TEXT MINING
STATISTICAL MODELING OF TEXTUAL DATA
LECTURE 1

Måns Magnusson, Mattias Villani

Division of Statistics and machine learning
Dept. of Computer and Information Science
Linköping University

Overview 'Statistics for Textual Data'

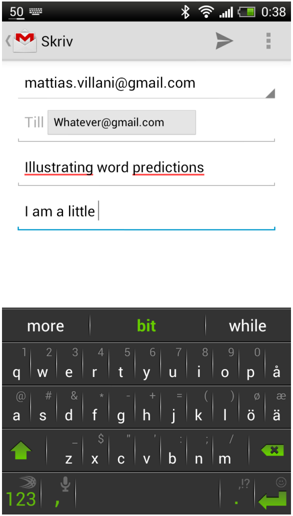
- **Lecture 1**
  - **Language models and n-grams**
  - **Part-of-speech tagging**
- **Lecture 2**
  - **Text classification**
- **Lecture 3**
  - **Topic models**

Some definitions

A neutron walks into a bar and asks how much for a drink. The bartender replies "for you, no charge".

- Tokens
- Types / word types
- Vocabulary
- Sentence / Document / text segment / context
- Corpus

LANGUAGE MODELS - PREDICT THE NEXT WORD



Language models

- Let w_i denote the ith word in a sentence. Let $\mathbf{w}_1^k = w_1 w_2 \cdots w_k$ denote a sentence of k tokens.

- The probability of a sentence

$$p(\mathbf{w}_1^n) = p(w_1) \cdot p(w_2|w_1) \cdot p(w_3|\mathbf{w}_1^2) \cdots p(w_n|\mathbf{w}_1^{n-1})$$

- Probability distribution over the next token in a sentence:

$$p(w_k|\mathbf{w}_1^{k-1})$$

- Example:

$$p(\text{mall}|\text{I like to go to the}) = 0.2$$

$$p(\text{school}|\text{I like to go to the}) = 0.001$$

- Add beginning of sentence token/tag <s>.

Unigram models

- **Unigram language models** ignores the previous words:

$$p(w_n|w_1, ..., w_{n-1}) = p(w_n)$$

- $p(w_n)$ can be estimated using **maximum likelihood (ML)** estimation as:

$$\hat{p}_{ML}(w_n) = \frac{C(w_n)}{N}$$

where $C(w_n)$ is the number of tokens of word type w_n

- Simulating a text from a bag-of-words model gives rubbish.

Much asks into neutron asks.

Language models - n-grams

- The **bigram** model

$$p(w_n|w_1, ..., w_{n-1}) = p(w_n|w_{n-1})$$

- ML estimate:

$$\hat{p}(w_n|w_{n-1}) = \frac{\text{Number of times word } w_n \text{ follows directly after } w_{n-1}}{\text{Number of times } w_{n-1} \text{appears in the text}}$$

- Alternative formulation

$$\hat{p}(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})}$$

- The bigram language model can therefore be estimated from unigram and bigram counts.

- **Trigram model**: $p(w_n|w_{n-1}, w_{n-2})$ and so on.

Evaluating language models

- "Unsupervised" models - no ground truth
- Evaluating language models by **Perplexity** (**PP**)

$$\mathrm{PP} = \sqrt[N]{\frac{1}{P(w_1 w_2 \cdots w_N)}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \cdots w_{i-1})}}$$

- Minimizing perplexity is the same as maximizing probability of a sentence
- Evaluate on hold-out test set
- Not necessarily correlated with human judgement

Software n-gram models

Python NLTK

- `nltk.bigrams()`
- `nltk.trigrams()`
- `nGramModel = nltk.NgramModel(2,text7)` # Training a bigram model from text7
- `nGramModel.generate(num_words=50)` # Simulate a text with 50 words from the model.

R

- `tm` package
- `ngram` package

THE SPARSITY PROBLEM - UNIGRAM CASE

- Maximum likelihood estimator (MLE) for unigram model:

$$\hat{p}_{ML}(w_n) = \frac{C(w_1)}{N}$$

where N is the number of tokens in training corpus.

- Problem with MLE: words not in training corpus are deemed impossible!

$$C(w_1) = 0 \;\Rightarrow\; \hat{p}_{ML}(w_n) = 0$$

- Fixing the MLE: **add-one smoothing (Laplace smoothing)**

$$Pr_{Lap}(w_1) = \frac{C(w_1) + 1}{N + V},$$

where V is the number of word types in vocabulary.

The sparsity problem - n-grams

- **Bigrams** looks for pairs of consecutive words $w_1 w_2$.
- The number of possible outcomes is now $B = V^2$.

- n-grams can have a **huge outcome space** $B = V^n$.
- Lots of n-grams are unseen in training corpus. **Sparsity** problems!

- **Add-one smoothing for n-grams**

$$Pr_{Lap}(w_1 w_2 \cdots w_n) = \frac{C(w_1 w_2 \cdots w_n) + 1}{N + B},$$

where $C(w_1 w_2 \cdots w_n)$ is the number of n-grams $w_1 w_2 \cdots w_n$ in the training corpus.

Likelihood inference for multinomial data

- **Data**: $y = (n_1, ...n_B)$, where n_b counts the number of observations in the bth category. $\sum_{j=1}^{B} n_j = N$.
- Example: A recent survey among consumer smartphones owners in the U.S. showed that among the $N = 513$ respondents:
  - $n_1 = 180$ owned an iPhone
  - $n_2 = 230$ owned an Android phone
  - $n_3 = 62$ owned a Blackberry phone
  - $n_4 = 41$ owned some other mobile phone.
- Let $\theta_1 = Pr(\text{owns iPhone})$, $\theta_2 = Pr(\text{owns Android})$ etc
- **Likelihood**

$$p(n_1, n_2, ..., n_B | \theta_1, \theta_2, ..., \theta_B) = const \cdot \prod_{j=1}^{B} \theta_j^{n_j}$$

- **Maximum likelihood** (ML) estimator

$$\hat{\theta}_b = \frac{n_b}{N}$$

Bayesian smoothing for multinomial data

- **ML problematic when data is sparse**. $n_b = 0 \Rightarrow \hat{\theta}_b = 0$.
- **Smoothing** using a **Bayesian prior**.
- **Prior:** $\theta \sim \mathrm{Dirichlet}(\alpha_1, ..., \alpha_B)$ with density

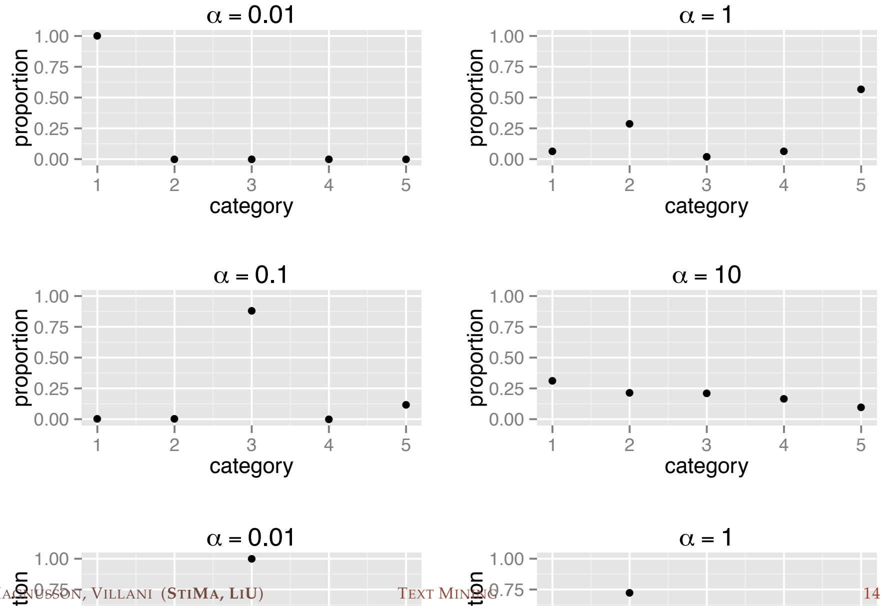
$$p(\theta_1, \theta_2, ..., \theta_B) \propto \prod_{j=1}^{B} \theta_j^{\alpha_j - 1}$$

- Distribution over the simplex
- **Expected value** and **variance** of the *Dirichlet*$(\alpha_1, ..., \alpha_B)$ distribution

$$\mathrm{E}(\theta_b) = \frac{\alpha_b}{\sum_{j=1}^{B} \alpha_j} \qquad \mathrm{V}(\theta_b) = \frac{\mathrm{E}(\theta_b)\left[1 - \mathrm{E}(\theta_b)\right]}{1 + \sum_{j=1}^{B} \alpha_j}$$

- Note that $\sum_{j=1}^{B} \alpha_j$ is a **precision** parameter.

The Dirichlet distribution



Bayesian smoothing for multinomial data

- **Posterior** distribution (Likelihood $\times$Prior)

$$\textit{Posterior}: \quad \theta|n_1, ..., n_B \sim \text{Dirichlet}(n_1 + \alpha_1, ..., n_B + \alpha_B)$$

- **Posterior expected value**

$$E(\theta_b|n_1, ..., n_B) = \frac{n_b + \alpha_b}{N + \sum_{j=1}^{B} \alpha_j}$$

- **Add-one (Laplace) smoothing** obtained with uniform prior $\alpha_1 = ... = \alpha_B = 1$

$$E(\theta_b|n_1, ..., n_B) = \frac{n_b + 1}{N + B}$$

  where $B = V^n$.

- Not a great solution when $B >> N$. Too much probability mass on unseen words.
- Uniform prior distribution over all n-grams is stupid.

Other smoothing methods

- **Linear interpolation** combines trigram, bigram and unigrams:

$$\hat{p}_{LI}(w_n|w_{n-1}, w_{n-2}) = \lambda_1 \hat{p}(w_n|w_{n-1}, w_{n-2}) + \lambda_2 \hat{p}(w_n|w_{n-1}) + \lambda_3 \hat{p}(w_n)$$

- The parameters λ_1, λ_2 and λ_3 can be chosen by cross-validation.

- **Katz back-off** N-gram model: use N-gram if available, otherwise back-off to $N-1$ gram:

$$\hat{p}_{katz}(w_n|w_{n-N+1}^{n-1}) = \left\{ \begin{array}{cc} \hat{p}(w_n|w_{n-N+1}^{n-1}) & \texttt{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1}) \cdot \hat{p}_{katz}(w_n|w_{n-N+2}^{n-1}) & \texttt{otherwise} \end{array} \right\}$$

- **PoS-based N-grams**: use word classes to better distribute probability mass to unseen trigrams. Verb-Verb-Verb is not a likely sequence.

Part-of-speech tagging

- **Part-of-Speech** (**PoS**) or **word classes** - verb, noun, adjective, preposition etc:
- Examples from 45-tag Penn Treebank:
  - JJ - **Adjective**. JJR - comparative. JJS - superlative
  - NN - **Noun**, singular or mass, NNS - plural NNP - Proper noun, singular NNPS - Proper noun, plural
  - VB - **Verb**, base form. VBD - past tense.
- Brown corpus in NLTK: **The**/at **Fulton**/np-tl **County**/nn-tl **Grand**/jj-tl **Jury**/nn-tl **said**/vbd **Friday**/nr ...

A probabilistic model for POS tagging

- **POS tagging**: determine the sequence of POS tags

$$t_1^n = t_1 t_2 \cdots t_n$$

for the words in the sentence

$$w_1^n = w_1 w_2 \cdots w_n$$

- **Note**: each word gets a POS tag

$$\begin{array}{cccc} w_1 & w_2 & \cdots & w_n \\ t_1 & t_2 & \cdots & t_n \end{array}$$

- **Aim**: posterior distribution of the tags

$$p(t_1^n | w_1^n)$$

A probabilistic model for POS tagging, cont.

- Bayes theorem:

$$p(t_1^n|w_1^n) = \frac{p(w_1^n|t_1^n)p(t_1^n)}{p(w_1^n)}$$

- Since $p(w_1^n)$ does not depend on t_1^n, we can use

$$p(t_1^n|w_1^n) \propto p(w_1^n|t_1^n)p(t_1^n)$$

- **Problem**: outcome space of t_1^n is enormous. Example: $n = 5$ with 45-tag set: $45^5 = 184528125$.
- Example

	I	am	great	at	grammar	$p(t_1^n\|w_1^n)$
	t_1	t_2	t_3	t_4	t_5	0.001
1	*JJ*	*VB*	*JJ*	*VB*	*VBD*	0.002
2	*VB*	*VB*	*JJ*	*JJ*	*VBD*	0.002
3	*NN*	*JJ*	*NNP*	*VB*	*JJ*	0.005
⋮	⋮	⋮	⋮	⋮	⋮	⋮
45^5	*JJ*	*VB*	*DT*	*VB*	*NN*	0.003

A probabilistic model for POS tagging, cont.

- Two simplifying assumptions makes the problem manageable.
- **Assumption 1**: **each word depends only on its tag**:

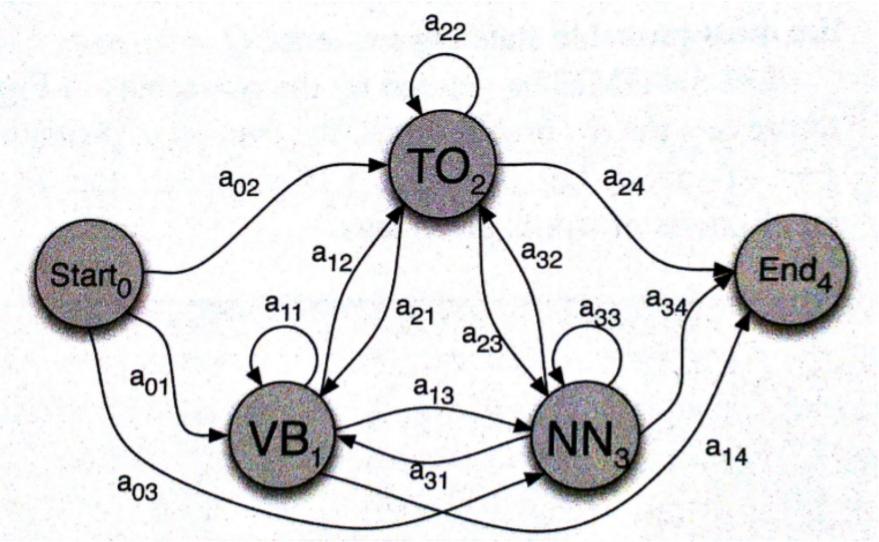
$$p(w_1^n|t_1^n) = \prod_{i=1}^{n} p(w_i|t_i)$$

- **Assumption 2**: **Bigram assumption** for the **tags**:

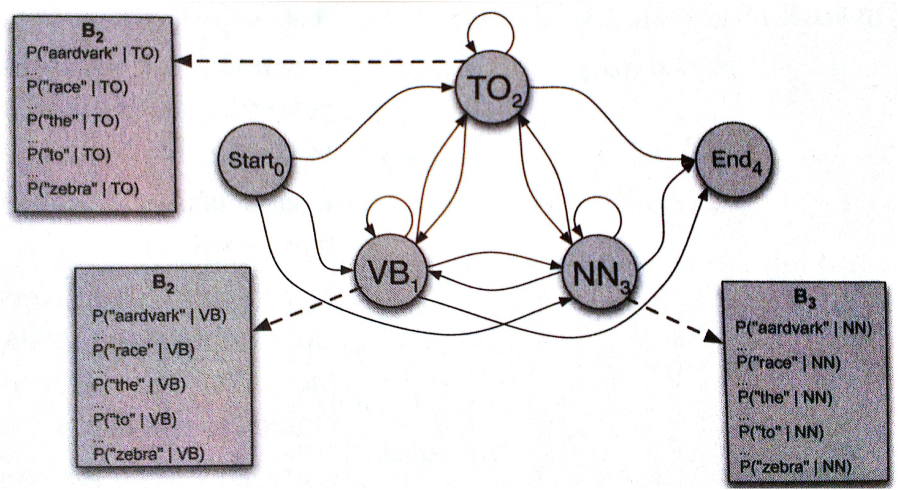
$$p(t_1^n) = \prod_{i=1}^{n} p(t_i|t_{i-1})$$

- **Hidden Markov model (HMM)**

Markov model for POS tags - HMM model



Observation likelihoods - HMM model



Part-of-speech tagging, cont.

- The POS prior

$$p(t_1^n) = \prod_{i=1}^{n} p(t_i|t_{i-1})$$

  can be estimated as a bigram model from a tagged corpus.

- The word distribution $p(w_i|t_i)$ can be estimated by

$$\hat{p}(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

- The solution to the prediction (parsing) problem

$$\mathrm{argmax}_{t_1^n}\, p(t_1^n|w_1^n)$$

  can be found by the **Viterbi** algorithm.

Part-of-speech tagging, cont.

- Gibbs sampling can be used to draw samples from the posterior

$$p(t_1^n | w_1^n)$$

  both using supervised data, semi-supervised and unsupervised.

- Can be combined with more complex probabilistic models