An Introduction to Hidden Markov Models

Max Heimel

Fachgebiet Datenbanksysteme und Informationsmanagement
Technische Universität Berlin

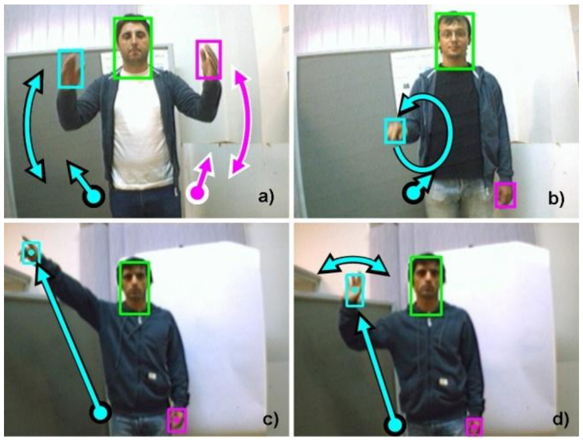
http://www.dima.tu-berlin.de/

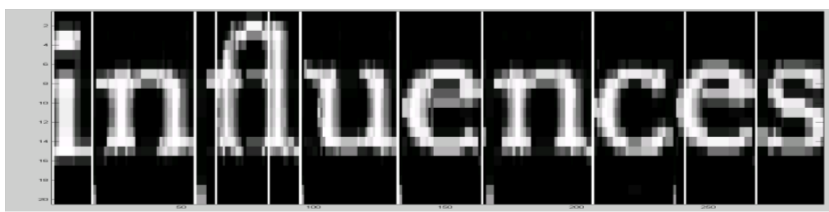
Agenda

- **Motivation**
- An Introduction to Hidden Markov Models
  - What is a Hidden Markov Model?
- Algorithms, Algorithms, Algorithms
  - What are the main problems for HMMs?
  - What are the algorithms to solve them?
- Hidden Markov Models for Apache Mahout
  - A short overview
- Outlook
  - Hidden Markov Models and Map Reduce
  - Take-Home Messages

Motivation

- Pattern recognition: finding structure in sequences.






```
AAB24882        TYHMCQFHCRYVNNHSGEKLYECNERSKAFSCPSHLQCHKRRQIGEKTHEHNQCGKAFPT 60
AAB24881        --------------------YECNQCGKAFAQHSSLKCHYRTHIGEKPYECNQCGKAFSK 40
                                    ****: .***:  * *:** *  :****.:* *******..

AAB24882        PSHLQYHERTHTGEKPYECHQCGQAFKKCSLLQRHKRTHTGEKPYE-CNQCGKAFAQ- 116
AAB24881        HSHLQCHKRTHTGEKPYECNQCGKAFSQHGLLQRHKRTHTGEKPYMNVINMVKPLHNS 98
                 **** *:***********:***:**.: .***************     :  *.: :
```

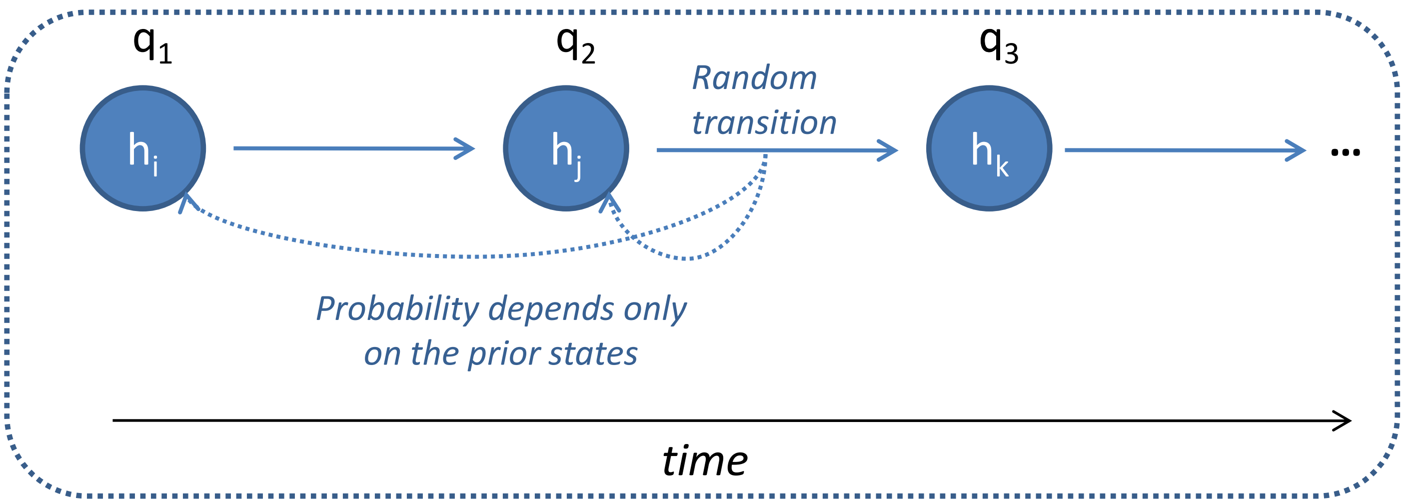
Goals of this talk

- Demonstrate, how sequences can be modeled
  - Using so-called Markov chains.
- Present the statistical tool of Hidden Markov Models
  - A tool to find underlying processes to a given sequence.
- Give an understanding of the main problems associated with Hidden Markov Models
  - And the corresponding applications.
- Present the Apache Mahout implementation of Hidden Markov Models
- Give an outlook on implementing Hidden Markov Models for Map/Reduce

Agenda

- Motivation
- **An Introduction to Hidden Markov Models**
  - **What is a Hidden Markov Model?**
- Applications for Hidden Markov Models
  - What are the main problems for HMMs?
  - What are the algorithms to solve them?
- Hidden Markov Models for Apache Mahout
  - A short overview
- Outlook
  - Hidden Markov Models and Map Reduce
  - Take-Home Messages

- Markov Chains model sequential processes.
- Consider a discrete random variable *q* with states $\{h_1, \ldots, h_n\}$.
- State of *q* changes randomly in discrete time steps.
- Transition probability depends only on the *k* previous states.
  - Markov Property



Markov Chain

Markov Chains II

- Most simplest Markov chain:
  - Transition Probability depends only on the previous state (i.e. k=1):

    $$P(q_t = h_i | q_{t-1}, \ldots, q_1) = P(q_t = h_i | q_{t-1})$$

  - Transition Probability is time invariant:

    $$P(q_t = h_i | q_{t-1}) = P(q_{t-1} = h_i | q_{t-2})$$

- In this case, the Markov chain is defined by:
  - An $(n \times n)$ Matrix *T* containing state change probabilities:

    $$T_{ij} = P(q_t = h_i | q_{t-1} = h_j)$$

  - An n-dimensional vector π containing initial state probabilities:

    $$\pi_i = P(q_1 = h_i)$$

  - Since π and *K* contain probabilities, they have to be normalized:

    $$\sum_{i=1}^{n} \pi_i = 1 \qquad \forall a : \sum_{i=1}^{n} K_{ai} = 1$$

Markov Chains III

- Markov Chains are used to model sequences of states.
- Consider the weather:
  - Each day can either be rainy or sunny.
  - If a day is rainy, there is a 60% chance the next day will also be rainy.
  - If a day is sunny, there is a 80% chance the next day will also be sunny.
  - We can now model the weather as a Markov Chain:

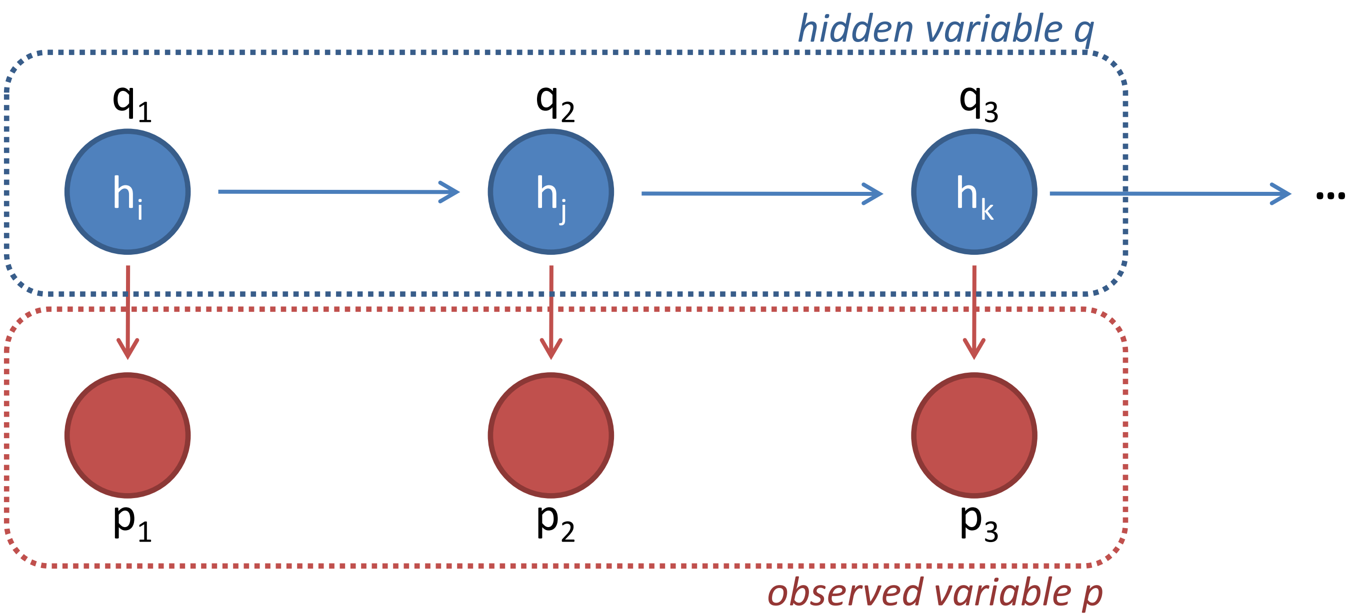
$$T = \begin{array}{lcc} & \text{rainy} & \text{sunny} \\ \text{rainy} & 0.6 & 0.4 \\ \text{sunny} & 0.2 & 0.8 \end{array} \qquad \pi = \begin{array}{lc} \text{rainy} & 0.5 \\ \text{sunny} & 0.5 \end{array}$$

- Examples for the use of Markov chains are:
  - Google Page Rank
  - Random Number Generation, Random Text Generation
  - Queuing Theory
  - Modeling DNA sequences
  - Physical processes from Thermodynamics & statistical Mechanics

Hidden Markov Models I

- Now consider a „hidden“ Markov chain
  - We can not directly observe the states of the hidden Markov chain.
  - However: we can observe effects of the „hidden states“.

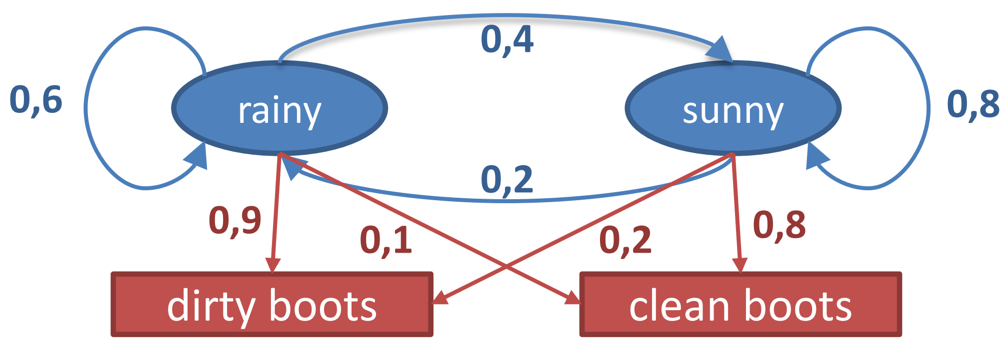
- Hidden Markov Models (HMMs) are used to model such a situation:
  - Consider a Markov chain and a random – not necessarily discrete - variable *p.*
  - The state of p is chosen randomly, based only on the current state of *q*.



Hidden Markov Models II

- The "simplest" HMM has a discrete observable variable *p*
  - The states of *q* are take from the set $\{o_1, \ldots, o_m\}$
- In this case, the HMM is defined by the following parameters:
  - The matrix T and vector π of the underlying Markov Chain.
  - An $(n \times m)$ matrix O containing output probabilities:

    $$O_{ij} = P(p_t = o_j | q_t = h_i)$$
  - Again, O needs to be normalized: $\sum_{i=1}^{n} O_{ij} = 1$
- Consider a prisoner in solitary confinement:
  - The prisoner cannot directly observe the weather.
  - However: he can observe the condition of the boots of the prison guards.



Agenda

- Motivation
- An Introduction to Hidden Markov Models
  - What is a Hidden Markov Model?
- **Applications for Hidden Markov Models**
  - **What are the main problems for HMMs?**
  - **What are the algorithms to solve them?**
- Hidden Markov Models for Apache Mahout
  - A short overview
- Outlook
  - Hidden Markov Models and Map Reduce
  - Take-Home Messages

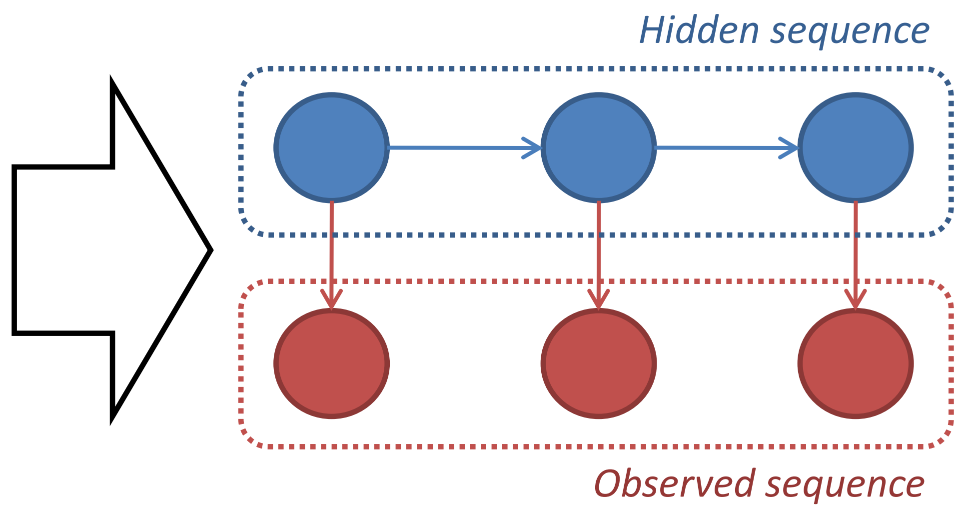
Problems for Hidden Markov Models

Problem	Given	Wanted
Evaluation	*Model, Observed Sequence*	Likelihood the model produced the observed sequence
Decoding	*Model, Observed Sequence*	Most likely hidden sequence
Learning (unsupervised)	*Observed Sequence*	Most likely model that produced the observed sequence
Learning (supervised)	*Observed- & Hidden sequence*	Most likely model that produced the observed & hidden sequence.

$$T = \begin{pmatrix} T_{11} & \dots & T_{1n} \\ \vdots & & \vdots \\ T_{n1} & \dots & T_{nn} \end{pmatrix} \qquad \pi = \begin{pmatrix} \pi_1 \\ \vdots \\ \pi_n \end{pmatrix}$$

$$O = \begin{pmatrix} O_{11} & \dots & \dots & O_{1m} \\ \vdots & & & \vdots \\ O_{n1} & \dots & \dots & O_{nm} \end{pmatrix}$$

Model



Evaluation

- Compute the likelihood that a given model *M* produced a given observation sequence *O*.

$$P(p_1 = O_1, \ldots, p_T = O_T | M)$$

- The likelihood can be efficiently calculated using dynamic programming:
  - Forward algorithm:
    - Reproduce the observation through the HMM, computing:
      $\alpha_i(t) = P(p_1 = O_1, \ldots, p_t = O_t, q_t = h_i | M)$
  - Backward algorithm:
    - Backtrace the observation through the HMM, computing:
      $\beta_i(t) = P(p_t = O_t, \ldots, p_T = O_T | q_t = h_i)$

- Typical application:
  - Selecting the most likely out of several competing models.
  - Customer behavior modeling: select the most likely customer profile
  - Physics: select the most likely thermodynamics process

Decoding

- Compute the most likely sequence of hidden states for a given model M and a given observation sequence O.

$$H = \text{argmax}_H P(q_1 = H_1, \ldots, q_T = H_T | M, O)$$

- The most likely hidden path can be computed efficiently using the Viterbi-algorithm
  - The Viterbi algorithm is based on the Forward Algorithm.
  - It traces the most likely hidden states while reproducing the output sequence.
- Typical Applications:
  - POS tagging (observed: sentence, hidden: part-of-speech tags)
  - Speech recognition (observed: frequencies, hidden: phonemes)
  - Handwritten letter recognition (observed: pen patterns, hidden: letters)
  - Genome Analysis (observed: genome sequence, hidden: “structures”)

1. Supervised Learning

- Given an observation sequence O and the corresponding hidden sequence H, compute the most likely model M that produces those sequences:
  $$M = \text{argmax}_M P(p_1 = O_1, q_1 = H_1, \ldots, p_T = O_T, q_t = H_T | M)$$
- Solved using "instance counting"
  - Count the hidden state transitions and output state emissions.
  - Use the relative frequencies as estimate for transition probabilities of M.

2. Unsupervised Learning

- Given an observation sequence O, compute the most likely model M that produces this sequence:
  $$M = \text{argmax}_M P(p_1 = O_1 \ldots, p_T = O_T | M)$$
- Solved using the Baum-Welch algorithm
  - Rather expensive iterative algorithm, but produces guaranteed EM result.
  - Requires a Forward step and a Backward Step through the model per iteration.
- Alternative: Viterbi training
  - Not as expensive as Baum-Welch, but does not guarantee EM result
  - Requires only a Forward step through the model per iteration.

Agenda

- Motivation
- An Introduction to Hidden Markov Models
  - What is a Hidden Markov Model?
- Applications for Hidden Markov Models
  - What are the main problems for HMMs?
  - What are the algorithms to solve them?
- **Hidden Markov Models for Apache Mahout**
  - **A short overview**
- Outlook
  - Hidden Markov Models and Map Reduce
  - Take-Home Messages

- Apache Mahout will contain an implementation of Hidden Markov Models in its upcoming 0.4 release.
  - The implementation is currently sequential (i.e. not Map/Reduce enabled).
  - The implementation covers Hidden Markov models with discrete output states.

- The overall implementation structure is given by three main and two helper classes:
  - *HmmModel*
    - Container for HMM parameters
  - *HmmTrainer*
    - Methods to train a HmmModel from given observations
  - *HmmEvaluator*
    - Methods to analyze (evaluate / decode) a given HmmModel
  - *HmmUtils*
    - Helper methods, e.g. to validate and normalize a HmmModel
  - *HmmAlgorithms*
    - Helper methods containing implementations of Forward, Backward and Viterbi algorithm.

HmmModel

- HmmModel is the main class for defining a Hidden Markov Model.
  - It contains the transition matrix K, emission matrix O and initial probability vector π.

- Construction from given Mahout matrices K, O and Mahout vector pi:
  - `HmmModel model = new HmmModel(K, O, pi);`

- Construction of a random model with n hidden and m observable states:
  - `HmmModel model = new HmmModel(n, m);`

- Offers serialization and deserialzation from/to JSON:
  - `HmmModel model = HmmModel.fromJson(String json);`
  - `String json = model.toJson();`

HmmTrainer

- Offers a collection of learning algorithms.
- Supervised learning from hidden and observed state sequences:
  - ```
    HmmModel trainSupervised(int hiddenStates,
    int observedStates, int[] hiddenSequence,
    int[] observedSequence, double pseudoCount);
    ```
    Used to avoid zero probabilities

- Unsupervised learning using the Viterbi algorithm:
  - ```
    HmmModel trainViterbi(HmmModel initialModel,
    int[] observedSequence, double pseudoCount, double epsilon,
    int maxIterations, boolean scaled);
    ```
    Use log-scaled implementation –
    slower but numerically more stable

- Unsupervised learning using the Baum-Welch algorithm:
  - ```
    HmmModel trainBaumWelch(HmmModel initialModel,
    int[] observedSequence, double epsilon,
    int maxIterations, boolean scaled);
    ```
    Use log-scaled implementation –
    slower but numerically more stable
    ```

HMMEvaluator

- Offers algorithms to evaluate an HmmModel

- Generating a sequence of output states from the given model:
  - `int[] predict(HmmModel model, int steps);`

- Computing the model likelihood for a given observation:
  - `double modelLikelihood(HmmModel model, int[] observations, boolean scaled);`

Use log-scaled implementation –
slower but numerically more stable

- Compute most likely hidden path for given model and observation:
  - `int[] decode(HmmModel model, int[] observations, boolean scaled);`

Use log-scaled implementation –
slower but numerically more stable

Agenda

- Motivation
- An Introduction to Hidden Markov Models
  - What is a Hidden Markov Model?
- Applications for Hidden Markov Models
  - What are the main problems for HMMs?
  - What are the algorithms to solve them?
- Hidden Markov Models for Apache Mahout
  - A short overview
- **Outlook**
  - **Hidden Markov Models and Map Reduce**
  - **Take-Home Messages**

Outlook

- How can we make HMMs Map Reduce enabled?

- Problem:
  - All the presented algorithms are highly sequential!
  - There is no easy way of parallelizing them.

- However:
  - Hidden Markov Models are often compact (n, m not "too large")
  - The most typical application on a trained HMM is decoding, which can be performed fairly efficient on a single machine.
    - ➔ *Trained HMMs can typically be efficiently used within Mappers/Reducers.*
  - The most expensive – and data intensive – application on HMMs is training.
    - ➔ *Main Goal: parallelize HMM training.*
  - Approaches to parallelizing learning:
    - For supervised learning: trivial, only need to count state changes.
    - For unsupervised learning: tricky, ongoing research :
      - Merging of trained HMMs on subsequences
      - Alternative representations allows training via parallelizable algorithms (e.g. SVD)

Take-Home Messages I

- Hidden Markov Models (HMMs) are a statistical tool to model processes that produce observations based on a hidden state sequence:
  - HMMs consist of a discrete hidden variable that randomly and sequentially changes its state and a random observable variable.
  - Hidden state change probability depends only on the *k* prior hidden states
    - A typical value for *k* is 1.
  - Probability of the observable variable depends only on current hidden state.
- Three main problems for HMMs: evaluation, decoding and training:
  - Evaluation: Likelihood a given model generated a given observed sequence.
    ➔ Forward Algorithm
  - Decoding: Most likely hidden sequence for a given observed sequence and model.
    ➔ Viterbi Algorithm
  - Training: Most likely model that generated a given observed sequence (unsupervised) or a given observed and hidden sequence (supervised).
    ➔ Baum-Welch Algorithm

Take-Home Messages II

- HMMs can be applied whenever an underlying process generates sequential data:
  - Speech Recognition, Handwritten Letter Recognition, Part-of-speech tagging, Genome Analysis, Customer Behavior Analysis, Context aware Search, …
- Mahout contains a HMM implementation in its upcoming 0.4 release.
  - Three main classes: HmmModel, HmmTrainer, HmmEvaluator
  - HmmModel is a container class for representing model parameters.
  - HmmTrainer contains implementations for the learning problem.
  - HmmEvaluator contains implementations for the evaluation and decoding problem.
- Porting HMMs to Map/Reduce is non-trivial
  - Typically, HMMs can be used within a Mapper/Reducer.
  - Most data-intensive task for HMMs: training
  - Porting HMM training to Map/Reduce is ongoing research.

Thank you!