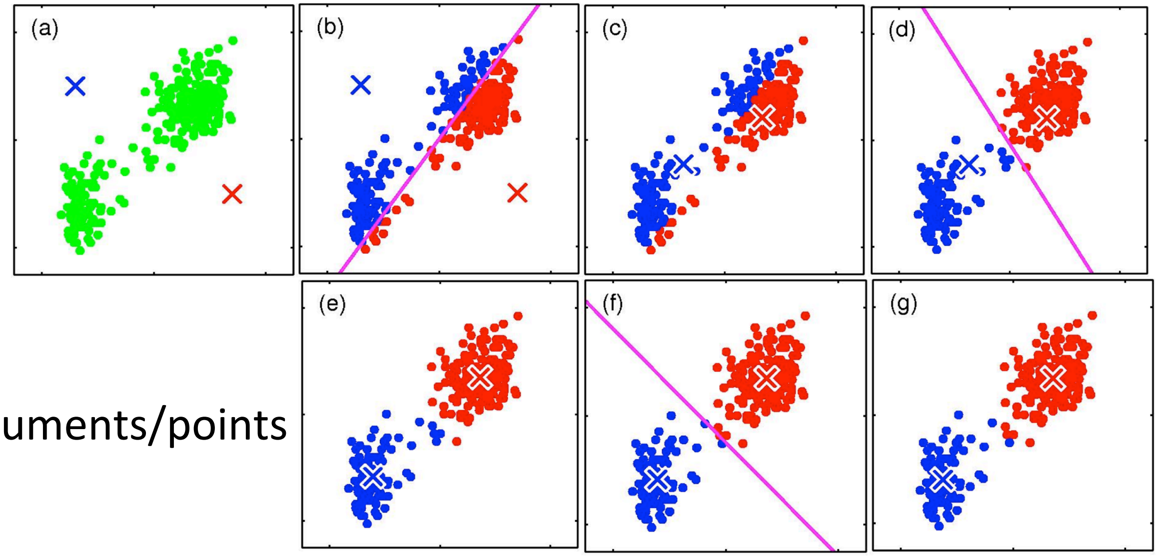
Matthew Purver
(with material from Jurafsky & Martin)

Unsupervised Methods

- So far our tasks have been **supervised** or **one-class**:
  - Text classification (into known classes)
  - Sequence modelling (with known labels)
  - Language modelling (for one class of "language")
- Many problems are about discovering classes or distinctions we don't know in advance:
  - Document clustering
  - Word sense disambiguation
  - Grammar induction
  - Language acquisition
  - Topic modelling
  - Unsupervised tagging (POS, DA, …)
  - …

Document Clustering

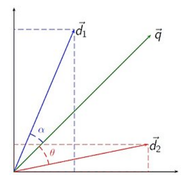
- Simplest approach is k-means
  - (remember this from Data Mining)



- Algorithm pseudocode:
  - Initialise with k random documents/points as centroids
  - **Expectation** step:
    - For all documents, assign to closest centroid
    - Set of documents sharing a centroid is now a cluster
  - **Maximisation** step:
    - For all clusters, re-calculate centroid
  - Repeat until converged

K-means Clustering

- What do we need?
  - The number of clusters k
    - More advanced algorithms can choose this, see e.g. x-means
  - A suitable distance metric
    - e.g. cosine distance between bag-of-words vectors



$$d_1 = \langle 0.2, 0.8, \ldots \rangle$$

$$d_2 = \langle 0.8, 0.1, \ldots \rangle$$

$$\cos(d_1, d_2) = \frac{d_1 \bullet d_2}{\|d_1\| \|d_2\|}$$

$$d_2 = \frac{0.2 * 0.8 + 0.8 * 0.1 + \ldots}{\sqrt{0.2^2 + 0.8^2 + \ldots} * \sqrt{0.8^2 + 0.1^2 + \ldots}}$$

    - Scaling can be important: normalisation, weights
    - e.g. TF-IDF weighting (see Information Retrieval lectures)
      - TF = term frequency = frequency of term in this document
        usually smoothed, log-weighted e.g. $tf_t = 1 + \log(f_t)$
      - IDF = inverse document frequency = rarity of term across collection
        usually smoothed, log-weighted e.g. $idf_t = \log(\frac{N}{1 + df_t})$
  - Some luck with initialisation
    - No guarantee of finding a global minimum!
    - Often worth trying multiple different runs

Expectation-Maximisation

- K-means is a special example of a very common general approach
  - **Expectation-Maximisation (EM)**
  - Useful whenever we have two unknowns
    - e.g. unknown labels Q, unknown model parameters Θ
    - This is very often the case! E.g. part-of-speech tagging, parsing, …

- Alternate steps:
  - E: estimate labels based on current parameters

$$\hat{Q}^t = \underset{Q}{\operatorname{argmax}}\, F(Q, \hat{\Theta}^t)$$

    - (for k-means: estimate labels given current centroid distances)
  - M: estimate parameters based on current labels

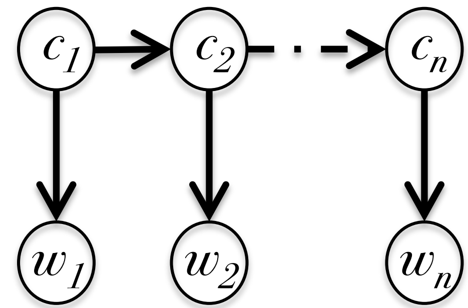
$$\hat{\Theta}^{t+1} = \underset{\Theta}{\operatorname{argmax}}\, F(\hat{Q}^t, \Theta)$$

    - (for k-means: estimate centroid distances given current labels)

- But:
  - No guarantee of finding a global minimum, only local
  - Sensitive to initial conditions, learning parameters

Brown Clustering

- Remember class-based language models:
  - words generated from **classes**
  - ideally LOCATION, PERSON etc
  - what if we don't know the classes?
    - (and/or don't have labelled data)

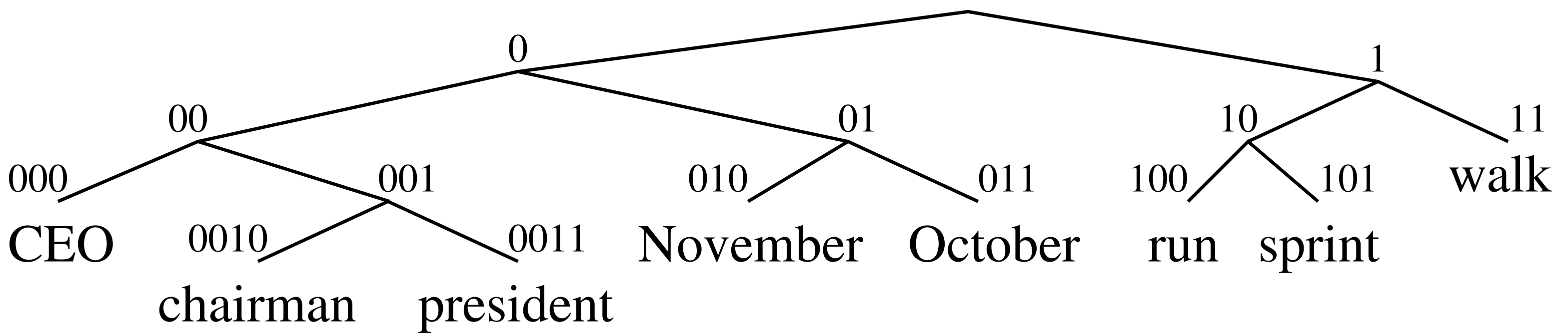


$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid c_i) P(c_i \mid c_{i-1})$$

- Start with *V* words in *V* classes
  - E step: merge two classes which give smallest decrease in $P(w_1 \ldots w_n)$
  - M step: re-estimate class language model parameters
- Either:
  - Stop when you have desired number of classes *C*
  - Carry on until all are merged into one big cluster …

Brown Clustering

- Order of merging clusters gives a tree:



- An added bonus: word vectors!
  - Represent each word by the binary string from root to leaf
    - ‘chairman’ = 0010, ‘president’ = 0011, ‘sprint’ = 101
  - Now we have meaningful word representations
    - Similarity between words ≈ vector similarity

An Aside: Lexical Semantics

- Until now, we've treated words (or n-grams etc) as discrete symbols …
  - Distinct features in classifier models
  - Distinct symbols in sequence models
- … and we'll do a lot of this in formal grammars too

  `mary hires a detective`

  → $\exists x.detective'(x)\ \&\ hire'(mary',x)$
- But words themselves have meaning!
  - They can be synonyms, antonyms, hypernyms, hyponyms

    `good, bad; hot, cold; animal, bird, penguin`
  - They can entail each other

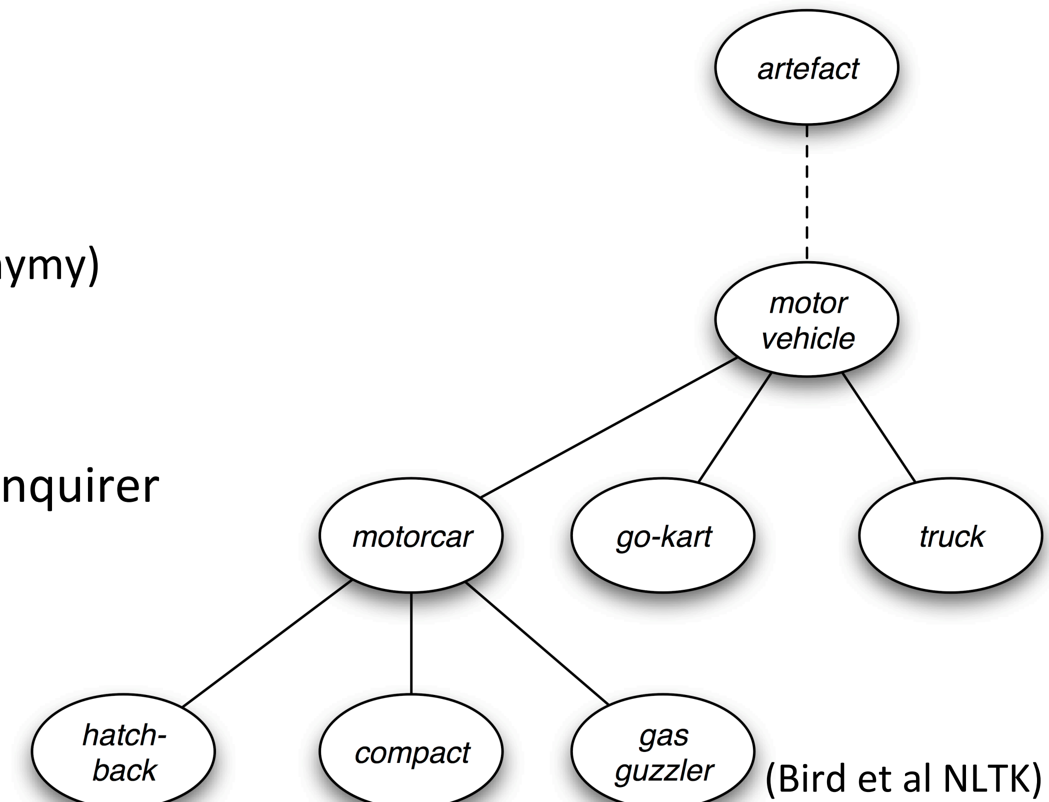
    `bachelor = unmarried, male, human, …`
  - They can be more or less similar to each other

    `good, bad, terrific, great, awful, terrible`
- We are often going to want to know this …

Lexical Semantics: Ontologies

- Some useful ontology resources exist
  - Good for specific relations
  - Limited, but high-quality

- You might use:
  - WordNet
    - (thesaurus, hypo/hypernymy)
  - VerbNet / FrameNet
    - (verbs & arguments)
  - SentiWordNet / General Enquirer
    - (sentiment & emotions)



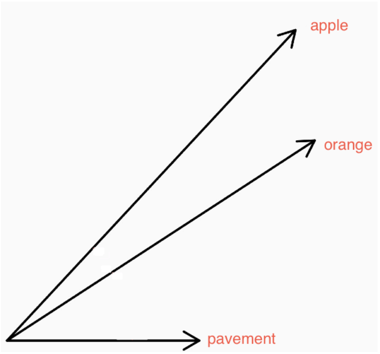
(Bird et al NLTK)

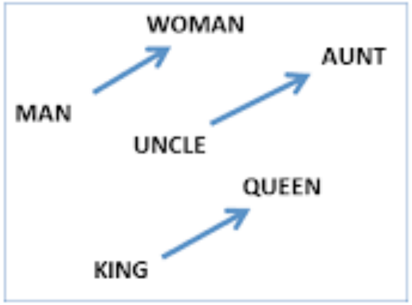
Lexical Semantics: Vectors

- Most state-of-the-art NLP now uses vector representations
  - Geometric notions of similarity, negation, entailment etc
  - "distributional semantics" – see later lectures
- Based on language modelling!
  - Why does this work?

    `fill the X, empty the X, …`

    - high probability $p(X|w_{1..n}) \approx$ similar meaning

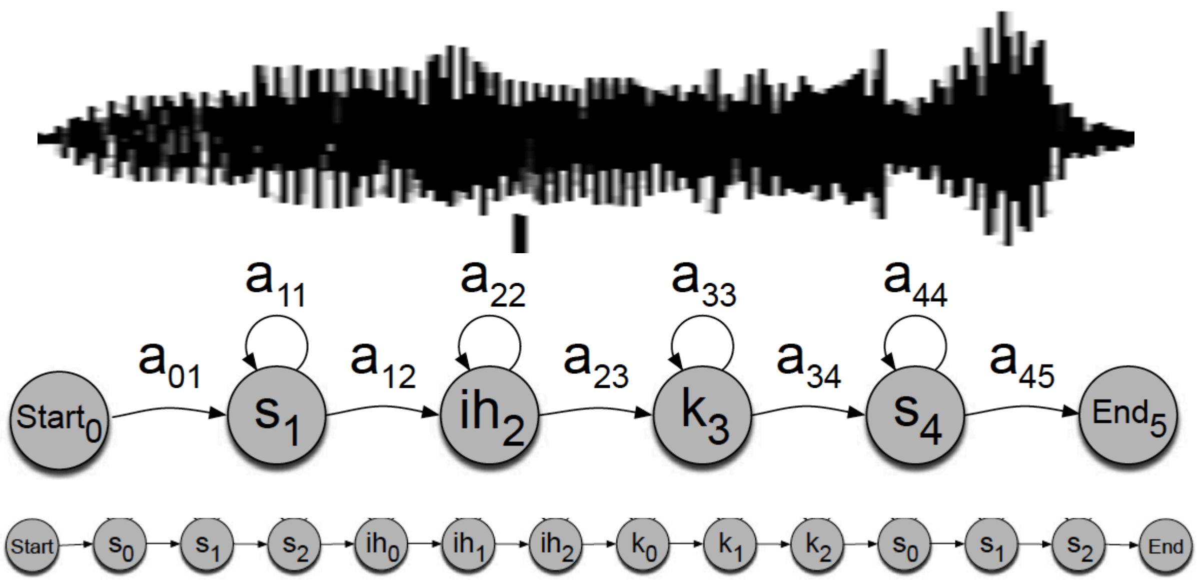




- You might use:
  - Brown clusters (Brown et al 1992)
  - word2vec (Mikolov et al 2013)
  - GloVe (Pennington et al 2014)
  - (pre-trained vectors available for each)

HMM Learning

- We know how to learn HMM models from fully labelled data
- But we usually don’t have fully labelled data
  - Why?
- Speech recognition
  - e.g. a phoneme HMM for the word “six”
  - Observations are acoustic features: what do we align it with & how?



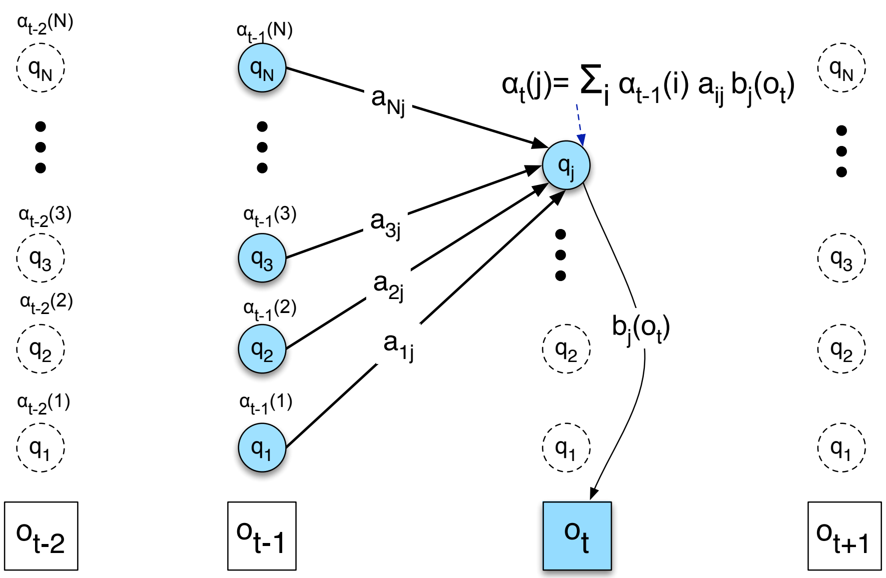
HMM Learning

- Speech recognition
  - In fact, we usually don't even know where words start & end:

I want a flight from Boston to Denver

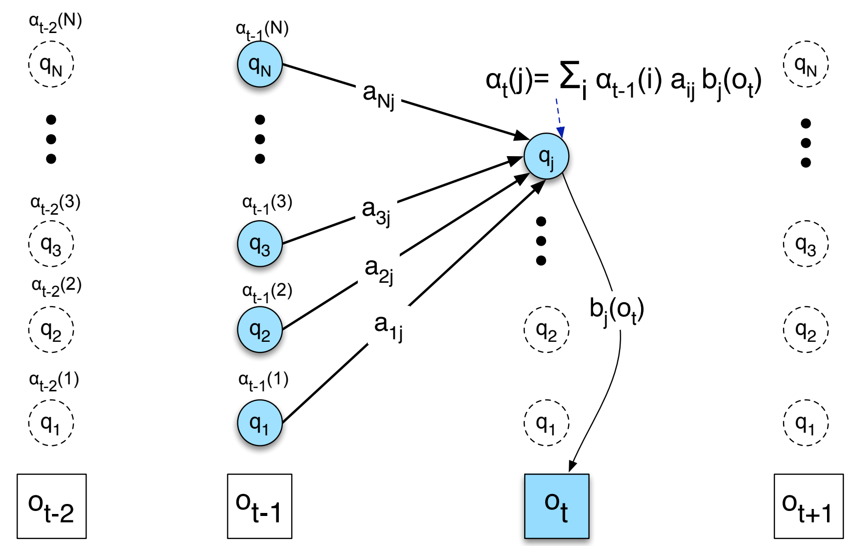
HMM Learning as EM

- So we have two unknowns:
  - State sequence (HMMs are **latent variable** models)
  - Model parameters (transition & emission probabilities A,B)
- Use EM to learn
  - E step:
    - find expected state occupancy and transition counts given A, B
  - M step:
    - re-estimate A, B given estimated counts
- Do this using dynamic programming:
  - Remember Forward algorithm?

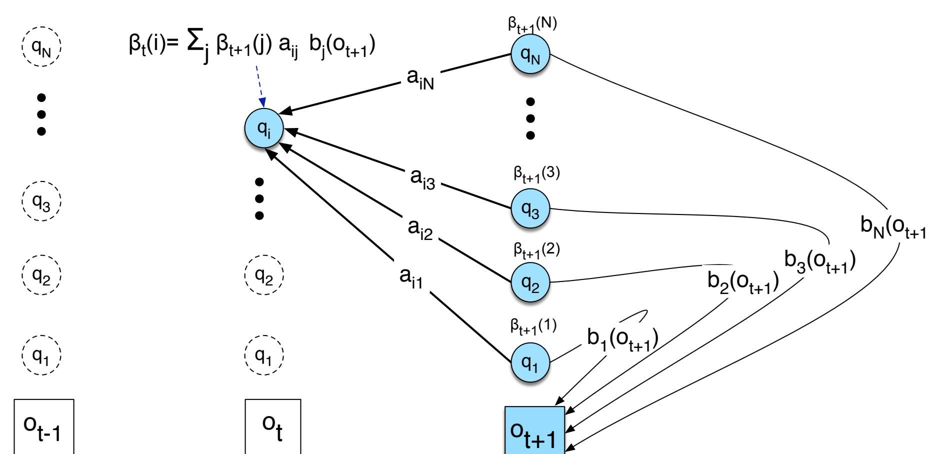


Forward & Backward Probabilities

- "Forward" probability
  - $\alpha_t(j)$ = probability of getting to word t and being in state j
    (having observed words $o_1 \ldots o_t$)



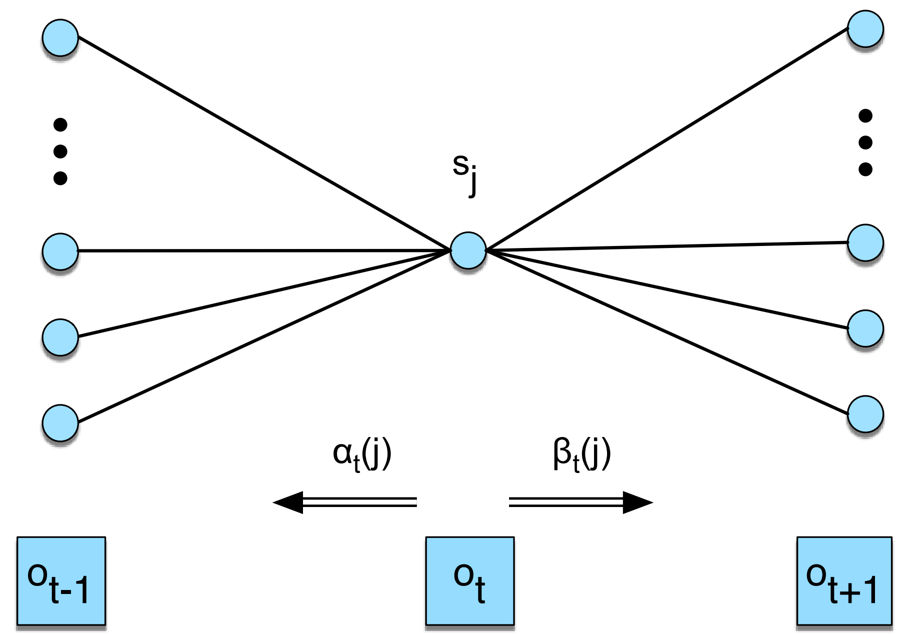
- "Backward" probability
  - $\beta_t(i)$ = probability of getting from word t in state i to the end
    - (observing words $o_{t+1} \ldots o_N$)



HMM Learning

- We can combine them in the Forward-Backward (Baum-Welch) algorithm

  - Emission probabilities:
    - expected number of times seeing word t in state j, from: $p(o_t, s_j) = \alpha_t(j)\beta_t(j)$
    - sum over time for each word type, normalise over sum for all word types

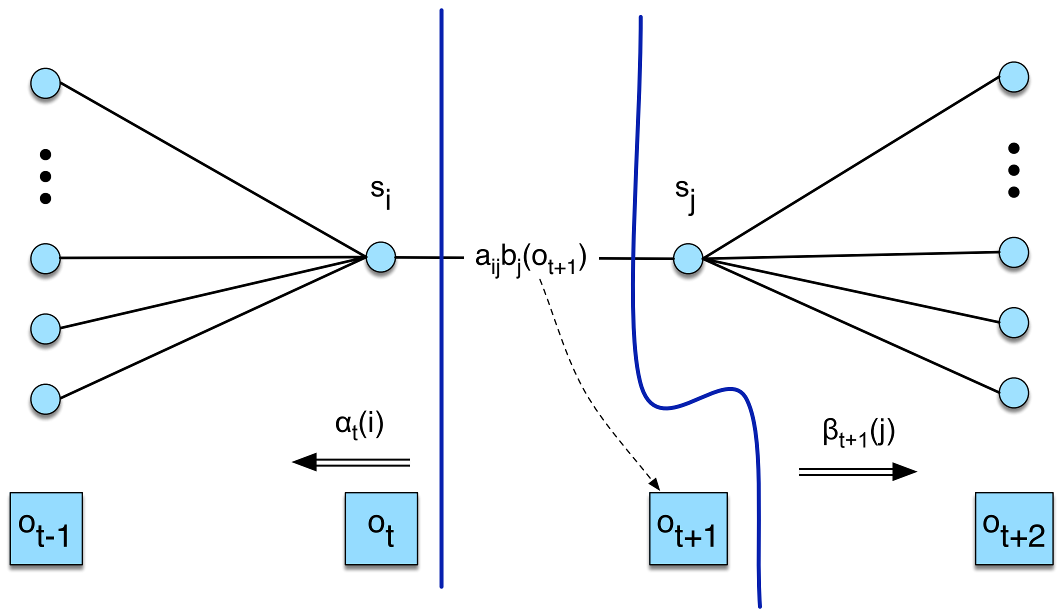


HMM Learning

- We can combine them in the Forward-Backward (Baum-Welch) algorithm
  - Transition probabilities:
    - expected number of times going state i at time t to state j at time t+1, from:

$$p(o_{t+1}, s_i \rightarrow s_j) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

    - sum over time for each state j, normalise over sum for all states



Grammar Induction

- How do we estimate rule probabilities for a grammar?
  - e.g. S → NP VP NP → DET N
  - If we have a manually annotated treebank, we can count: n(NP → DET N)/n(NP)
  - But of course, usually we don't …
  - ... and parsing is a very ambiguous process: many possible parses for any sentence and grammar
- Use EM!
  - Initial estimate of rule probabilities
  - E step: parse all sentences, count successful appearances
  - M step: re-estimate rule probabilities
  - Efficient: the **Inside-Outside algorithm**
- We can even do this to learn the grammar:
  - Invent possible rules
  - E-M to re-estimate rule probabilities, prune unlikely rules

Incremental Grammar Induction

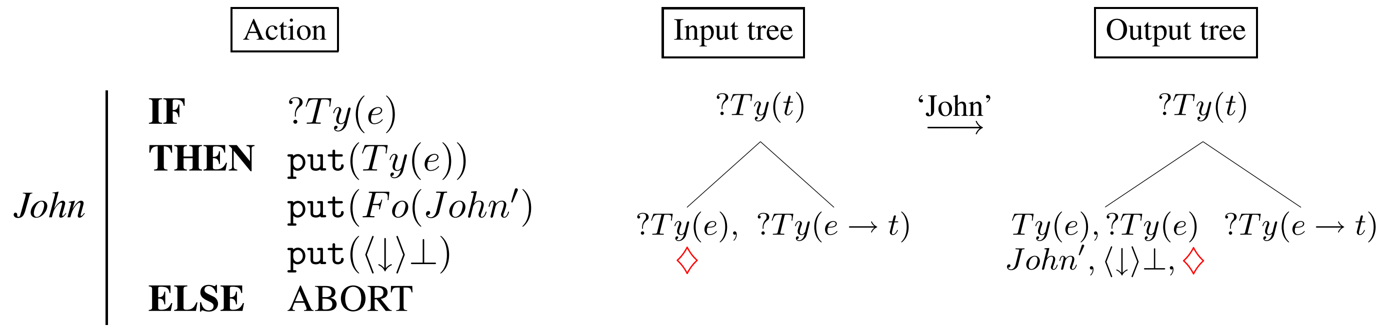


Figure 2: Lexical action for the word ‘John’

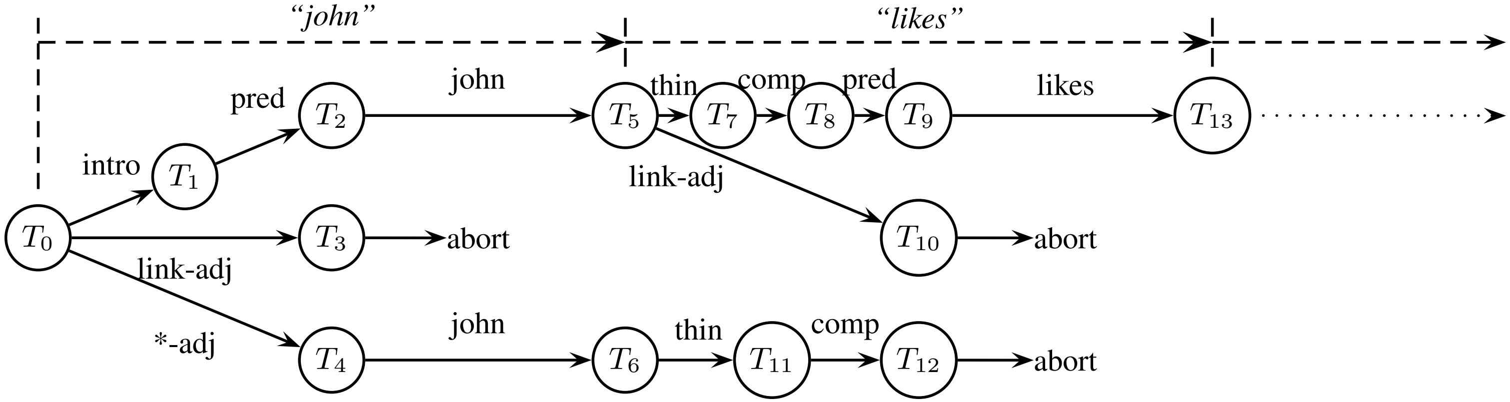
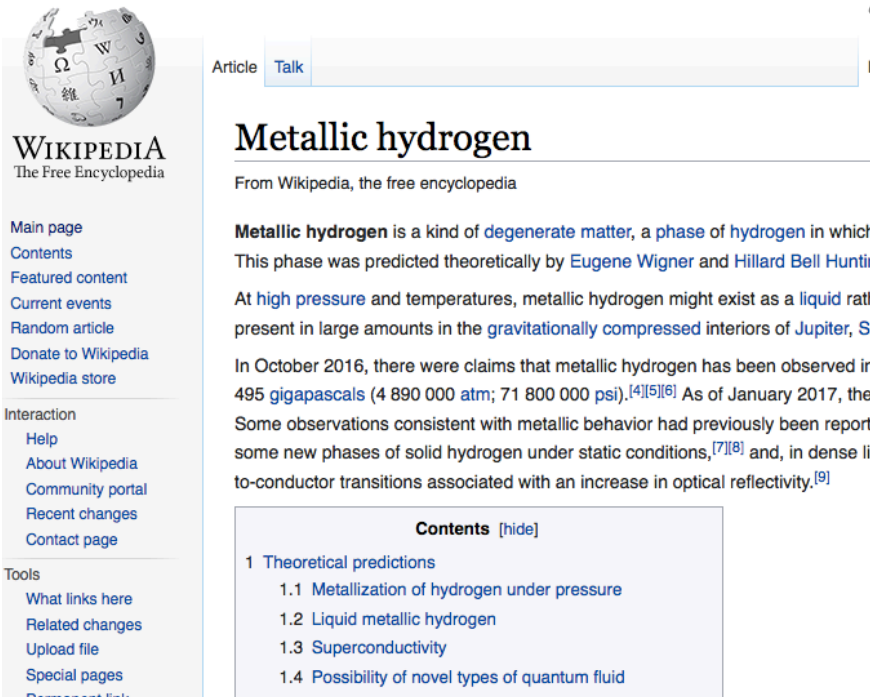


Figure 3: DS parsing as a graph: actions (edges) are transitions between partial trees (nodes).

Eshghi et al, CMCL 2013

Topic Modelling

- "What is this document about?"
  - "what is this section about?"
  - "what is this part of this broadcast about?"
  - etc.



- How can we approach this?
  - ~~Most common words~~
  - Most "characteristic" words? (IR)
  - Language model differences?
- But: we'd like something transferable
  - i.e. track topics between documents
- And we'd like some short description
  - i.e. characterise topicality via reduced dimensions

Latent Semantic Analysis

- We already know about dimensionality reduction
  - (remember PCA from Data Mining)
- We can find low-rank approximations to matrices (Deerwester et al, 1988)
  - “matrix factorisation”: usually Singular Value Decomposition (SVD)
- Factorise a term-document matrix X into eigenvectors & eigenvalues :

$$
\begin{matrix} & X & & & U & & \Sigma & & V^T \\ & (\mathbf{d}_j) & & & & & & & (\hat{\mathbf{d}}_j) \\ & \downarrow & & & & & & & \downarrow \\
(\mathbf{t}_i^T) \rightarrow & \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix} & = & (\hat{\mathbf{t}}_i^T) \rightarrow & \begin{bmatrix} \begin{bmatrix} \, \\ \mathbf{u}_1 \\ \, \end{bmatrix} \dots \begin{bmatrix} \, \\ \mathbf{u}_l \\ \, \end{bmatrix} \end{bmatrix} & \cdot & \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{bmatrix} & \cdot & \begin{bmatrix} [& \mathbf{v}_1 &] \\ & \vdots & \\ [& \mathbf{v}_l &] \end{bmatrix} \end{matrix}
$$

en.wikipedia.org

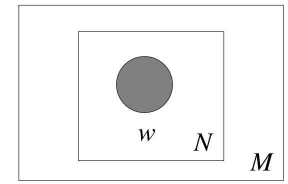
- Keep the *k* vectors with highest eigenvalues as “topics”
  - Latent dimensions which explain the variance
- This gives a good model for measuring topical similarity
  - Often more robust than word-based metrics
  - But it’s not very meaningful (how can we understand the “topics”?)

Latent Dirichlet Allocation

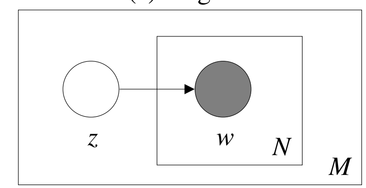
- We can build a probabilistic version
  - (but just language modelling isn't enough – why?)
- Generative model (Hoffman, 1999):
  - "probabilistic Latent Semantic Indexing" (PLSI)
  - Assume latent topic variable z
    - Each topic is a word distribution
  - Model document as **mixture** of topics

$$p(w \mid d) = p(w \mid z)p(z \mid d)$$

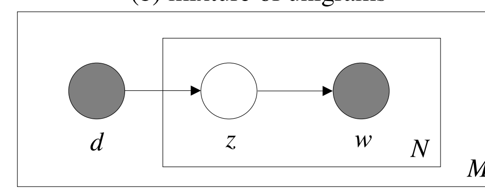
  - Estimate using EM again
- Gives us interpretable topics!
  - But: no way of applying to a new document



(a) unigram

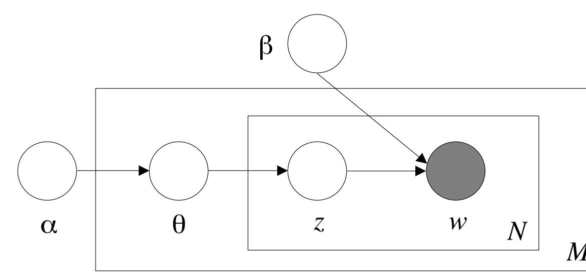


(b) mixture of unigrams



(c) pLSI/aspect model

- Latent Dirichlet Allocation (LDA, Blei et al. 2003)
  - Fully generative Bayesian model
  - Assume document-topic mixtures drawn from an underlying distribution
    - (and similarly for topic-word distributions)
  - Estimate using a variant of EM (e.g. Gibbs Sampling)



(Blei et al., 2003)

Latent Dirichlet Allocation

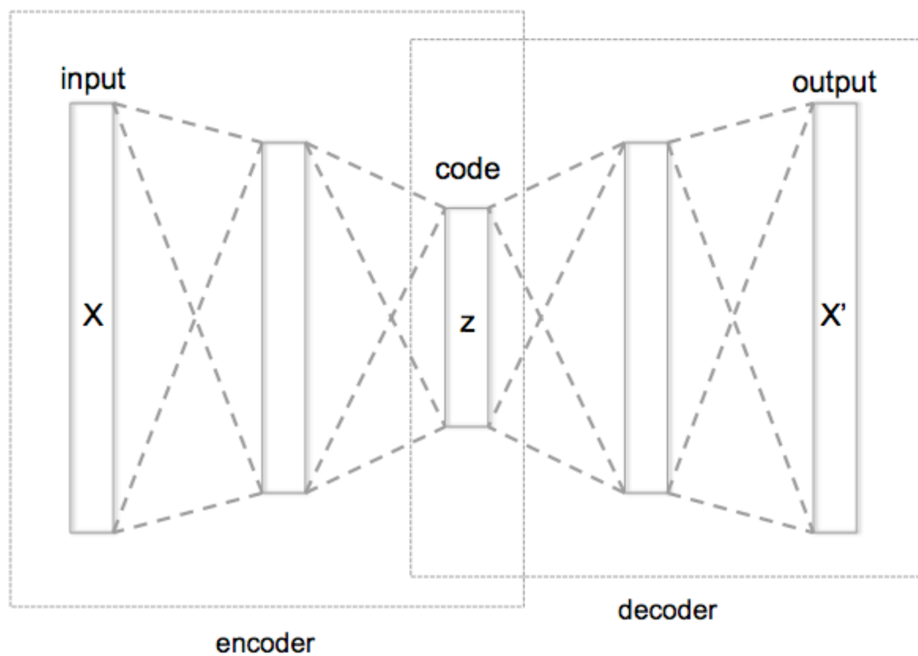
- Available LDA implementations e.g. MALLET, gensim
- Sensitive to:
  - Number of topics N
  - Hyperparameters α, β
    - "spikiness" of distributions
    - How distinct should topics be?
    - How distinct should documents be?
    - Automatic optimisation possible
      - (but be careful!)
- Output:
  - N topics as word distributions
  - MLE topic distributions for any document
- Often find small number of "junk" topics
  - (see e.g. Griffiths et al 2005 for solutions)

0	time session sorry today great send next now one work thanks see thank please help make able perhaps look
1	feel life think know way things now like want make self feelings people change maybe someone much need others
2	right well great sure appointment feel thank just lol tonight please know get sorry say bye meeting last though
3	eating eat food weight sick drink meal now lunch control great chocolate absolutely day healthy dinner put use really
4	time husband mum family feel children now dad want see said friends also kids home life got school daughter
5	people say angry situation anger situations said way social others like one friends talk someone person behaviour saying know
6	get go know like need things going just think try want one something time good now make day start

(Howes et al, 2014) – topics from therapy for depression

Auto-encoders

- We can achieve similar dimensionality reduction using neural networks:
  - "Autoencoders" – learn optimal reduced encoding from which the input can be reconstructed with some accuracy
  - (more about this later)



http://en.wikipedia.org

Summary

- Estimating models without full supervision:
  - General class of models: latent variable models
  - General method for estimation: Expectation-Maximisation (EM)

- Specific implementations in common areas:
  - Clustering: k-means, Brown etc
  - Generative models: Forward-Backward etc
  - Topic models: pLSI, LDA etc

- No guarantee of exact solution
  - Sensitive to parameters & initialisation
  - Often need multiple runs with
  - Not always obvious how to evaluate & compare