Tampere University

Text classification

Tampere University

- Classification or categorization: given some text, decide to which class it belongs to
- **Language identification** and **genre classification** are examples of text classification
- In **sentiment analysis** one classifies a movie or product review as positive or negative
- In **spam detection** one classifies an email message as spam or ham
- Spam detection is a problem in supervised learning
- A training set is readily available: the positive (spam) in my spam folder, the negative (ham) examples in my inbox

- **Wholesale Fashion Watches -57% today. Designer watches for cheap ...**
- **You can buy ViagraFr$1.85 All Medications at unbeatable prices! ...**
- Sta.rt earn*ing the salary yo,u d-eserve by o'btaining the prope,r crede'ntials!
- The practical significance of hypertree width in identifying more ...
- We will motivate the problem of social identity clustering: ...
- Good to see you my friend. Hey Peter, It was good to hear from you. ...
- PDS implies convexity of the resulting optimization problem (Kernel Ridge ...

Tampere University

Language modeling approach

- In this approach, we define one n-gram language model for
  - $P(message|spam)$ by training on the spam folder, and one for
  - $P(message|ham)$ by training on the inbox
- Then we can classify a new message with an application of Bayes' rule:

$$\underset{c \in \{spam, ham\}}{\operatorname{argmax}} P(c|message) = \underset{c \in \{spam, ham\}}{\operatorname{argmax}} P(message|c)\, P(c)$$

- Count the total number of spam and ham messages to estimate $P(c)$
- This approach works well for spam detection, just as it did for language identification

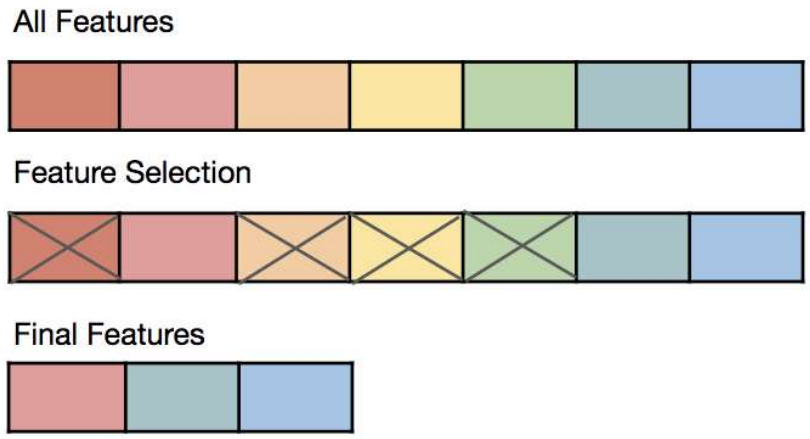
Machine learning approach

- Represent the message as a set of feature/value pairs and apply a classification algorithm h to the feature vector X
- Make the two approaches compatible by thinking of the unigrams as features
- The features are the words in the vocabulary: “a,” “aardvark,” …, and the values are the number of times each word appears in the message
- Large and sparse feature vector
- If there are 100,000 words in the language model, then the feature vector has length 100,000, but for a short email message almost all the features will have count zero
- This unigram representation has been called the **bag of words** model
- The notion of order of the words is lost; a unigram model gives the same probability to any permutation of a text
- Higher-order n-gram models maintain some local notion of word order

- With bigrams and trigrams the number of features is squared or cubed
- We can add in other, non-n-gram features:
  - the time the message was sent,
  - whether a URL or an image is part of the message,
  - an ID number for the sender of the message,
  - the sender's number of previous spam and ham messages, and so on

- The choice of features is the most important part of creating a good spam detector
  - more than the choice of algorithm for processing the features
- There is a lot of training data
  - if we can propose a feature, the data can accurately determine if it is good or not
- It is necessary to update features
  - spam detection is an **adversarial task**; the spammers modify their spam in response to the spam detector's changes

Feature selection



- It is expensive to run algorithms on a very large feature vector
- **Feature selection** keeps only the features that discriminate between spam and ham
- E.g., the bigram “of the” may be equally frequent in spam and ham, so there is no sense in counting it
- Often the top hundred or so features do a good job of discriminating between classes
- After choosing a set of features, we can apply any of the supervised learning techniques; k-nearest-neighbors, SVMs, decision trees, naive Bayes, or logistic regression
- All of these achieve an accuracy in the 98%–99% range
- With a carefully designed feature set, accuracy can exceed 99.9%

Tampere University

Information retrieval

- IR is the task of finding documents that are relevant to a user's need for information
- The best-known examples of information retrieval systems are search engines on the Web
- A Web user can type a query such as

  [AI book]

  into a search engine and see a list of relevant pages
- The characteristics of an IR system:

1. **A corpus of documents**
   - What you treat as a document: a paragraph, a page, or a multipage text
2. Queries posed in a **query language**
   - A query specifies what the user wants to know
   - The query language:
     - a list of words: [AI book];
     - specify a phrase of words that must be adjacent: [“AI book”];
     - contain Boolean operators: [AI AND book];
     - include non-Boolean operators such as [AI NEAR book] or [AI book site:www.aaai.org]
3. A **result set**
   - The documents that the IR system judges to be **relevant** to the query, likely to be of use to the person who posed the query, for the particular information need expressed in the query
4. A **presentation** of the result set
   - Simple as a ranked list of document titles or as complex as a rotating color map of the result set projected onto a three-dimensional space, rendered as a 2D display

TF & IDF

- **Frequency** with which a query term appears in a document (TF)
  - For the query [farming in Kansas], documents that mention “farming” frequently will have higher scores

- **Inverse document frequency** of the term (IDF)
  - “in” appears in almost every document and has a high document frequency, and thus a low inverse document frequency
  - it is not as important to the query as “farming” or “Kansas”

- **Length of the document**
  - A million-word document will probably mention all the query words, but may not actually be about the query
  - A short document that mentions all the words is a much better candidate

Perplexity

- A different way of describing the probability of a sequence is with a measure called perplexity, defined as

$$Perplexity(c_{1:N}) = P(c_{1:N})^{-1/N}$$

- It can be thought of as the reciprocal of probability, normalized by sequence length
- It can also be thought of as the weighted average branching factor of a model

- Suppose there are 100 characters in our language, and our model says they are all equally likely
- Then for a sequence of any length, the perplexity will be 100
- If some characters are more likely than others, and the model reflects that
- Then the model will have a perplexity less than 100

- The parameter values λ_i can be fixed, or they can be trained with an expectation–maximization (EM) algorithm
- It is also possible to have the values of λ_i depend on the counts:
  - if we have a high count of trigrams, then we weigh them relatively more;
  - if only a low count, then we put more weight on the bigram and unigram models
- Reducing the variance in the language model is the goal
- Note that the expression $P(c_i|c_{i-2:i-1})$ asks for $P(c_1|c_{-1:0})$ when $i = 1$, but there are no characters before c_1
- We can introduce artificial characters, for example, defining c_0 to be a space character or a special “begin text” character
- Or we can fall back on lower-order Markov models, in effect defining $c_{-1:0}$ to be the empty sequence and thus $P(c_1|c_{-1:0}) = P(c_1)$

- Most IR systems use models based on the statistics of word counts
- The Okapi BM25 **scoring function** has been used in search engines such as the open-source Lucene project
- A scoring function takes a document and a query and returns a numeric score; the most relevant documents have the highest scores
- In the BM25 function, the score is a linear weighted combination of scores for each of the words that make up the query
  - Term frequency (TF)
  - Inverse document frequency (IDF)
  - Length of the document

The Okapi BM25 function

- Assume we have an index of the N documents in the corpus so that we can look up $TF(q_i, d_j)$, the count of the number of times word q_i appears in document d_j
- Also assume a table of document frequency counts, $DF(q_i)$, that gives the number of documents that contain the word q_i
- Then, given a document d_j and a query consisting of the words $q_{i:N}$, we have

$$BM25(d_j, q_{i:N}) = \sum_{i=1}^{N} IDF(q_i) \cdot \frac{TF(q_i, d_j) \cdot (k+1)}{TF(q_i, d_j) + k \cdot (1 - b + b \cdot |d_j|/L)}$$

- where $|d_i|$ is the length of document d_i in words, and L is the average document length in the corpus: $L = \sum_i |d_i| / N$
- We have two parameters, k and b, that can be tuned by cross-validation; typical values are $k = 2.0$ and $b = 0.75$

Inverse document frequency

- $IDF(q_i)$ is the inverse document frequency of word q_i:

$$IDF(q_i) = \log \frac{N - DF(q_i) + 0.5}{DF(q_i) + 0.5}$$

- Of course, it would be impractical to apply the BM25 scoring function to every document in the corpus
- Instead, systems create an **index** ahead of time that lists, for each vocabulary word, the documents that contain the word
- This is called the **hit list** for the word
- Then when given a query, we intersect the hit lists of the query words and only score the documents in the intersection

Tampere University

IR system evaluation

- To assess whether our IR system is performing well, we give it a set of queries and the result sets are scored w.r.t. human relevance judgments
- Traditionally, there have been two measures used in the scoring: recall and precision
- Imagine that an IR system has returned a result set for a single query, for which we know which documents are and are not relevant, out of a corpus of 100 documents:

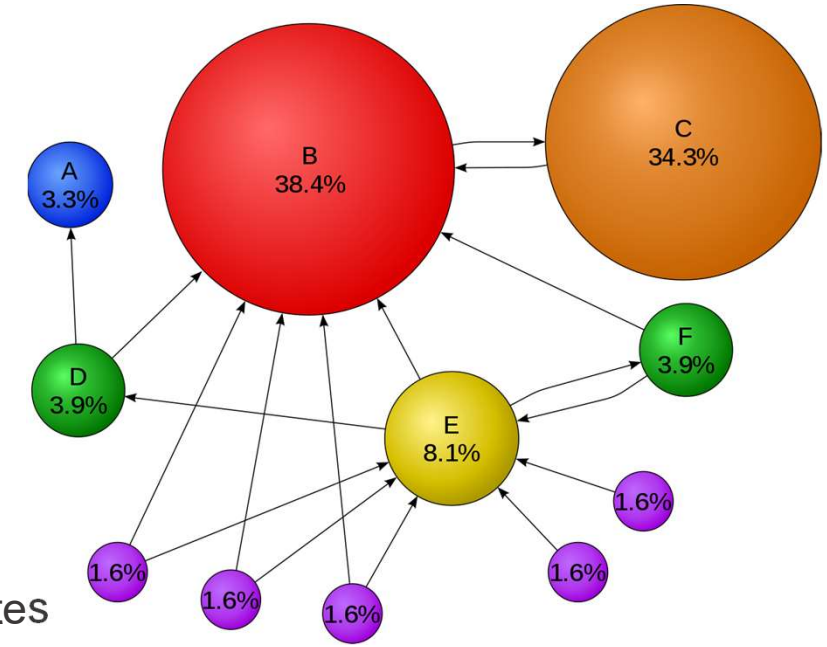
	In result set	Not in result set
Relevant	30	20
Not relevant	10	40

	In result set	Not in result set
Relevant	30	20
Not relevant	10	40

- **Precision** measures the proportion of documents in the result set that are actually relevant. Here, the precision is $30/(30+10) = .75$
- The false positive rate is $1 - .75 = .25$
- **Recall** measures the proportion of all the relevant documents in the collection that are in the result set. Now, recall is $30/(30+20) = .60$
- The false negative rate is 1 −.60 = .40
- In a very large document collection (Web) recall is difficult to compute, because there is no easy way to examine every page on the Web for relevance
- Either estimate recall by sampling or ignore it completely and just judge precision
- In the case of a Web search engine, there may be thousands of documents in the result set, so it makes more sense to measure precision for several different sizes, such as “P@10” (precision in the top 10 results) or “P@50,” rather than to estimate precision in the entire result set

The PageRank algorithm

- One of the original ideas that set Google apart from other Web search engines when it was introduced in 1997
- PageRank solves the problem of the tyranny of TF scores:
  - if the query is [IBM], how do we make sure that IBM's home page is the first result, even if another page mentions the term "IBM" more frequently?
- `ibm.com` has many in-links (links to the page), so it should be ranked higher: each in-link is a vote for the quality of the linked-to page
- If we only counted in-links, then a Web spammer could create a network of pages and have them all point to a page of his choosing, increasing the score of that page
- Therefore, PageRank weights links from high-quality sites more heavily



- What is a high-quality site?
- One that is linked to by other high-quality sites
- The definition is recursive, but we will see that the recursion bottoms out properly
- The PageRank for a page p, $PR(p)$, is defined as:

$$PR(p) = \frac{1-d}{N} + d\sum_i \frac{PR(in_i)}{C(in_i)},$$

- where N is the total number of pages in the corpus, in_i are the pages that link in to p, and $C(in_i)$ is the count of the total number of out-links on page in_i

- The constant d is a damping factor
- It can be understood through the **random surfer model**:
  - A Web surfer who starts at some random page and begins exploring
  - With probability d (assume $d = 0.85$) the surfer clicks on one of the links on the page (choosing uniformly among them), and
  - with probability $1 - d$ she gets bored with the page and restarts on a random page anywhere on the Web

- The PageRank of page p is then the probability that the random surfer will be at page p at any point in time
- PageRank can be computed by an iterative procedure: start with all pages having $PR(p) = 1$, and iterate the algorithm, updating ranks until they converge

Information extraction

- The process of acquiring knowledge by skimming a text and looking for occurrences of a particular class of object and for relationships among objects
- A typical task is to extract instances of addresses from Web pages, with database fields for street, city, state, and zip code; or instances of storms from weather reports, with fields for temperature, wind speed, and precipitation
- In a limited domain, this can be done with high accuracy
- As the domain gets more general, more complex linguistic models and more complex learning techniques are necessary
- There are no complete models, so for the limited needs of information extraction,
  - we define limited models that approximate the full English model, and concentrate on just the parts that are needed for the task at hand

Finite-state automata for information extraction

- The simplest type of information extraction system is an **attribute-based extraction** system that assumes that the entire text refers to a single object and the task is to extract attributes of that object
- For example, the problem of extracting from the text

  Lenovo ThinkPad T430. Our price: 399.00€

  the set of attributes

  {Manufacturer=Lenovo, Model=ThinkPadT430, Price=399.00€}
- We can address this problem by defining a **template** (a pattern) for each attribute we would like to extract
- The template is defined by a finite state automaton, the simplest example of which is the **regular expression**

Tampere University

Regexs

- Regular expressions are used in Unix commands such as **`grep`**, in programming languages such as Perl, and in word processors such as Microsoft Word
- how to build up a regular expression template for prices in euros:
- `[0-9]` any digit from 0 to 9
- `[0-9]+` one or more digits
- `[.][0-9][0-9]` a period followed by two digits
- `([.][0-9][0-9])?` a period followed by two digits, or nothing
- `[0-9]+([.][0-9][0-9][€])?` 249.99€ or 1.23€ or 1000000€ or …

- Templates are often defined with three parts: prefix / target / postfix regex
- The prefix looks for strings such as “price:” and the postfix could be empty
- Some clues about an attribute come from the attribute value itself and some come from the surrounding text
- If a regular expression for an attribute matches the text exactly once, then we can pull out the portion of the text that is the value of the attribute
- If there is no match, give a default value or leave the attribute missing, if there are several matches, we need a process to choose among them
- One strategy is to have several templates for each attribute, ordered by priority
- E.g., the top-priority template for price might look for the prefix “our price:”; if not found, look for the prefix “price:” and if that is not found, the empty prefix
- Alternative strategy: take all the matches and to choose among them
- Taking the lowest price that is within 50% of the highest price will select 78.00€ as the target from the text

List price 99.00€, special sale price 78.00€, shipping 3.00€.

Relational extraction

- One step up from attribute-based systems are **relational extraction** systems, which deal with multiple objects and the relations among them
- Thus, when these systems see the text “249.99€,” they need to determine not just that it is a price, but also which object has that price
- System FASTUS handles news stories about corporate mergers and acquisitions. It reads the story
  - *Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.*
- and extracts the relations:

$$e \in \text{JointVentures} \wedge \text{Product}(e, \text{“golf clubs”}) \wedge \text{Date}(e, \text{“Friday”})$$
$$\wedge \text{Member}(e, \text{“Bridgestone Sports Co”}) \wedge \text{Member}(e, \text{“a local concern”})$$
$$\wedge \text{Member}(e, \text{“a Japanese trading house”})$$

FASTUS

- A relational extraction system can be built as a series of **cascaded finite-state transducers**
- The system consists of a series of small, efficient *finite-state automata* (FSAs)
- Each receives text as input, transduces the text into a different format, and passes it along to the next automaton

- FASTUS consists of five stages:
  1. Tokenization
  2. Complex-word handling
  3. Basic-group handling
  4. Complex-phrase handling
  5. Structure merging

Tampere University

Tokenization & complex words

- *Tokenization* segments the stream of characters into tokens (words, numbers, and punctuation)
- For English, tokenization can be fairly simple; just separating characters at white space or punctuation does a fairly good job
- Tokenizers can also deal with markup languages HTML, SGML, and XML
- The second stage handles *complex words*, including collocations such as "set up" and "joint venture," as well as proper names such as "Bridgestone Sports Co."
- These are recognized by a combination of lexical entries and finite-state grammar rules
- For example, a company name might be recognized by the rule
    - CapitalizedWord+ ("Company" | "Co" | "Inc" | "Ltd")

Tampere University

Basic groups

- The third stage handles *basic groups*, meaning noun groups and verb groups
- The idea is to chunk these into units that will be managed by the later stages
- We have simple rules that only approximate the complexity of English, but have the advantage of being representable by finite state automata
- The example sentence emerges as the following sequence of tagged groups:
  1. NG: Bridgestone Sports Co.
  2. VG: said
  3. NG: Friday
  4. NG: it
  5. VG: has set up
  6. NG: a joint venture
  7. PR: in
  8. NG: Taiwan
  9. PR: with
  10. NG: a local concern
  11. CJ: and
  12. NG: a Japanese trading house
  13. VG: to produce
  14. NG: golf clubs
  15. VG: to be shipped
  16. PR: to
  17. NG: Japan
- NG = noun group, VG = verb group, PR = preposition, and CJ = conjunction
- *Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.*

Complex phrases & structure merging

- The fourth stage combines the basic groups into *complex phrases*
  - Again, the aim is to have rules that are finite-state and thus can be processed quickly, and that result in (nearly) unambiguous output phrases
- One type of combination rule deals with domain-specific events
- `Company+ SetUp JointVenture ("with" Company+)?` captures one way to describe the formation of a joint venture
- This stage is the first one in the cascade where the output is placed into a database template as well as being placed in the output stream
- The final stage *merges structures* that were built up in the previous step
- If the next sentence says "*The joint venture will start production in January*," then this step will notice that there are two references to a joint venture, and that they should be merged into one

- Finite-state template-based information extraction works well for a restricted domain where only certain subjects will be discussed, and one knows how they will be mentioned
- The cascaded transducer model helps modularize the necessary knowledge, easing construction of the system
- These systems work especially well when they are reverse-engineering text that has been generated by a program
- For example, a shopping site on the Web is generated by a program that takes database entries and formats them into Web pages; a template-based extractor then recovers the original database
- Finite-state information extraction is less successful at recovering information in highly variable format, such as text written by humans on a variety of subjects

Tampere University

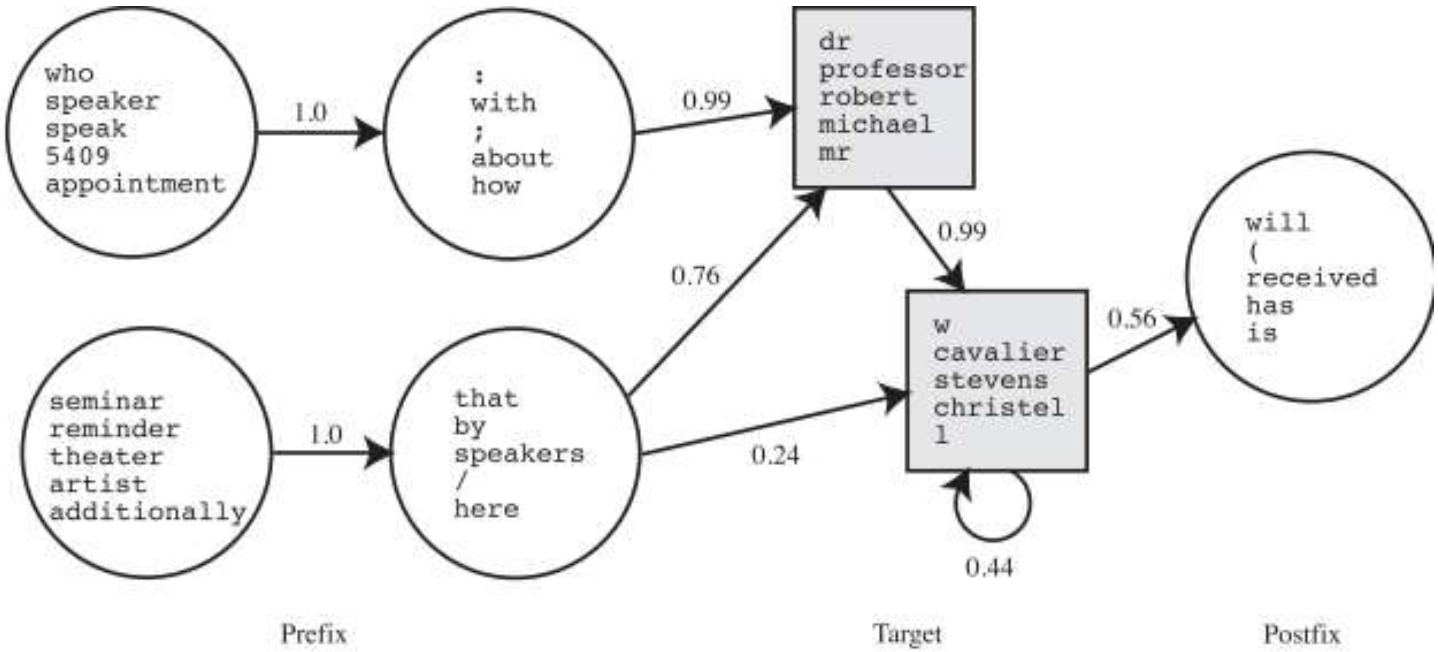
Probabilistic models for IR

- When information extraction must be attempted from noisy or varied input, it is better to use a probabilistic model rather than a rule-based model
- The simplest probabilistic model for sequences with hidden state is the **hidden Markov model**, or HMM
  - It models a progression through a sequence of hidden states, x_t, with an observation at each step
- We build a separate HMM for each attribute
- The observations are the words of the text, and the hidden states are whether we are in the target, prefix, or postfix part of the attribute template, or in the background (not part of a template)
- Here is a brief text and the most probable (Viterbi) path for that text for two HMMs, one trained to recognize the speaker in a talk announcement, and one trained to recognize dates
- The "*" indicates a background state:

Tampere University

• Text:	There	will	be	a	seminar	by	Dr.	Andrew	McCallum	on		Friday	
• Speaker:	*	*	*	*	PRE	PRE	TRGT	TRGT	TRGT		POST	*	
• Date:	*	*	*	*	*	*	*	*	*		PRE	TRGT	

- HMMs are probabilistic, and thus tolerant to noise
- In a regex, if a single expected character is missing, it fails to match
- With HMMs there is graceful degradation with missing characters/ words, and we get a probability indicating the degree of match, not just a Boolean match/fail
- HMMs can be trained from data; they don't require laborious engineering of template
- Thus they can more easily be kept up to date as text changes over time


who
speaker
speak
5409
appointment
1.0
:
with
;
about
how
0.99
dr
professor
robert
michael
mr
0.99
will
(
received
has
is
0.76
w
cavalier
stevens
christel
l
0.56
seminar
reminder
theater
artist
additionally
1.0
that
by
speakers
/
here
0.24
0.44
Prefix
Target
Postfix

- Our HMM templates have a certain level of structure:
  - they all consist of one or more target states, and
  - any prefix states must precede the targets,
  - postfix states most follow the targets, and
  - other states must be background
- This structure makes it easier to learn HMMs from examples
- With a partially specified structure, the forward-backward algorithm can be used to learn both the transition probabilities $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1})$ between states and the observation model, $\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$, which says how likely each word is in state
- For example, the word “Friday” would have high probability in one or more of the target states of the date HMM, and lower probability elsewhere

- The HMM automatically learns a structure of dates that we find intuitive
- The date HMM might have one target state in which
  - the high-probability words are “Monday,” “Tuesday,” etc., and
  - which has a high-probability transition to a target state with words “Jan”, “January,” “Feb,” etc.
- The prefix covers expressions such as “Speaker:” and “seminar by,”
- The target has one state that covers titles and first names and another state that covers initials and last names
- Once the HMMs have been learned, we can apply them to a text, using the Viterbi algorithm to find the most likely path through the HMM states
- One approach is to apply each attribute HMM separately; in this case you would expect most of the HMMs to spend most of their time in background states
- This is appropriate when the extraction is sparse — when the number of extracted words is small compared to the length of the text

- An alternative is to combine all attributes into one HMM, which finds a path that wanders through different target attributes, first finding a speaker target, then a date target, etc.
- Separate HMMs when we expect just one of each attribute in a text and one big HMM when the texts are more free-form and dense with attributes
- With either approach, in the end we have a collection of target attribute observations, and have to decide what to do with them
  - If every expected attribute has one target filler then we have an instance of the desired relation
  - If there are multiple fillers, we need to decide which to choose, as we discussed with template-based systems
- HMMs supply probability numbers that can help make the choice
- If some targets are missing, we need to decide if this is an instance of the desired relation at all, or if the targets found are false positives
- A machine learning algorithm can be trained to make this choice