# Tagged Text and Tagging

* In tagged text each token is assigned a part of speech tag (POS)
* A tagger is a program which automatically ascribes tags to words in text
* From the context we are (most often) able to determine the tag, but some sentences are genuinely ambiguous and hence so are the tags

# Various POS tag sets

* A tagged text is tagged according to a fixed small set of tags
* There are various tagsets:
  + Brown Tagset: 87 tags, Versions with extended tags <original>-<more>
  + Penn treebank tags: 35+9 punctuation tags
  + Universal POS Tagset: 12 tags

# Tagging as Sequence Classification

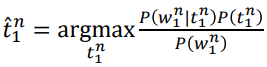
* Classification (earlier):
  + A well-defined set of observations O
  + A given set of classes, S = {s1, s2, …, sk}
  + Goal: a classifier, γ, a mapping from sequences of elements from O to sequences of elements from S: γ (o1, o2, …, on) = (sk1, sk2, …, skn)

# Baseline Tagger

* Establish a baseline classifier in all classification tasks
* Compare the performance of other classifiers to the baseline
* For tagging, a natural baseline is the **Most Frequent Class Baseline**
  + Assign each word the tag which occurred most frequently for that word in the training set
  + For words unseen in the training set, assign the most frequent tag in the training set

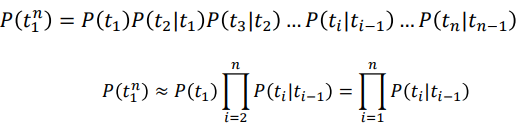
# Hidden Markov Model (HMM) Tagger

|  |  |
| --- | --- |
| Extension of Language Model | Extension of Naïve Bayes |
| Two layers:   * Observed: the sequence of words * Hidden: the tags/classes where each word is assigned a class | * NB assigns a class to each observation * An HMM is a sequence classifier: it assigns a sequence of classes to a sequence of words |

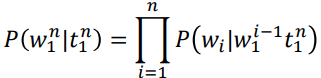
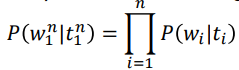
* HMM is a **probabilistic tagger**
* The goal is to decide: 
* Using Bayes theorem 
* This simplifies to  because the denominator is the same for all tag sequences

# HMM: Simplifying Assumption 1

* For the tag sequence, we apply the chain rule
* We then assume the Makov (chain) assumption (assuming a special start tag

*t0*  and *P(t1) = P(t1 | t0)*

# HMM: Simplifying Assumption 2

* Applying the chain rule, i.e. a word depends on all the tags and on all the preceding words 
* We make the simplifying assumption , i.e. a word depends only on the immediately preceding tag, hence

# Training HMM

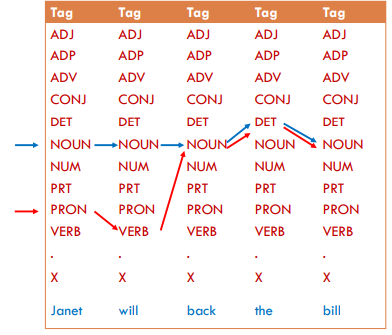
* From a tagged training corpus, we can estimate the probabilities with Maximum Likelihood (as in Language Models and Naïve Bayes)

# Putting it all together

* From a trained model, it is straightforward to calculate *the probability of a sentence with a tag sequence*
* *To find the best tag sequence,* we could – in principle – calculate this for all possible tag sequences and choose the one with the highest score
* **Impossible in practice**

# Possible Tag Sequences

* The number of possible tag sequences
* = the number of paths through the trellis
* = mn
  + m is the number of tags in the set
  + n is the number of tokens in the sentence
  + Here: 125 = 250.000



# Viterbi Algorithm (dynamic programming)

* Walk through the word sequence
* For each word keep track of all the possible tag sequences up to this word and the probability of each sequence
* If two paths are equal from a point on, then the one scoring best at this point will also score best at the end – discard the other one

# HMM trigram tagger

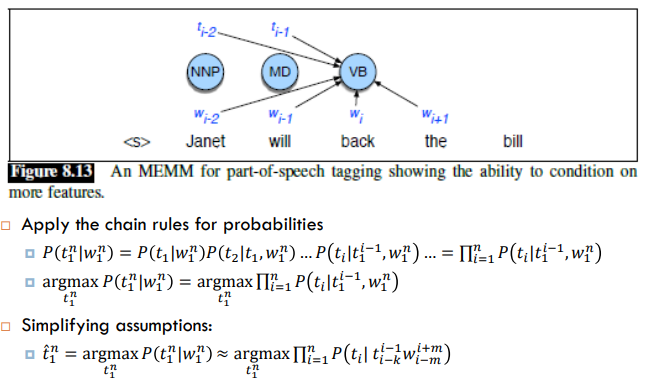
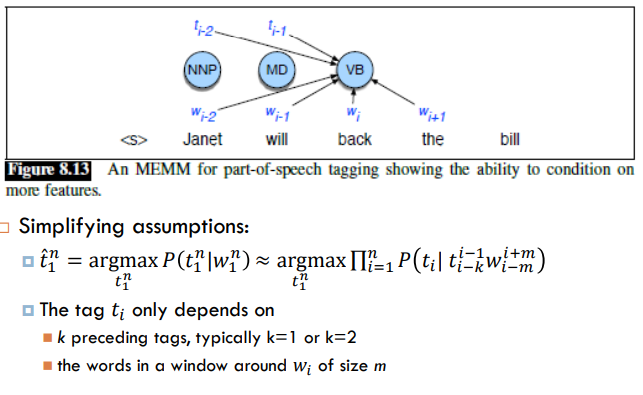
* Take two preceding tags into consideration
* Add two initial special states and one special end state

# Challenges for the HMM-Tagger

* Even with Viterbi, it is expensive:
  + For the trigram, the size of the trellis: (n+2) x m3
  + With *n* words in the sequence and *m* tags in the model
  + Example: 6 words
    - With 12 tags: 15.552
    - With 87 tags: 5.926.527
* We have probably not seen all tag trigrams during training
  + We must use back-off or interpolation to lower n-grams
* Words not observed during training
  + How can we include e.g. morphological features? (e.g. -ly 🡪 ADV)

# Discriminative Tagging

* The goal of tagging is to decide 
* **HMM is generative**. It estimates 
* As for text classification, we could instead use a discriminative procedure and try to estimate the tag sequence directly 



# Feature Extraction



* We use a template to extract features from preceding tag(s) and neighboring words
* The actual number of features my be large
* Observe that properties may be combined into one feature
* Remarks:
  + The extracted features correspond to J&M’s “small features”, *fk (yi-1, yi, X, i)*
  + J&M include the tag into the feature
    - There are alternative ways of presenting this
    - We do not have to include the class
* Features for unknown words
  + We may include features which inspect properties of the word
    - Wi contains a particular prefix (from all prefixes of length <= 4)
    - Wi contains a particular suffix (from all suffixes of length <= 4)
    - Wi contains a number / upper-case letter / hyphen / is all upper case
    - Wi ‘s word shape / short word shape

# Decoding

* Goal:
* Simplest alternative: **Greedy Sequence Decoding**
  + Choose the best tag for the first word in the sequence
  + The choose the best tag for the second word in the sequence, given the choice for the first word 
  + And so on, tagging one word at a time, until we have finished the sentence

# Training a Greedy Classifier

* The training examples are extracted from a tagged corpus
* For each word occurrence in the corpus, one training example:
  + The class is the correct tag
  + The features are extracted from the context
* One can then
  + Train any multi-class ML-algorithm, e.g. multinomial logistic regression
  + Apply greedy tagging on untagged texts

# Shortcomings of greedy decoding

* Early decisions, considers only one tag at a time 🡪
* Compare to HMM which considers whole tag sequences and chooses the most probable sequence

# Maximum Entropy Markov Models (MEMM)

* If the model uses a limited history, one **may use a form of Viterbi decoding**
* We may then find 
* This should make a better result
* However,
  + The greedy sequence decoding does surprisingly well
  + And equally surprising: using preceding tags as features does not improve the tagger that much compared to not including them

# Conditional Random Fields

* Even if we use Viterbi decoding and find the most probable overall tag sequence, we so far trained or model on the greedy task
* What we ideally should have done was to train the model on the task of predicting the optimal whole sequence
* **Conditional Random Fields (CRFs) is a generalization compared to MEMM**:
  + Makes it possible to optimize training for whole tag sequences
  + Slow in training
  + Considered the best tool for sequence labelling until a few years ago
* Currently, **neural networks (“deep learning”) are considered the best tool**

|  |  |  |
| --- | --- | --- |
|  | **Generative** | **Discriminative** |
| **Classification** | Naïve Bayes | Logistic Regression |
| **Sequence Labelling** | HMM | CRF |

# Information Extraction (IE) Basics