technische universität
dortmund

Build Your Own Handwriting Recognizer

— ICDAR 2011 Tutorial, Bejing, China—

Gernot A. Fink & Szilárd Vajda

TU Dortmund University, Dortmund, Germany
September 18, 2011

- Introduction . . . *Handwriting Recognition Basics*
- MM-based Handwriting Recognition *Fundamentals*
- ESMERALDA . . . *The Development Environment*
- Building the Recognizer
- Adding a Language Model

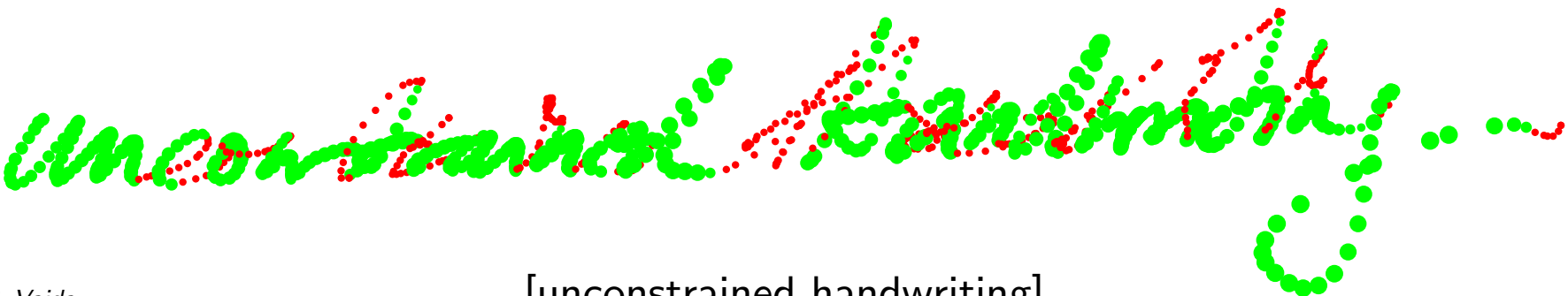
Why Handwriting Recognition?

Script: Symbolic form of archiving speech

- Several different *principles* exist for writing down speech
- *Numerous* different writing systems developed over the centuries
- **Focus here:** Alphabetic writing systems
  ⇒ Especially / most well known: Roman script

Handwriting: In contrast to characters created *by machines*

- Most "natural" form of script in almost all cultures
- Typeface adapted for manual creation ⇒ Cursive script
- "Printed letters" as imitation of machine printed characters
- **Frequently:** Free combination, i.e. mixing of both styles



[unconstrained handwriting]

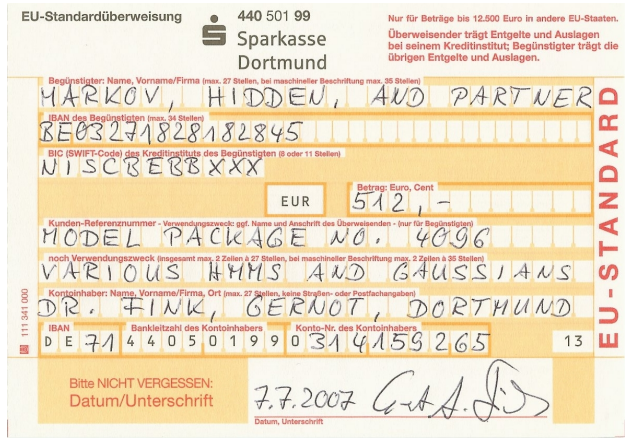
Applications: Off-line vs. On-line Processing

Main objective: Automated document processing
(e.g. Analysis of addresses, forms; archiving)

Basic principle:

- Optical capture of completed typeface
(via scanner, possibly camera)
- *Off-line* analysis of resulting "image data"

⇒ "Optical Character Recognition" (OCR)



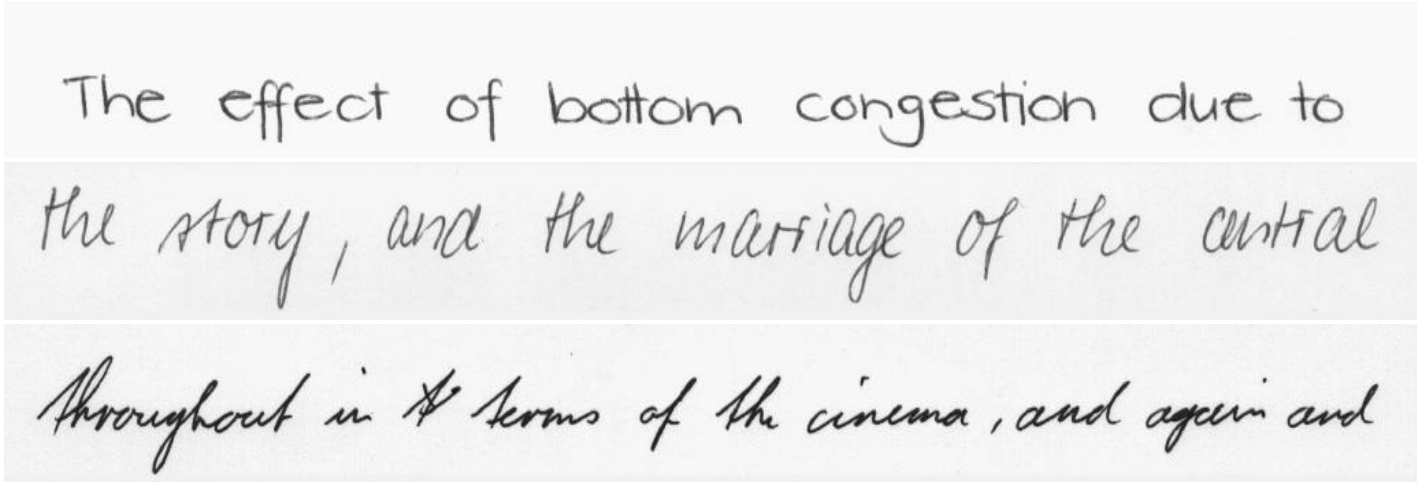
Additionally: Human-Machine-Interaction (HMI)

Basic principle:

- Capture pen trajectory during writing
(using specialized sensors & pens)
- *On-line* analysis of movement data

Why is Handwriting Recognition Difficult?

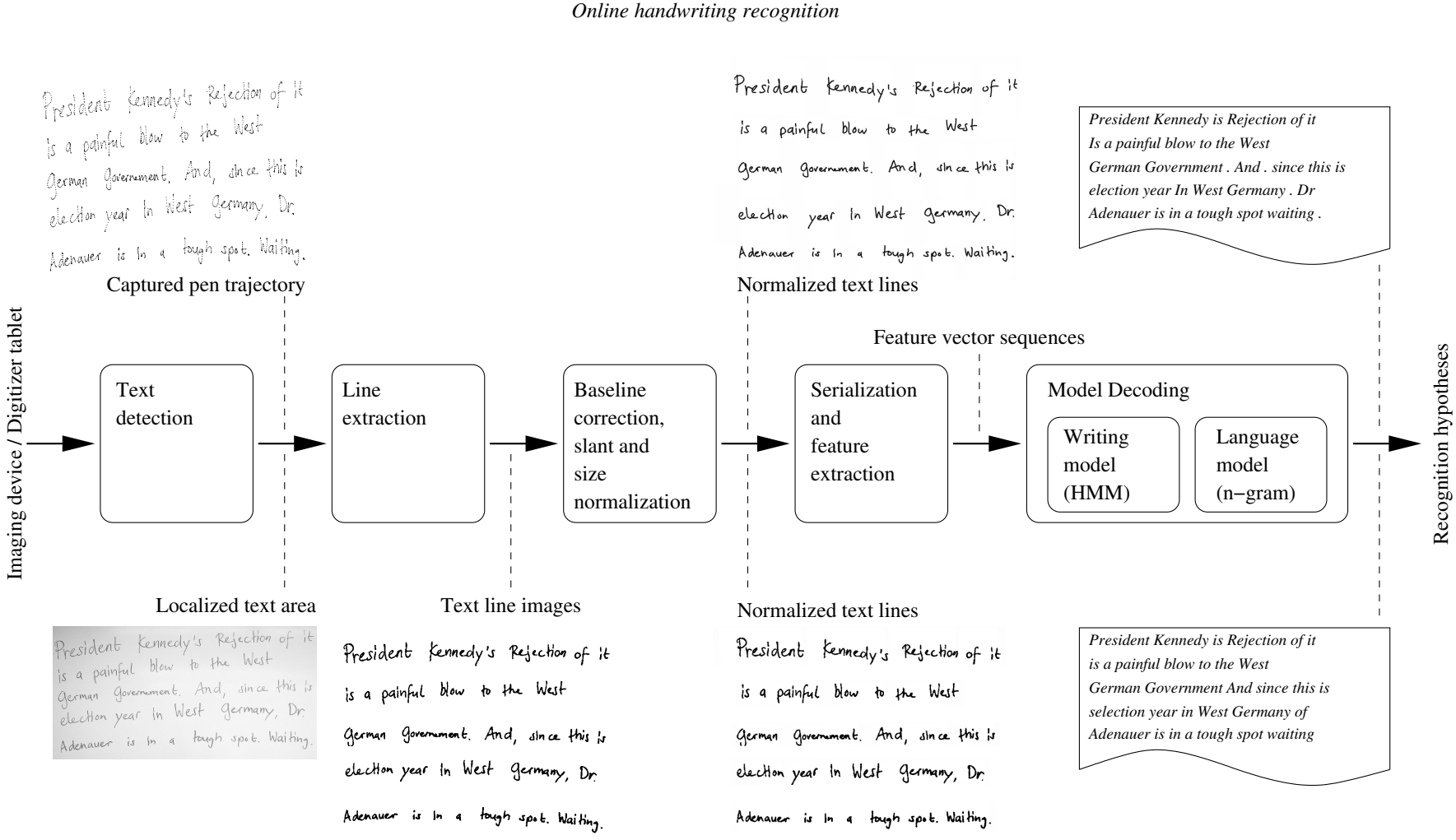
- High variability of individual characters
  - Writing style



  - Stroke width and quality

  - Size of the writing
  - Variation even for single writer!
- Segmentation of *cursive* script problematic
  ⇒ "Merging" of adjacent characters

General Architecture



Focus of this Tutorial

Processing type: Offline

- Handwriting data captured using scanner or camera

Script type & Writing style:

- Alphabetic scripts, especially Roman script
- *No* restriction w.r.t. writing style, size etc.
  $\Rightarrow$ Unconstrained handwriting!

Methods: Statistical Recognition Paradigm

- Markov Models for segmentation free recognition
- Statistical *n*-gram models for text-level restrictions

Goal: Learn how to ...

- Use the ESMERALDA development environment.
- Build a *working* handwriting recognizer.

Overview

“Traditional” Recognition Paradigm

Segmentation
+
Classification:

Original Image

Potential elementary segments, strokes, ...

Alternative segmentations

1.

2.

⋮

n.

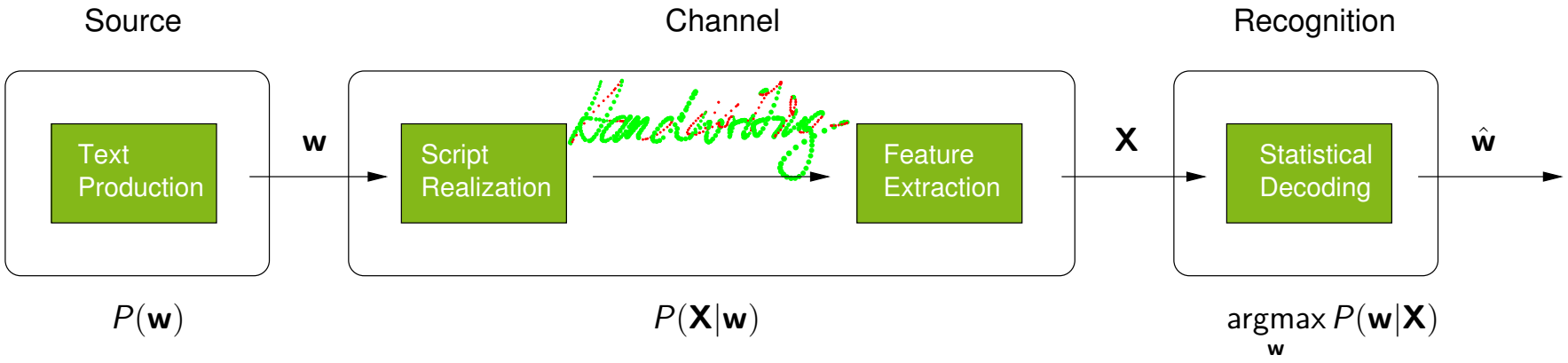
✓ Segment-wise classification possible using various standard techniques

↯ Segmentation is
- costly,
- heuristic, and
- needs to be optimized manually

↯ *Segmentation is especially problematic for unconstrained handwriting!*

Statistical Recognition Paradigm: The Channel Model

(Model originally proposed for automatic speech recognition)



Wanted: Sequence of words/characters $\hat{\mathbf{w}}$, which is most probable for given signal/features $\mathbf{X}$

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}}\, P(\mathbf{w}|\mathbf{X}) = \underset{\mathbf{w}}{\operatorname{argmax}}\, \frac{P(\mathbf{w})P(\mathbf{X}|\mathbf{w})}{P(\mathbf{X})} = \underset{\mathbf{w}}{\operatorname{argmax}}\, P(\mathbf{w})P(\mathbf{X}|\mathbf{w})$$

The Channel Model II

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}}\, P(\mathbf{w}|\mathbf{X}) = \underset{\mathbf{w}}{\operatorname{argmax}}\, \frac{P(\mathbf{w})P(\mathbf{X}|\mathbf{w})}{P(\mathbf{X})} = \underset{\mathbf{w}}{\operatorname{argmax}}\, P(\mathbf{w})P(\mathbf{X}|\mathbf{w})$$

Two aspects of modeling:

- Script (appearance) model: $P(\mathbf{X}|\mathbf{w})$ $\Rightarrow$ Representation of words/characters

  Hidden-Markov-Models

- Language model: $P(\mathbf{w})$ $\Rightarrow$ Restrictions for sequences of words/characters

  Markov Chain Models / n-Gram-Models

Specialty: Script or trajectories of the pen (or features, respectively) interpreted as *temporal* data

✓ Segmentation performed implicitly! $\Rightarrow$ "segmentation free" approach

Script or pen movements, respectively, must be serialized!

Overview

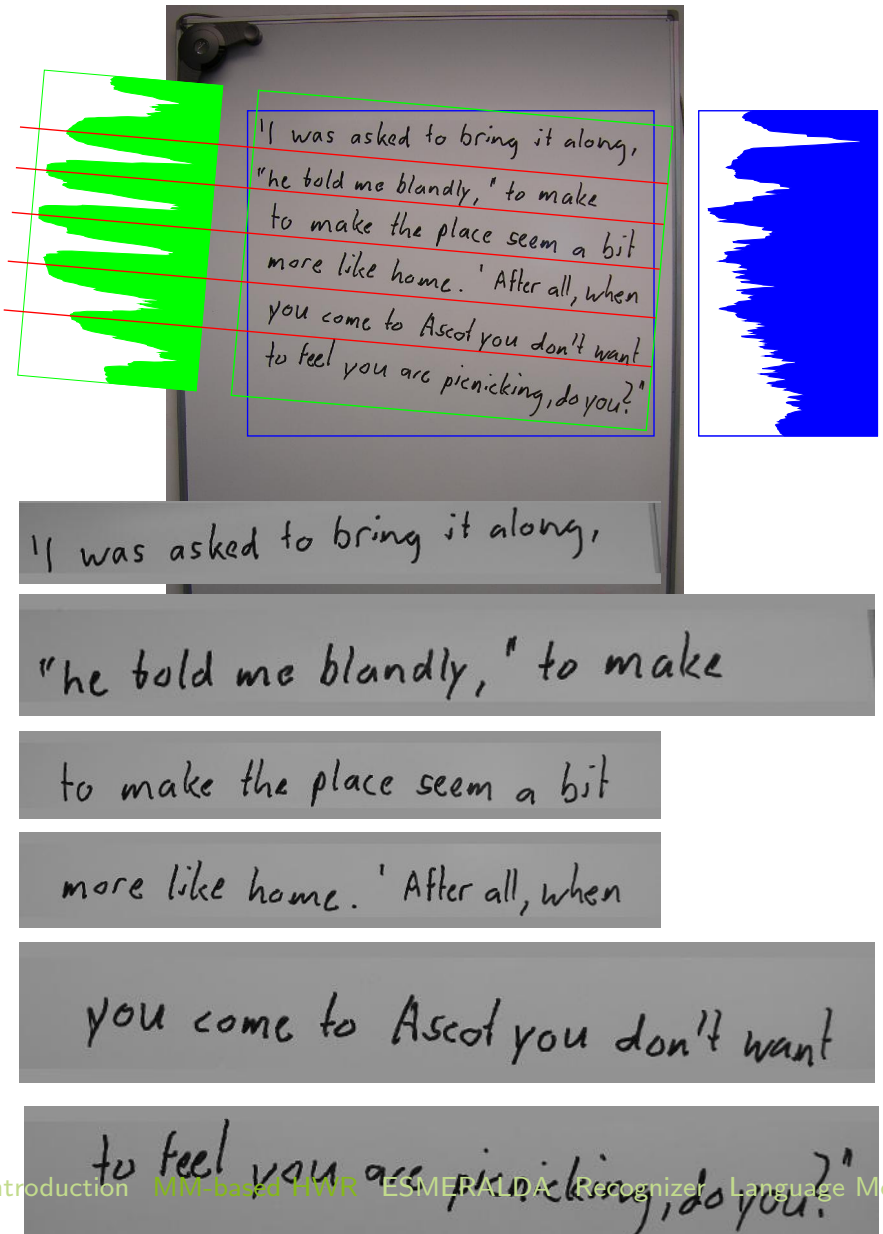
Preprocessing I

Line Extraction

Basis: Document containing handwritten text

Principle Method:
(cf. e.g. [14, 1])

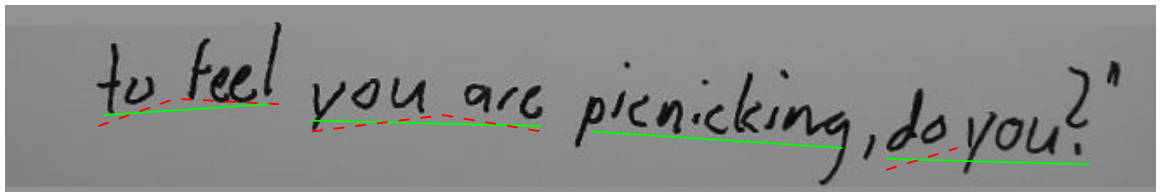
1. Find text regions (if necessary)
2. Correct orientation of text region (minimize entropy of horizontal projection histogram)
3. Extract text lines (segment at minima of projection histogram)



(IAM-OnDB image courtesy of H. Bunke, University of Bern)

Preprocessing II

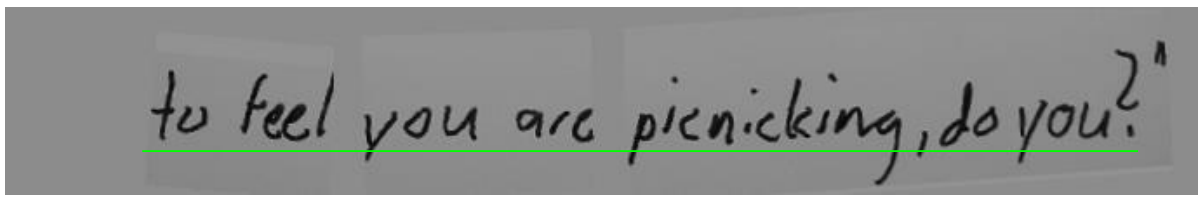
Baseline Estimation:



Potential method:

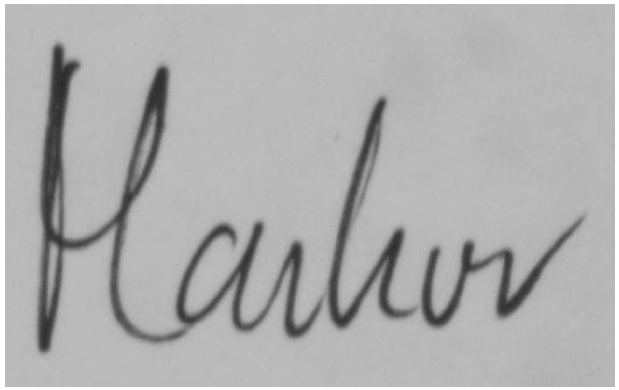
- Initial estimate based on horiz. projection histogram
- Iterative refinement and outlier removal (cf. [2, 12])

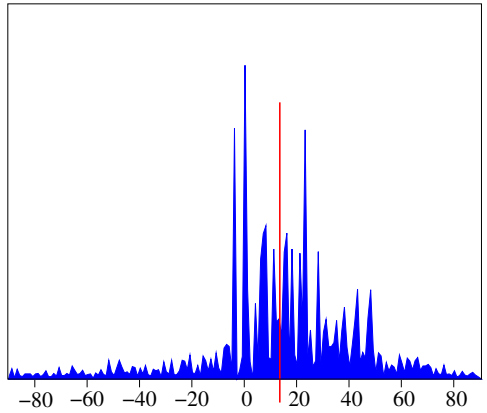
Skew and Displacement Correction:



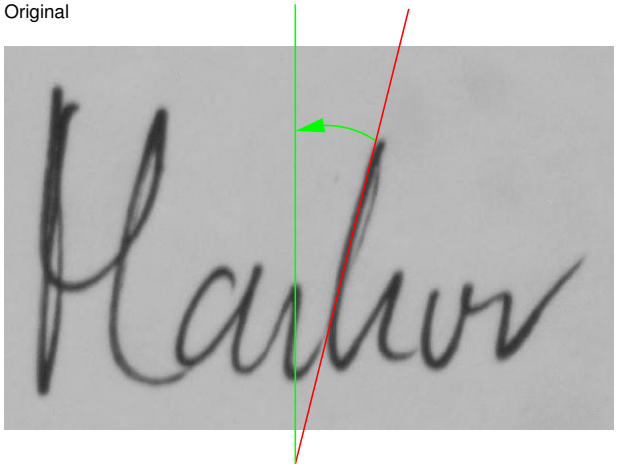
Preprocessing III

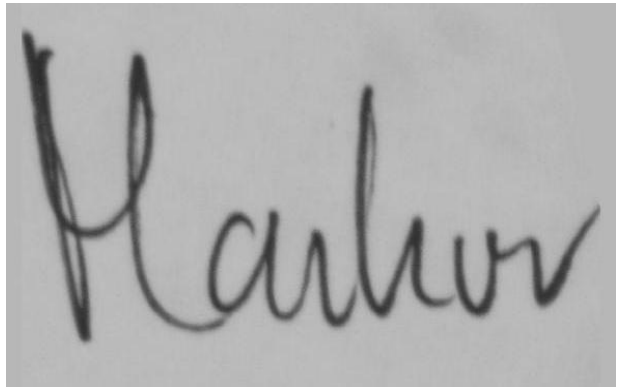
Slant estimation: E.g. via mean orientation of edges obtained by Canny operator (cf. e.g. [14])





Slant normalization (by applying a shear transform)





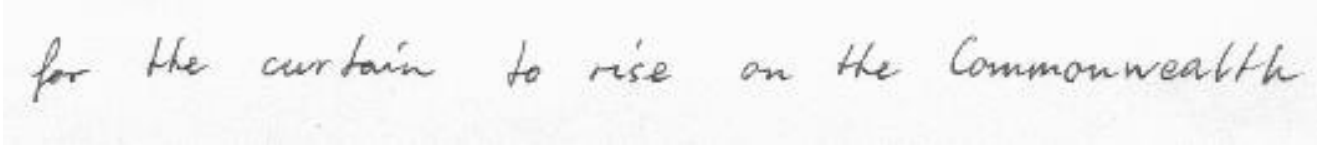
Preprocessing IV

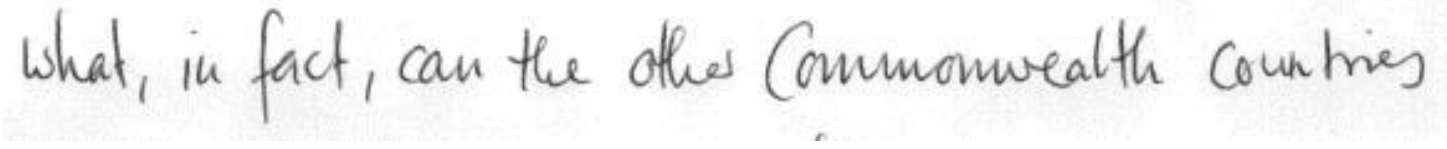
Note: Depending on writer and context script might largely vary in size!

Methods for size normalization:

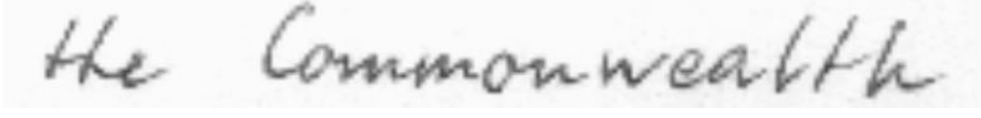
- "manually", heuristically, to predefined width/height???
- depending on estimated core size (← estimation crucial!)
- depending on estimated character width [9]

Original text lines (from IAM–DB)





Results of size normalization (avg. distance of contour minima)



Serialization: The Sliding Window Method

Problem: Data is two-dimensional, images of writing!

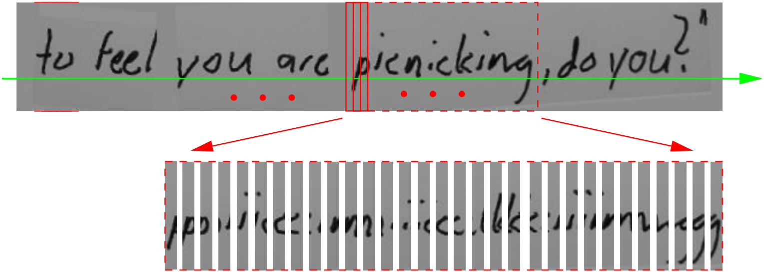
↯ No chronological structure inherently defined!

Exception: Logical sequence of characters within texts

Solution: Sliding-window approach

First proposed by researchers at Daimler-Benz Research Center, Ulm [3], pioneered by researchers at BBN [13]

- Time axis runs in writing direction / along baseline
- Extract small overlapping analysis windows



[Frames shown are for illustration only but actually too large!]

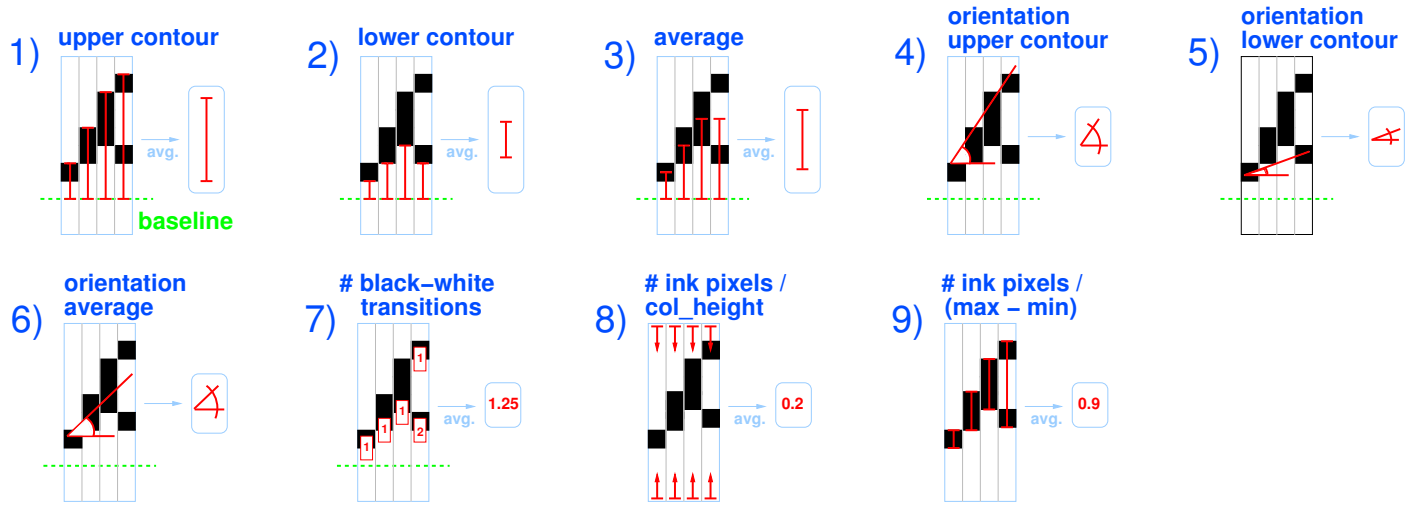
Feature Extraction

Basic Idea: Describe appearance of writing within analysis window

- No "standard" approaches or feature sets
- No holistic features used in HMM-based systems

Potential Methods:

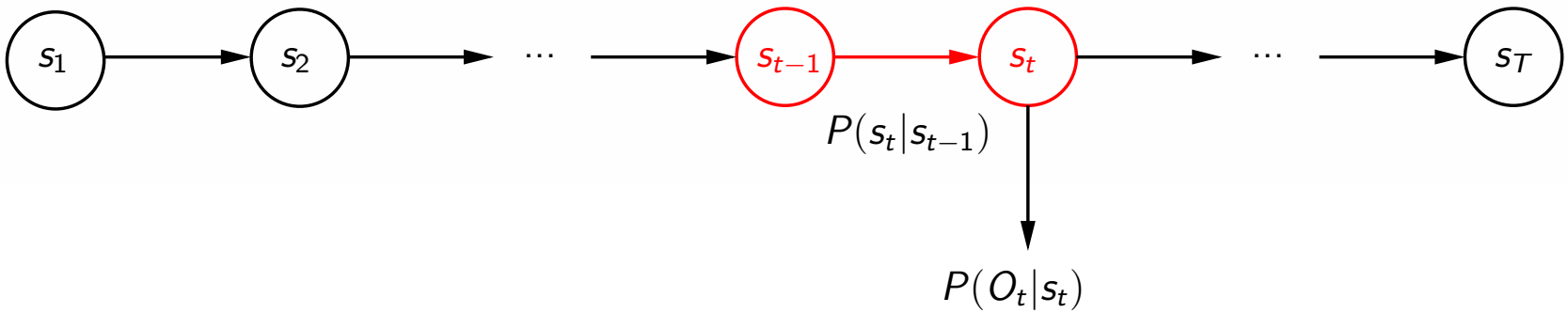
- (For OCR) Local analysis of gray-value distributions (cf. e.g. [1])
- Salient elementary geometric shapes (e.g. vertices, cusps)
- Heuristic geometric properties (cf. e.g. [15])



Additionally: Compute dynamic features
(i.e. discrete approximations of temporal derivatives, cf. e.g. [5])

Overview

Hidden Markov Models: Two-Stage Stochastic Processes



1. Stage: discrete stochastic process $\approx$ "probabilistic" finite state automaton

   stationary: Process independent of absolute time t

   causal: Distribution s_t only dependent on previous states

   simple: *particularly* dependent only on *immediate* predecessor state ($\hat{=}$ first order)

   $\Rightarrow P(s_t|s_1, s_2, \ldots s_{t-1}) = P(s_t|s_{t-1})$

2. Stage: Output O_t generated for every time t depending on current state s_t

   $\Rightarrow P(O_t|O_1 \ldots O_{t-1}, s_1 \ldots s_t) = P(O_t|s_t)$

   Note: Only outputs can be observed $\Rightarrow$ hidden Markov model

Hidden-Markov-Models: Formal Definition

A Hidden-Markov-Model λ of *first order* is defined as:

- a finite set of states:

$$\{s|1 \leq s \leq N\}$$

- a matrix of state transition probabilities:

$$\mathbf{A} = \{a_{ij}|a_{ij} = P(s_t = j|s_{t-1} = i)\}$$

- a vector of start probabilities:

$$\boldsymbol{\pi} = \{\pi_i|\pi_i = P(s_1 = i)\}$$

- state specific output probability distributions:

$$\mathbf{B} = \{b_{jk}|b_{jk} = P(O_t = o_k|s_t = j)\} \text{ (discrete case)}$$

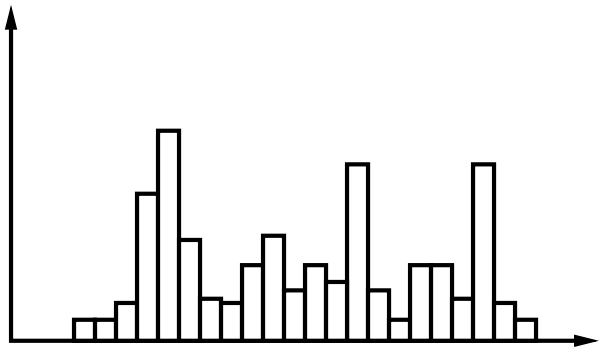
or

$$\{b_j(O_t)|b_j(O_t) = p(O_t|s_t = j)\} \text{ (continuous case)}$$

Modeling of Outputs

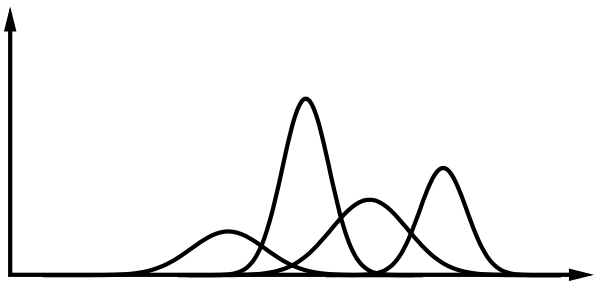
Discrete inventory of symbols: Very limited application fields

- ✓ Suited for discrete data only (e.g. DNA)
- ↯ Inappropriate for non-discrete data – use of vector quantizer required!



Continuous modeling: Standard for most pattern recognition applications processing sensor data

- ✓ Treatment of real-valued vector data (i.e. vast majority of "real-world" data)
- ✓ Defines distributions over $\mathbb{R}^n$



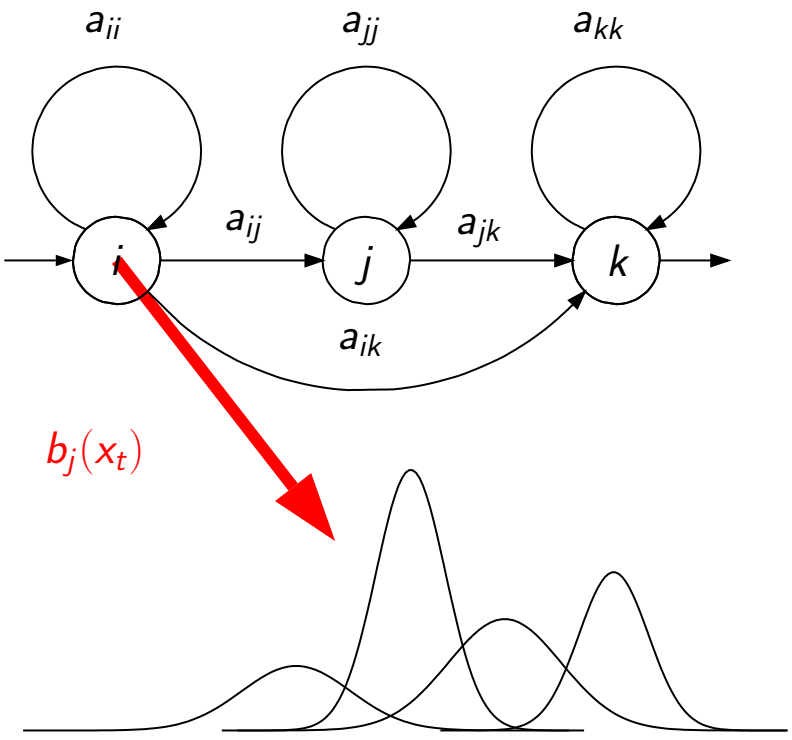
Problem: No general parametric description

Procedure: Approximation using mixture densities

$$p(\mathbf{x}) \triangleq \sum_{k=1}^{\infty} c_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{C}_k)$$

$$\approx \sum_{k=1}^{M} c_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{C}_k)$$

Modeling of Outputs – II

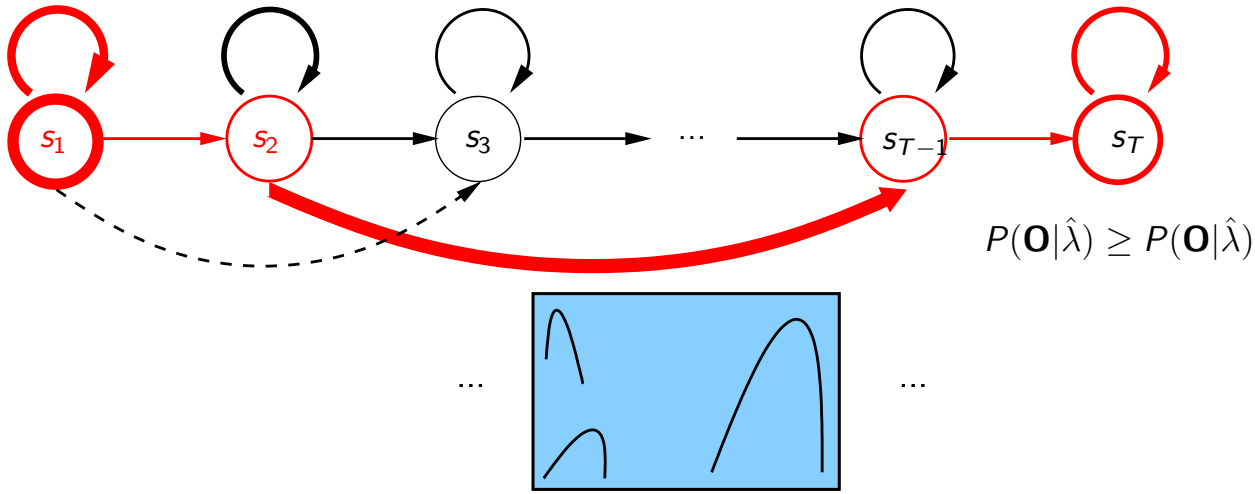


Mixture density modeling:

- Base Distribution?
  ⇒ Gaussian Normal densities
- Shape of Distributions
  (full / diagonal covariances)?
  ⇒ Depends on pre-processing of the data (e.g. redundancy reduction)
- Number of mixtures?
  ⇒ Clustering (. . . and heuristics)
- Estimation of mixtures?
  ⇒ e.g. Expectation-Maximization

Note: In HMMs integrated with general parameter estimation

Usage Concepts for Hidden-Markov-Models



Assumption: Patterns observed are generated by stochastic models which are comparable *in principle*

Scoring: *How well does the model describe some pattern?*
$\rightarrow$ Computation of the production probability $P(\mathbf{O}|\lambda)$

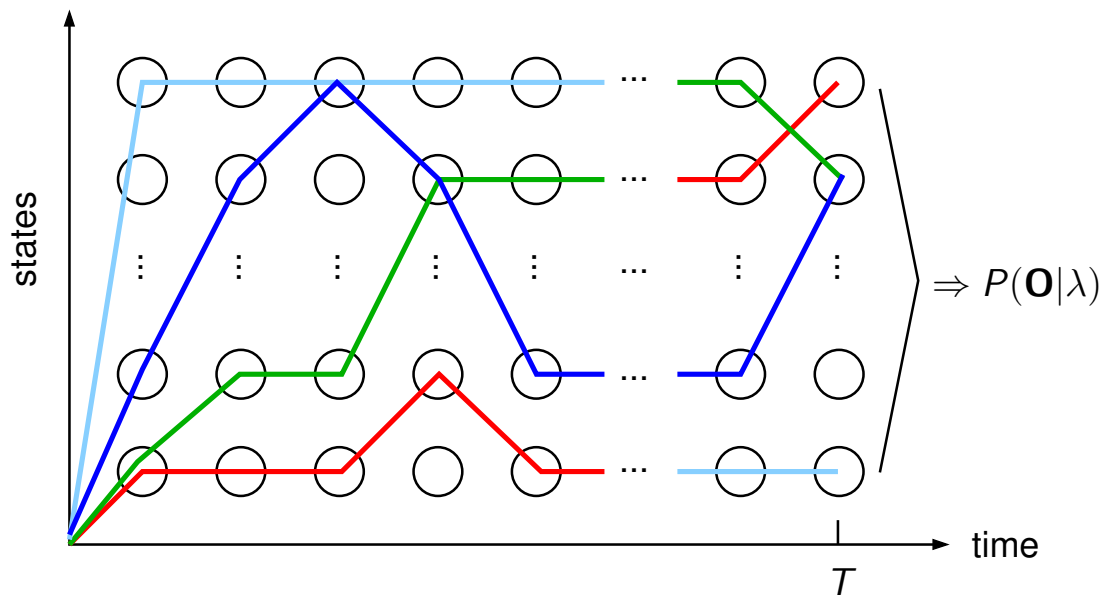
Decoding: *What is the "internal structure" of the model?* ($\hat{=}$ "Recognition")
$\rightarrow$ Computation of the optimal state sequence
$\mathbf{s}^* = \underset{\mathbf{s}}{\operatorname{argmax}}\, P(\mathbf{O}, \mathbf{s}|\lambda)$

Training: *How to determine the "optimal" model?*
$\rightsquigarrow$ Improvement of a given model λ with $P(\mathbf{O}|\hat{\lambda}) \geq P(\mathbf{O}|\lambda)$

The Production Probability

Wanted: Assessment of HMMs' quality for describing statistical properties of data

Widely used measure: *Production probability* $P(\mathbf{O}|\lambda)$ that observation sequence $\mathbf{O}$ was generated by model λ – along an arbitrary state sequence



- Naive computation infeasible: Exponential complexity $O(TN^T)$

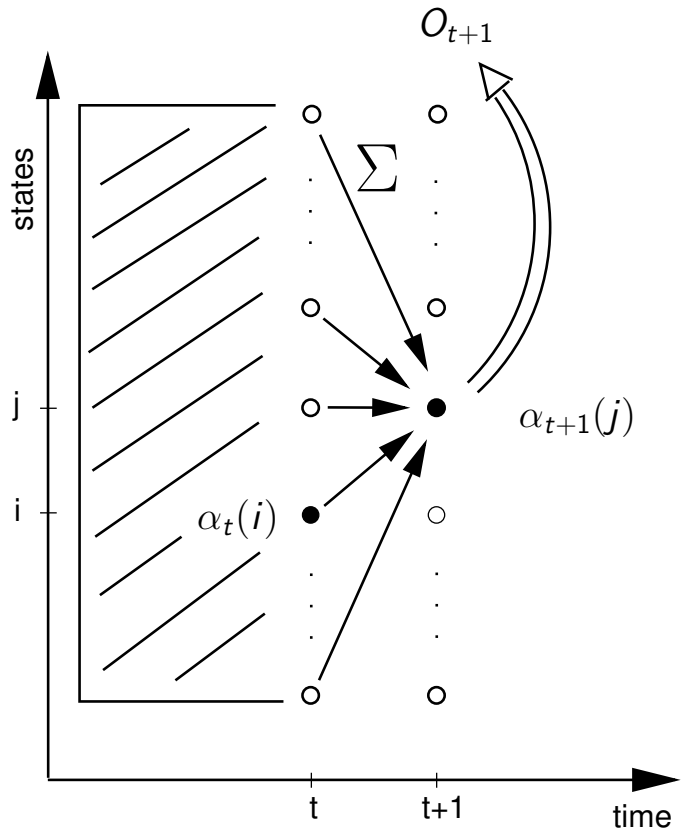
The Production Probability: The Forward-Algorithm

More efficient: Exploitation of the Markov-property, i.e. the "finite memory"
⇒ "Decisions" only dependent on immediate predecessor state

Let:
$\alpha_t(i) = P(O_1, O_2, \ldots O_t, s_t = i|\lambda)$
(*forward variable*)

1. $\alpha_1(i) := \pi_i b_i(O_1)$
2. $\alpha_{t+1}(j) := \left\{ \sum_{i=1}^{N} \alpha_t(i) a_{ij} \right\} b_j(O_{t+1})$
3. $P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

✓ Complexity: $O(TN^2)$!
(vs. $O(TN^T)$ from naive computation)



Note: There exists an analogous *Backward-Algorithm* required for parameter estimation.

Decoding

Problem: Global production probability $P(\mathbf{O}|\lambda)$ not sufficient for analysis if individual states are associated to meaningful segments of data

$\Rightarrow$ (Probabilistic) Inference of optimal state sequence $\mathbf{s}^*$ necessary

Maximization of posterior probability:

$$\mathbf{s}^* = \underset{\mathbf{s}}{\operatorname{argmax}}\, P(\mathbf{s}|\mathbf{O}, \lambda)$$

Bayes' rule:

$$P(\mathbf{s}|\mathbf{O}, \lambda) = \frac{P(\mathbf{O}, \mathbf{s}|\lambda)}{P(\mathbf{O}|\lambda)}$$

$P(\mathbf{O}|\lambda)$ irrelevant (constant for fixed $\mathbf{O}$ and given λ), thus:

$$\mathbf{s}^* = \underset{\mathbf{s}}{\operatorname{argmax}}\, P(\mathbf{s}|\mathbf{O}, \lambda) = \underset{\mathbf{s}}{\operatorname{argmax}}\, P(\mathbf{O}, \mathbf{s}|\lambda)$$

Computation of $\mathbf{s}^*$: *Viterbi-Algorithm*

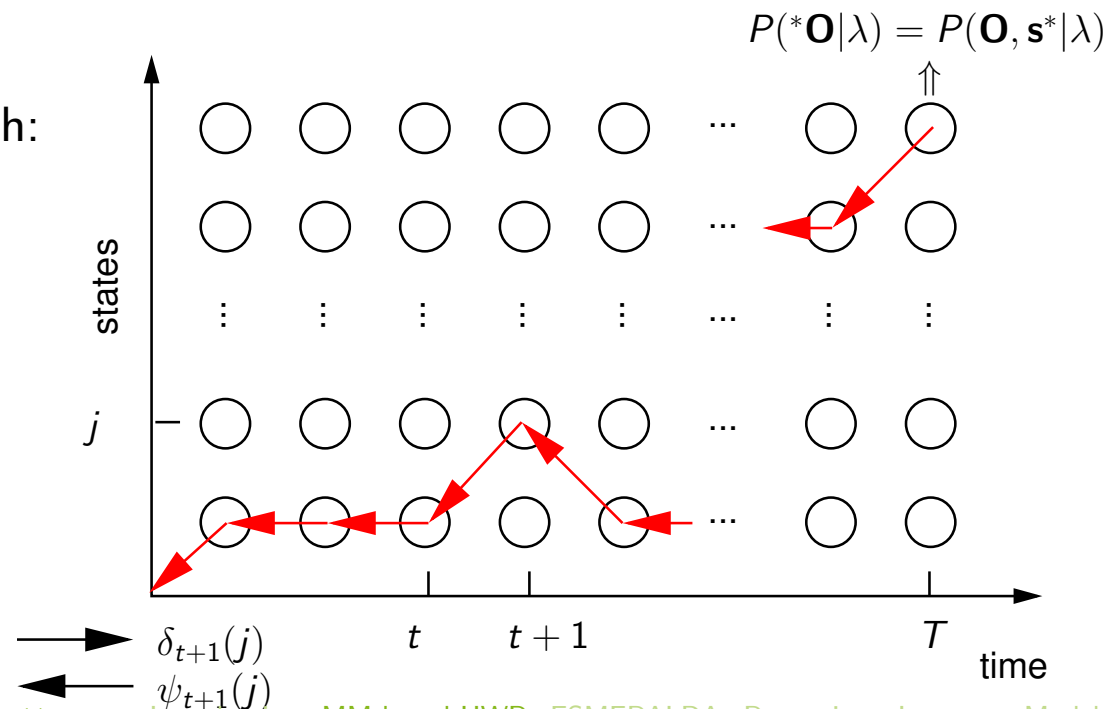
The Viterbi Algorithm

. . . inductive procedure for efficient computation of $\mathbf{s}^*$ exploiting Markov property

Let: $\delta_t(i) = \max_{s_1, s_2, \ldots s_{t-1}} P(O_1, O_2, \ldots O_t, s_t = i|\lambda)$

1. $\delta_1(i) := \pi_i b_i(O_1)$ $\qquad \psi_1(i) := 0$
2. $\delta_{t+1}(j) := \max_i(\delta_t(i)a_{ij})b_j(O_{t+1})$ $\qquad \psi_{t+1}(j) := \operatorname{argmax}_i \ldots$
3. $P^*(\mathbf{O}|\lambda) = P(\mathbf{O}, \mathbf{s}^*|\lambda) = \max_i \delta_T(i)$
   $s_T^* := \operatorname{argmax}_j \delta_T(j)$
4. Back-tracking of optimal path:
   $s_t^* = \psi_{t+1}(s_{t+1}^*)$

- ✓ Implicit *segmentation*
- ✓ Linear complexity in time
- STOP Quadratic complexity w.r.t. #states



Parameter Estimation – Fundamentals

Goal: Derive optimal (for some purpose) statistical model from sample data

Problem: No suitable analytical method / algorithm known

"Work-Around": Iteratively improve existing model λ
$\Rightarrow$ Optimized model $\hat{\lambda}$ better suited for given sample data

General procedure: Parameters of λ subject to growth transformation such that

$$P(\mathbf{O}|\hat{\lambda}) \geq P(\mathbf{O}|\lambda)$$

1. "Observe" model's actions during generation of an observation sequence
2. Original parameters are replaced by relative frequencies of respective events

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions out of state } i}$$

$$\hat{b}_i(o_k) = \frac{\text{expected number of outputs of } o_k \text{ in state } i}{\text{total number of outputs in state } i}$$

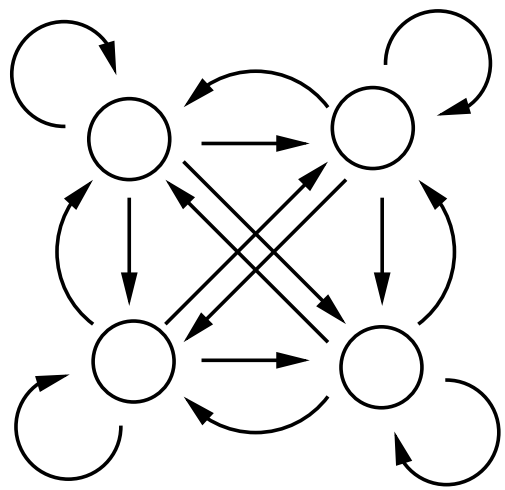
Limitation: Initial model required!

Configuration of HMMs: Topologies

Generally: Transitions between arbitrary states possible within HMMs ...
potentially with arbitrarily low probability

Topology of an HMM: Explicit representation of allowed transitions
(drawn as edges between nodes/states)

Any transition possible
$\Rightarrow$ *ergodic* HMM



Observation: Fully connected HMM does usually not make sense for describing chronologically organized data

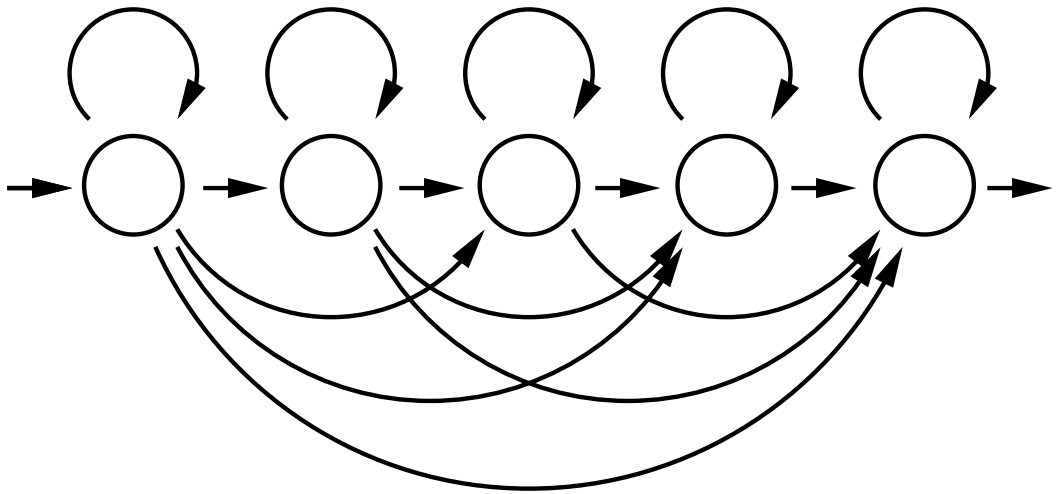
↯ "backward" transitions would allow arbitrary repetitions within the data

Configuration of HMMs: Topologies II

Idea: Restrict potential transition to *relevant* ones!

... by omitting irrelevant edges / setting respective transition probabilities to "hard" zeros (i.e. never modified!)

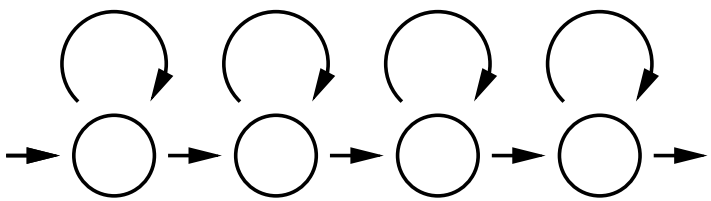
Structures/Requirements for modeling chronologically organized data:



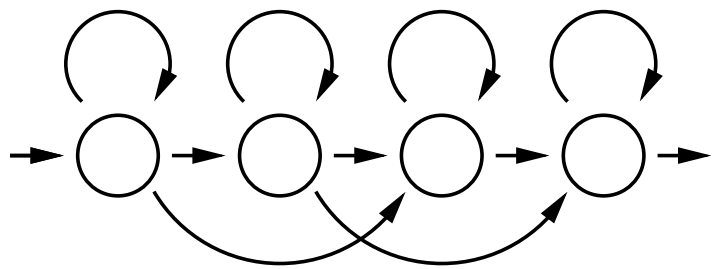
- "Forward" transitions (i.e. progress in time)
- "Loops" for modeling variable durations of segments
- "Skips" allow for optional/missing parts of the data
- Skipping of one or multiple states forward

Configuration of HMMs: Topologies III

Overview: The two most common topologies for handwriting (and speech) recognition:



linear HMM



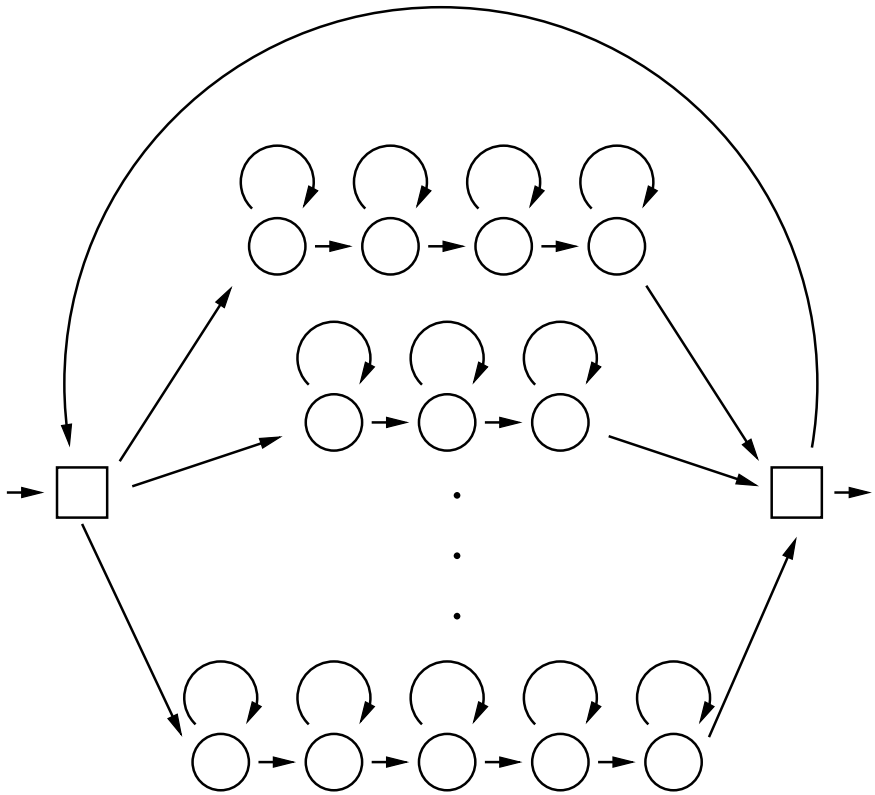
Bakis-type HMM

Note: General left-to-right models (allowing to skip any number of states forward) are not used in practice!

Configuration of HMMs: Compound Models

Goal: Segmentation

- Basic units: Characters
  [Also: (sub-)Stroke models]
- Words formed by concatenation
- Lexicon = parallel connection
  [Non-emitting states merge edges]
- Model for arbitrary text
  by adding loop



⇒ Decoding the model produces segmentation
(i.e. determining the optimal state/model sequence)

Overview

n-Gram Models: Definition

Goal: Calculate $P(\mathbf{w})$ for given word sequence $\mathbf{w} = w_1, w_2, \ldots, w_k$

Basis: n-Gram model = Markov chain model of order $n-1$

Method: Factorization of $P(\mathbf{w})$ applying Bayes' rule according to

$$P(\mathbf{w}) = P(w_1)P(w_2|w_1)\ldots P(w_T|w_1,\ldots,w_{T-1}) = \prod_{t=1}^{k} P(w_t|w_1,\ldots,w_{t-1})$$

Problem: Context dependency increases arbitrarily with length of symbol sequence
$\Rightarrow$ Limit length of the "history"

$$P(\mathbf{w}) \approx \prod_{t=1}^{T} P(\underbrace{w_t \mid w_{t-n+1},\ldots,w_{t-1}}_{n \text{ symbols}})$$

Result: Predicted word w_t and *history* form an n-tuple $\Rightarrow$ n-gram ($\hat{=}$ *event*)

$\Rightarrow$ **n-gram models** (typically: $n = 2 \Rightarrow$ bi-gram, $n = 3 \Rightarrow$ tri-gram)

n-Gram Models: Parameter Estimation

Basis: Relative frequency of events (i.e. n-gram counts $c(\mathbf{y}z)$)

Problem:

- Not *some* but *most* n-gram counts will be **zero**!
- It must be assumed that this is only due to insufficient training data!

$\Rightarrow$ estimate *useful* $P(z|\mathbf{y})$ for $\mathbf{y}z$ with $c(\mathbf{y}z) = 0$

Question: *What estimates are "useful"?*

- small probabilities!, smaller than *seen* events? $\rightarrow$ mostly not guaranteed!
- specific probabilities, not uniform for all unseen events

Solution:

1. Modify n-gram counts and gather "probability mass" for *unseen events*
2. Redistribute *zero-probability* to *unseen events* according to a more general distribution ($\hat{=}$ *smoothing* of empirical distribution)

   Note: Usually "more general" $\hat{=}$ $(n-1)$-gram model

Overview

Markov Models for HWR: Summary

- ✓ Stochastic model for sequential patterns with high variability
- ✓ Efficient algorithms for training and decoding exist
- ✓ Powerful combination of appearance model (i.e. writing $\hat{=}$ HMM) and language model ($\hat{=}$ n-gram model) possible
- ✓ Segmentation and classification are performed in an integrated manner: Segmentation free recognition
- ↯ Model structure (esp. for HMMs) needs to be pre-defined.
- ↯ Initial model required for training (of HMMs)
- ↯ Considerable amounts of training data necessary

"There is no data like more data!"

[Robert L. Mercer, IBM]

Further Reading

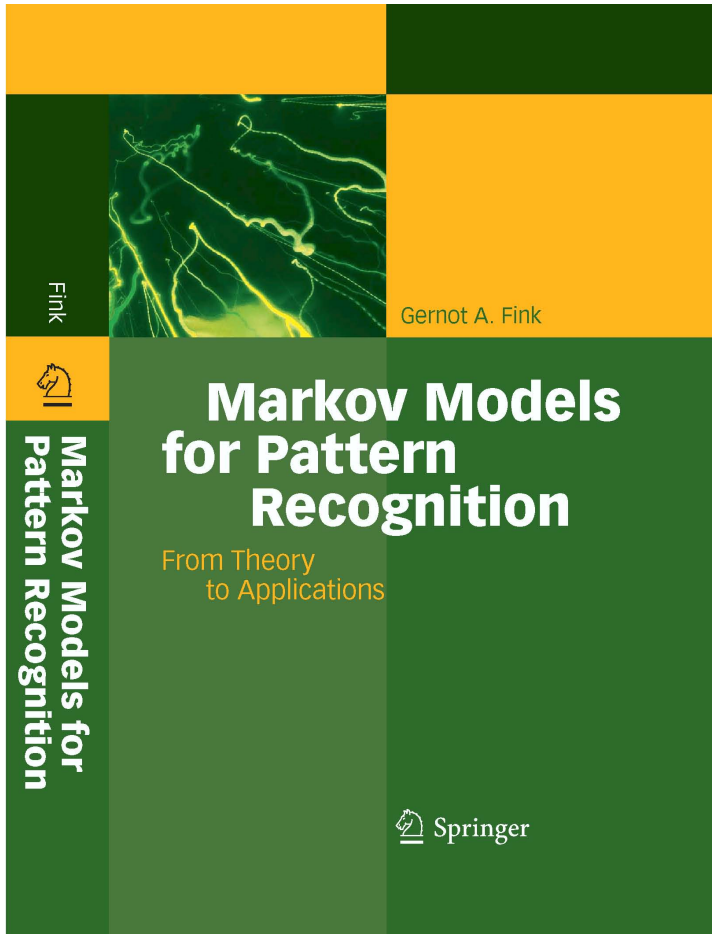
Textbook: Gernot A. Fink: *Markov Models for Pattern Recognition*. Springer, Berlin Heidelberg, 2008.

- ✓ (Electronic) Inspection copy available!
- ✓ Conference discount: 20%!

Survey Article: Thomas Plötz & Gernot A. Fink: Markov Models for Offline Handwriting Recognition: A Survey. *IJDAR*, 12(4):269–298, 2009.

- ✓ Open access publication!

Brand new: Thomas Plötz & Gernot A. Fink: *Markov Models for Handwriting Recognition*, SpringerBriefs in Computer Science, Springer, 2011.



Overview

What is ESMERALDA?

Framework for developing HMM-based pattern recognition systems

Developed/Maintained at TU Dortmund, Germany (cf. [6, 8])

Supports: (primarily)

- (SC)HMMs of different topologies
  (with user-definable internal structure)
- Incorporation of *n*-gram models
  (for long-term sequential restrictions)
- Gaussian mixture models (GMMs).

Used for numerous projects within:

- **Handwriting Recognition**,
- Automatic Speech Recognition,
- Analysis of biological sequences,
- Music analysis.

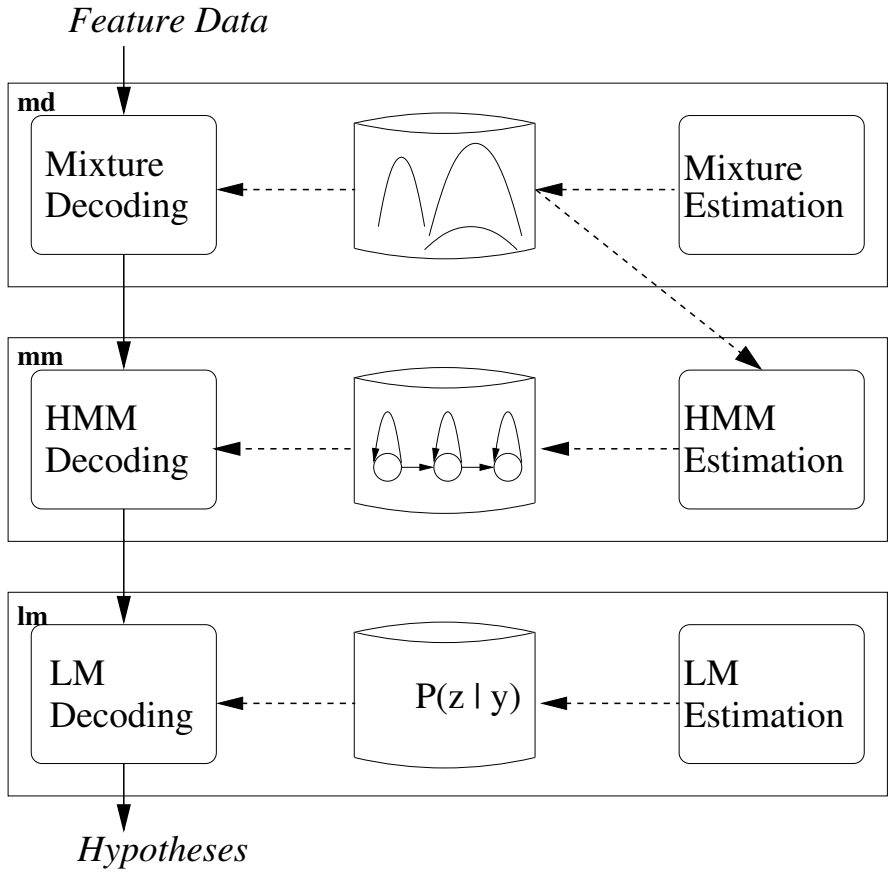
Availability: Open source software (LGPL) sourceforge.net/projects/esmeralda

Overview

ESMERALDA – System Architecture

Goal: Put together tractable set of conceptually simple yet powerful techniques in an integrated development environment

Modules: Libraries with API (each) and stand-alone programs for manipulating appropriate models and associated data



ESMERALDA – Modules

- *core modules*: estimation, adaptation, evaluation of Markov models

  md: *mixture densities*
  mm: *hidden Markov models*
  lm: *n-gram models* (language models)

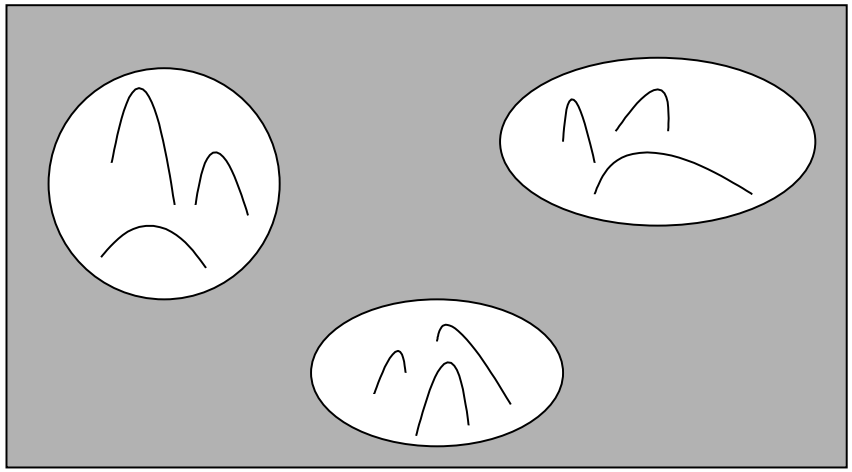
- *domain specific modules*: adoption to practical tasks

  dsp: feature extraction for speech recognition applications (`dsp_vad, dsp_fex`)
  isr: *incremental (speech) recognizer*
  im: (general) *image processing*
  pen: *handwriting recognition*
  seq: *biological sequence analysis*

- *auxiliary modules*: basic functionality for integrated development environment

  rs: *runtime system*
  mx: *math extension* (esp. linear algebra)
  dsp: *digital signal processing*
  fx: *feature manipulation*
  ev: *evaluation tools*

Mixture Densities – Overview

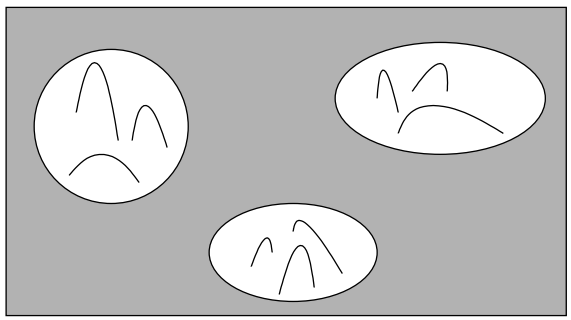


md: Module for efficient and robust estimation and evaluation of GMMs

- ✓ Unsupervised mixture estimation (based on k-means, Lloyd, LBG)
- ✓ Robust model training based on EM-Algorithm
- ✓ Maximum a-posteriori (MAP) mixture adaptation
- ✓ Efficient two-stage decoding
- ✓ Estimation and application of linear feature space transforms (PCA, LDA)

Mixture Densities – Interfaces

External representation: Collection of codebooks $\hat{=}$ codelibrary (`*.cl`)



→ contains (multiple) codebooks

→ Gaussian densities

- mean vectors $\vec{\mu}_i$
- covariance matrices $\mathbf{C}_i$
- prior probabilities p_i

→ classifier(s)

- extended parameter representation for fast evaluation
- multi-stage classification using subsets of feature vectors each

→ optional: linear feature transformation

- mean vectors and transformation matrix per codebook (LDA, PCA, . . .)
- "on-the-fly" transformation of feature vectors not requiring separate external representation

API: `man md_codebook ; md/codebook.h`

Mixture Densities – Tools

`md_k_means`: (also: `md_lbg`) vector quantization (k_means or LBG) and initialization of mixture densities in various configurations

`md_train`: mixture density training using EM in various configurations

`md_param`: "swiss-army knife" for codebook manipulation

- computation of codebook parameters from statistics (generated during training)
- ML or MAP parameter estimation
- computation of actual classifier parameters

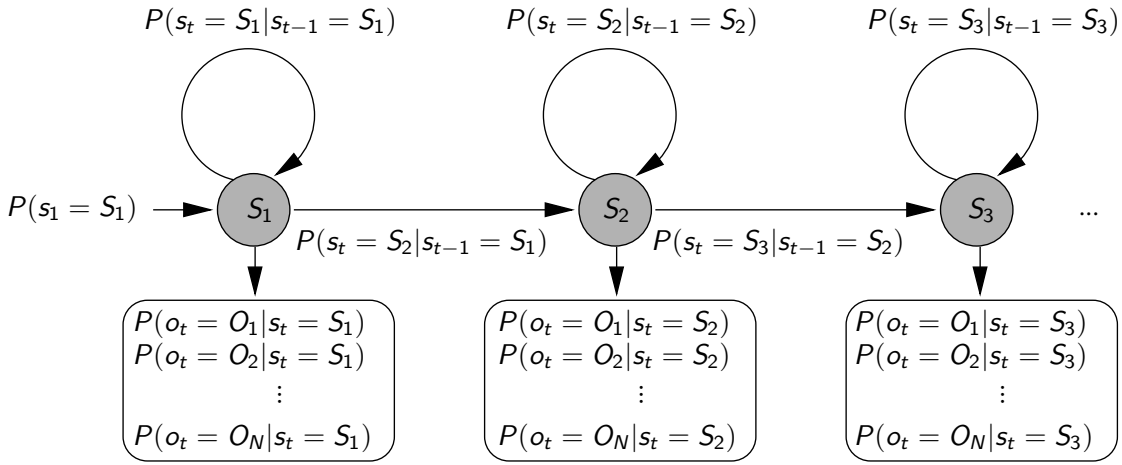
`md_classify`: performs mixture density classification – provides

- optimal mixture, i.e. codebook,
- optimal generalized mixture per sequence (GMM),
- optimal density per mixture, or
- scores for all densities in all codebooks

`md_filter`: (soon: deprecated) applies linear transformation to feature vectors

`md_init`: (soon: deprecated) estimates linear transformation on unlabeled features

Hidden Markov Models – Overview



mm: Module for definition, training, and evaluation of Hidden Markov Models

- ✓ Basic topologies (linear, left-right, Bakis, Profile, bounded left-right) and user-definable model structures
- ✓ Declarative configuration (EBNF) language for building structured models from elementary units – not specific to particular application domain
- ✓ Automatic model initialization
- ✓ Efficient Viterbi beam-search decoding
- ✓ Efficient training (Baum-Welch incl. forward-backward pruning)
- ✓ (Semi-)Supervised model adaptation – MLLR and MAP

Hidden Markov Models – Interfaces

External representation:

- codelibraries → mixture densities
- state files → transition probabilities and mixture weights (emissions)
- model files → named grouping of states with fix topology
- concepts → grammar based on elementary concepts ≡ models

Basics:

- trainable parameters: states/codelibraries – models/concepts: declarative
- models correspond to HMM with "own" parameters
- concepts defined by combination of volatile HMMs – parameters only copies

API:

- `man mm_concept ; mm/concept.h`,
- `man mm_model ; mm/model.h`,
- `man mm_state ; mm/state.h`

Hidden Markov Models – Tools (1)

`mm_init`: creation of elementary models with given name, topology, and variable number of states + initial estimation of model parameters

parameters:

- initial codelibrary [↗ md_k_means]
- specification of model topology
- annotated sample set
- (heuristic) strategy for model creation

`mm_train`: parameter estimation for HMMs – single training iteration

parameters:

- existing HMM (codelibrary, definition of states, models, and concepts)
- training set with annotation (e.g. on word or character level)
- output: raw statistics for parameter estimation
  (for both densities [$\hat{=}$ codebooks] and states [↗ mm_param])

Hidden Markov Models – Tools (2)

`mm_param`: creation of model parameters (states / models) from raw training statistics (so-called accumulators)

parameters:

- existing HMM without codelibrary (states, models, concepts)
- training statistics (`*.accu` – output from [↗ mm_train]
- output: updated state and model definitions

`mm_align`: model decoding (with optional bi-gram language model)

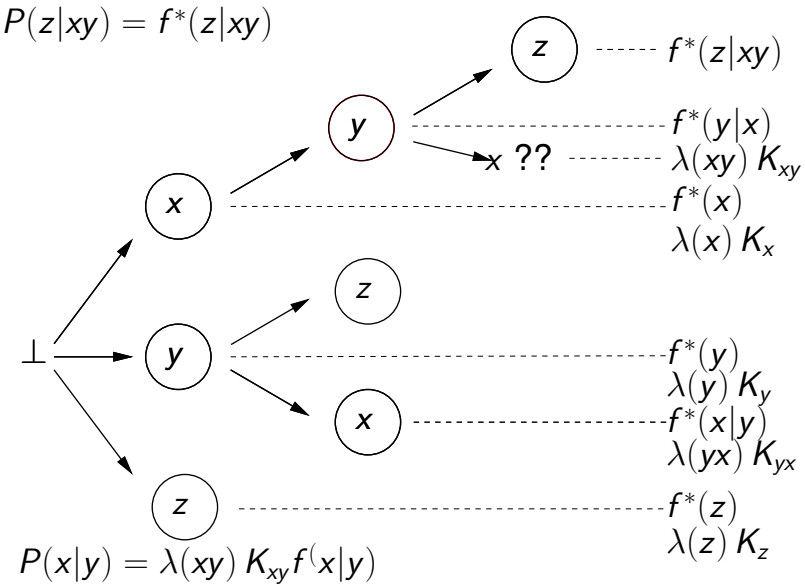
parameters:

- existing HMM (codelibrary, states, models, concepts)
- specification of recognition task (i.e. task *model*)
- validation / test set
- output: recognition hypotheses, optionally incl. detailed segmentation and score

`mm_adapt`: (offline) MLLR adaptation of HMMs

parameters similar to [↗ mm_align]

n-gram Models – Overview



Im: Module for estimation and evaluation of statistic language models

- ✓ Memory efficient estimation of n-gram statistics
- ✓ n-Gram estimation based on different smoothing techniques (e.g. absolute discounting and backing-off/interpolation)
- ✓ Efficient evaluation of long-span models (via tree representation)

n-gram Models – Interfaces (1)

External representations:

- Statistics (relative frequencies / counts, `*.count`)
  - contain counts $c(\vec{y}z)$ for n-grams of arbitrary given length
  - lexicon of symbols contained
- Models (`*.lm`)
  - contain n-gram scores and parameters
  - efficient representation in combined prefix-suffix tree

API: `lm/count.h; lm/ngram.h`

n-gram Models – Tools

`lm_count`: analysis of sample sets – determines lexicon of a text and $c(\vec{y}z)$

parameters:

- n-gram length
- sample set (ASCII text)
- output: relative counts (`*.count`)

`lm_param`: estimation of language model parameters

parameters:

- method for gaining probability mass (discounting)
- count data [↗ lm_count]
- output: language model

`lm_perp`: language model evaluation on sample data – calculation of perplexity

parameters:

- language model (`*.lm`) [↗ lm_param]
- sample data (ASCII text)
- output: perplexity

Auxiliary Modules – (Very) Brief Overview

Required for practical applications – *Integrated Development Einvironment*

`rs`: library for convenient (and unified) memory management, time measurement, basic data types, I/O

`mx`: library for efficient linear algebra computations, i.e. vector- and matrix handling

`dsp`: library for basic digital signal processing tasks plus tools for feature extraction (speech)

`fx`: library and tools for convenient and efficient manipulation of feature vectors

`ev`: tools for evaluation of pattern recognition experiments (segmentation and detection results)

Also: Integrated recognizer (`isr`) that combines HMMs and *n*-gram models into unified framework for "live" (speech) recognition

Example: Image Processing and HWR

`im:` Module that covers basic image processing functionality – `im_filter`

- standard operations: binarization, Canny, convolution . . . and MANY more
- warping operations: scaling, rotation, apply general affine transformation
- color transformation: RGB, HSV, HSI, YUV
- normalization: color const., contrast, etc.

`pen:` Supporting module for handwriting recognition

- Line extraction (`pen_rowextract`)
- Skew and displacement correction (`pen_skew`, `pen_displace`)
- Slant normalization (`pen_slant`)
- Size normalization (`pen_scale`)
- Feature extraction (`pen_fextract`)

. . . Basis for TU Dortmund handwriting recognition system ✓

Overview

Model Representation I

(Semi-)Continuous HMMs / 4 Components:

1. Set of densities (component densities for mixtures $\hat{=}$ "codebook")
   id, *type*, *mean vector*, *covariance matrix*
   $\Rightarrow$ `*.cl`, e.g.: `icdar2011.0.cl`
2. Set of states
   id, *transition probabilities*, *mixture weights*
   $\Rightarrow$ `*.state`, e.g.: `icdar2011.1.state`
3. Basic modeling units defining (local) topology
   name, *type/topology* (`Linear`, `Bakis`, ...), *list of assoc. states*
   $\Rightarrow$ `*.model`, e.g.: `icdar2011.0.model`
4. Compound models, i.e. higher-level model structures
   Defined using a context-free grammar
   $\Rightarrow$ `*.def`, e.g.: `icdar2011.swu.def`

Example: Word and task models

```
/* word model = concatenation of character models */
Tutorial := /T/ /u/ /t/ /o/ /r/ /i/ /a/ /l/ ;

/* task model = by looped parallel connection of characters */
<TASK> %= { /a/ | /b/ | ... /z/ | /A/ | ... /Z/ | <space> } + ;
```

Model Representation: Codebooks

Example: Codebook definition

```
# code-library generated by md_param
# type, # features, # codebooks
GaussDiag 18 1
# definition for codebook # 0
[...]
# definition for class # 0
# class name, prior probability
'' 0.20891
# class type is GaussDiag
# mean vector
7.48846e-05 2.00167e-05 0 2.04051e-05 0 1.96282e-05 0 [...]
# diagonal of covariance matrix
3.13068e-05 7.48686e-05 0 7.47434e-05 0 7.50068e-05 0 [...]
# definition for class # 1
[...]
```

- Created by `md_k_means` from *unlabelled* data
- Can be re-estimated / optimized on labelled data by `md_train`
- Will be *implicitly* re-estimated during HMM training

Model Representation: States

Example: Definition of model states

```
0 2:[0.857 0.142 ] 0/1023:[0.5811 0.0056 0 0.0018 0 0 0 ... ]
1 2:[0.775 0.224 ] 0/1023:[0 0.000948704 0 0 0 0 0 ... ]
2 2:[0.886 0.113 ] 0/1023:[0.0005 0 0 0 0 0 0 ... ]
3 2:[0.858 0.141 ] 0/1023:[0.4986 0.0130 0 3.107e-06 0 0 0 ... ]
[...]
```

- (First) Created by `mm_init` during HMM initialization based on heuristic procedures for allocating states to models (most useful for linear/Bakis topologies: min. unit length $\rightsquigarrow$ # states)
- Re-estimated during HMM training
- May be extended by special rules for defining (new!) compound models

Model Representation: Basic Models

Example: Basic character models

```
[-] Linear 4:[0 1 2 3 ]
/0/ Linear 4:[4 5 6 7 ]
/1/ Linear 4:[8 9 10 11 ]
/2/ Linear 4:[12 13 14 15 ]
[...]
/a/ Linear 4:[44 45 46 47 ]
/A/ Linear 4:[48 49 50 51 ]
[...]
/Z/ Linear 4:[316 317 318 319 ]
```

- Created by `mm_init` during HMM initialization based on *user-defined* template (only model names and topology)
- May be extended by special rules for defining (new!) compound models

Model Representation: Compound Models

Example: Word and task models

```
/* word model = concatenation of character models */
Tutorial := /T/ /u/ /t/ /o/ /r/ /i/ /a/ /l/ ;

/* task model = by looped parallel connection of characters */
<TASK> %= { /a/ | /b/ | ... /z/ | /A/ | ... /Z/ | <space> } + ;
```

- Created by you ;-)
  (possibly with the help of some tools)
- Used for defining training set annotations and specifications of recognition tasks
- Special rules may be used for extending model and state definitions (and thus define new trainable parameters)
- Otherwise completely declarative: Context free grammar for regular language
  (HMMs do not allow recursion)

Definition of Compound Models: Syntax

EBNF definition: regular expressions of the form <expression> ;

reference: <name>
If concept <name> exists, reference is created / error otherwise
Note: For all known models initial concepts with identical names are created.

sequence: $<\text{part}_1> <\text{part}_2> [<\text{part}_3> \ldots]>$
"series connection" of at least two models (or concepts)

alternative: $<\text{part}_1> | <\text{part}_2> [| <\text{part}_3> \ldots]>$
"parallel connection" of at least two models (or concepts)

loop: <part>+
feedback with at least one occurrence

optional: <part>?
maximum of one occurrence

synthesis: definition of permanent instances using

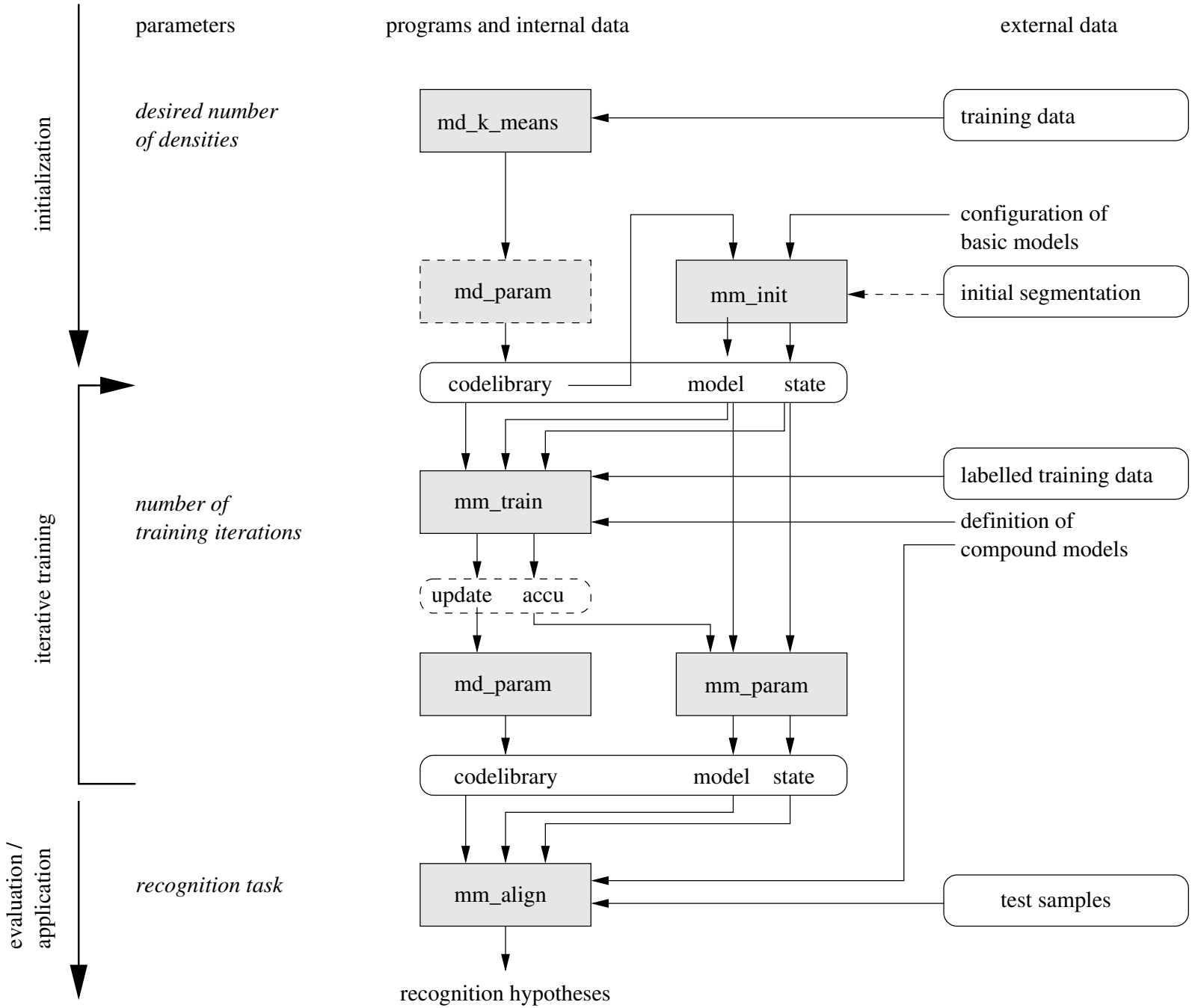
:= <name> := <expression>	<name> will be hypothesized
%= <name> %= <expression>	sequence of recognized parts hypothesized
<= <name> <= <expression>	states referred by <expression> copied (alignment)
== <name> == <expression>	same as for <= but aliases used

Overview

Development Cycle I

0. Data & Task Definition:
   - Preparation of the data
     Text extraction, line separation, preprocessing & normalization, serialization, feature extraction
     $\Rightarrow$ Feature vector sequences per text line
   - Model Definition
     - Which basic units exist / should be used?
     - Define compound models if necessary.
     - Annotate training data wrt. available models
1. Model Initialization:
   1. Initial codebook (for SC-HMM)
   2. Initial HMM
2. Parameter Re-Estimation:
   1. Apply HMM training (i.e. create improved model)
   2. Evaluate performance on validation set
   3. Redo until "convergence"
3. Rethink: Possibly re-consider design alternative (optional)

Development Cycle: Overview



Overview

Initializing the Codebook I

Resource	In theory	In ESMERALDA
Prerequisites:	feature representation	`iam-db.Corpus-a.lst`
Method:	k-means	`md_k_means`
Parameters:	dimension of feature vectors	`18`
	desired number of densities N	`1500`
	type of densities: Gaussians w. diagonal covariances	`-g`
Result:	set of densities (N or a few less)	`iam-db.0.cl`

Command line: Creating the initial codebook

```
md_k_means -g 18 1500 iam-db.Corpus-a.lst   > iam-db.0.cl \
                                            2> iam-db.0.cl.err
```

Remarks:

- Reports configuration/progress/actions — as all ESMERALDA tools — on `stderr`.
- Creates at most N densities — less if some clusters in the data are supported by too few samples ($5\times\#$ of parameters).
- Creates a *standard classifier* $\Rightarrow$ Sufficient for our purposes!

Initializing the Codebook II

What else can md_k_means do? Just ask: 'md_k_means -h'

```
usage: md_k_means [<option> ...] <dim> <classes> <ctrlfile>
        writes  the newly created codelibrary including a standard
                classifier for the full feature set to stdout
    where
        <dim>   specifies the dimension of input vectors
        <classes> specifies the number of classes to be created
        <ctrlfile> is the file holding the names of the data files
    valid options are
        -g      create only diagonal covariance matrices
        -p      use samples equidistantly distributed over the whole dataset(s)
                as initial prototypes
        [...]
        -T <fname> perform linear transformation of input data. File <fname>
                must contain transformation matrix and mean vector.
    valid expert options are (use carefully!)
        -f <n>  use 'n' as factor for min. required samples per class
                (<n> * #parameters) default: 5
        [...]
    valid general options are
        -h      display usage information
        -v      be more verbose (can be used multiple times)
        -V      display version information
```

Initializing the HMM I

... by means of a "flat start", i.e. without using annotations of the training data!

Resource	In theory	In ESMERALDA
Prerequisites:	feature representation	`iam-db.Corpus-a.lst`
Tool:	HMM initializer	`mm_init`
Parameters:	list of desired (character) models	`iam-db.model_frame`
	model topology for basic models	`Linear` (included in ↑)
	number of states per model	`8` (in '`-u 100x8`')
	creation method: *multiple states*	`ms`
Result:	initial HMM = initial states	`iam-db.0.state`
	+(initial) basic models	`iam-db.0.model`

Command line: Creating the initial model

```
mm_init   -M iam-db.0.model -S iam-db.mstat \
          iam-db.0.cl iam-db.model_frame empty.ini -u 100x4 ms \
          >iam-db.0.state \
          2> iam-db.0.state.err
```

Remarks:
- Desired number of states achieved via a trick: Statistics for unseen

Initializing the HMM II

What else can mm_init do? Just ask: 'mm_init -h' ...

```
usage: mm_init [<option> ...] <codebooks> <models> <init> [MC]
        writes  newly created state definitions to standard output
    where
        <codebooks>     specifies a file that holds definitions for one
                        or more codebooks (specify '-' if none is used)
        <models>        specifies a file that holds model frame definitions
        <init>          specifies a file with initialisation definitions
    valid options are
        -M <file> write (changed) models to <file>
        -S <file> write model statistic data to <file>
        [... several very special things ...]
    extensions can be configured using additional options and arguments for
        [MC]    model creation method (default: 'ss' = single state)
```

... and: 'mm_init 1 2 3 -h'

```
model creation usage: mm_init ... [<option> ...] <type>
    where
        <type>  models will be created using one of the following <type>s:
                'ss': single-state, i.e. one state per model
                'sts': single-state tied, i.e. one state used multiple times
                        (number given by minimum or low model duration)
                'ms': multiple-states, i.e. create multiple individual states
                        (number given by minimum or low model duration)
    valid model creation options are
        -u <n>x<t> treat unseen models as if <n> occurrances of <t> samples
                each were available (for semi-continuous models only)
        [...]
```

Re-Estimating HMM Parameters I

Resource	In theory	In ESMERALDA
Prerequisites:	training data annotated at word-level	`iam-db.Corpus-a.train`
Method:	Baum-Welch training	`mm_train -a none`
Parameters:	definition of word models	`iam-db.swu.def`
Input:	existing model	`iam-db.0.cl`, `iam-db.0.state`, `iam-db.0.model`
Result:	statistics for improved parameter estimates	`iam-db.1.accu`, `iam-db.1.update`

Command line: Re-estimating model parameters

```
mm_train  -q -b 200 -a none -U iam-db.1.update \
          iam-db.0.cl iam-db.0.state iam-db.0.model \
          ../../iam-db.swu.def \
          < ../iam-db.Corpus-a.train \
          > iam-db.1.accu 2> iam-db.1.accu.err
```

Remarks:

- Produces *only statistics* for computing new model parameters.

Re-Estimating HMM Parameters II

Resource	In theory	In ESMERALDA
Input:	parameter estimation statistics	`iam-db.1.accu`, `iam-db.1.update`
Tools:		`mm_param`, `md_param`
Parameters:	previous parameter set	`iam-db.0.cl`, `iam-db.0.state`, `iam-db.0.model`
Result:	re-estimated parameter sets	`iam-db.1.state`, `iam-db.1.cl`

Command lines: Creating parameter estimates from statistics

```
mm_param   -M iam-db.model \
           iam-db.0.state iam-db.0.model ../../iam-db.swu.def \
           <iam-db.1.accu >iam-db.1.state 2>iam-db.1.state.err
md_param   iam-db.0.cl \
           <iam-db.1.update >iam-db.1.cl 2>iam-db.1.cl.err
```

Remarks:

- By `md_param` also new models may be created if synthesizing operators were used to define compound model structures.

Decoding the Model I

Resource	In theory	In ESMERALDA
Prerequisites:	feature representation	`iam-db.Xeval-a.lst`
Method:	Viterbi beam-search	`mm_align`
Parameters:	specification of task model	`<CHARACTERS> +`
	definition of required compound models	`iam-db.swu.def, English.clex.def`
	beam witdh	`-b 75`
	word penalty	`-w 15`
Input:	model to be decoded	`iam-db.0.cl, iam-db.0.state, iam-db.0.model`
Result:	segmentation of data	`iam-db.Xeval-a-9-...chars`

Command line: Decoding the model

```
sed       's/$/ <CHARACTERS> + ;/' ../../iam-db.Xeval-a.lst | \
mm_align  -b 75 -w 15 iam-db.9.cl iam-db.9.state iam-db.model \
          ../../iam-db.swu.def ../../English.clex.def
          > iam-db.Xeval-a-9-b75-w15.chars [...]
```


Remarks: ▶ Recognition task is decoding an arbitrary sequence of characters

Decoding the Model II

What else can mm_align do? Just ask: 'mm_align -h' ...

```
usage: mm_align [<option> ...] <codebooks> <states> <models> [<concepts> ...]
       reads   alignment specifications from standard input
       writes  optimal hypotheses chains found to standard output
    where
       <codebooks>     specifies a file that holds definitions for one
                       or more codebooks (specify '-' if none is used)
       <states>        specifies a file that holds state definitions
       <models>        specifies a file that holds model definitions
       <concepts>      specifies optional files with concept definitions
    valid options are
       [...]
       -b <x>  set beam width to <x> (default: 50)
       -w <x>  set word penalty to <x>
       [...]
       -H <b>  set the hypothesis output format to <b>
               Note: -Hh gives help on hypothesis output format
       -S <b>  set score output format to <b>
               Note: -Sh gives help on score output format
       -L <ngram> use language model given by <ngram> during decoding
       -W <x>  set language model weight to <x> (default: 1.0)
    options for n-best search are
       -R <n>[:<m>] Generate <n> alternative results by computing n-best lists
               via A*-based backward search in Viterbi solutions and report
               <m> final results (default: all results generated)
       [...]
```

Evaluation I

Resource	In theory	In ESMERALDA
Prerequisites:	ground truth annotation of (test) data (i.e. reference word or character sequence)	`iam-db.Xeval-a.chdr`
Method:	computation of Levenshtein distance	`ev_seg`
Parameters:	segmentation of (test) data	`iam-db.Xeval-a-9-`...`.chars`
Result:	error statistics	accuracy = = 100%− error rate; sub./del./ins. breakdown

Command line: Evaluating recognizer output

```
paste   iam-db.Xeval-a.chdr iam-db.Xeval-a-9-b75-w15.chars | \
ev_seg
```

Evaluating recognizer output: Example

```
ev_seg version 1.00 running on polanski under Linux 3.0.1-030001-generic.
        [command line arguments were: 'ev_seg ']
ev_seg: reading segmentation information from stdin.
ev_seg: analyzing at hypotheses level only.
prozent WA/SR/WC/S/D/I   = 60.85+/-0.8  0.00 65.65    29.11  4.61  4.79
```

Overview

Overview

n-Gram Models: Introduction

Goal of statistical language modeling: Define a probability distribution over a set of symbol (= word) sequences

Origin of the name *Language Model*: Methods closely related to

- Statistical modeling of texts
- Imposing restrictions on word hypothesis sequences (especially in automatic speech recognition)

Powerful concept: Use of Markov chain models

Alternative method: Stochastic grammars

- ↯ Rules can not be learned
- ↯ Complicated, costly parameter training

$\Rightarrow$ Not widely used!

n-Gram Models: Example

Examples for statistical models fitting on slides extremely problematic! *Beware!*

Given an empirically defined language fragment:

```
I don't mind if you go
I don't mind if you take it slow
I don't mind if you say yes or no
I don't mind at all
```

[From the lyrics of the *Great Song of Indifference* by Bob Geldof]

Questions:

- How is the phrase ``I don't mind'' most likely continued?
- Which sentence is more plausible, to be expected, or rather "strange"?

  ``I take it if you don't mind'' or

  ``if you take it I don't mind''

n-Gram Models: Definition

Goal: Calculate $P(\mathbf{w})$ for given word sequence $\mathbf{w} = w_1, w_2, \ldots, w_k$

Basis: n-Gram model = Markov chain model of order $n-1$

Method: Factorization of $P(\mathbf{w})$ applying Bayes' rule according to

$$P(\mathbf{w}) = P(w_1)P(w_2|w_1)\ldots P(w_T|w_1,\ldots,w_{T-1}) = \prod_{t=1}^{k} P(w_t|w_1,\ldots,w_{t-1})$$

Problem: Context dependency increases arbitrarily with length of symbol sequence
$\Rightarrow$ Limit length of the "history"

$$P(\mathbf{w}) \approx \prod_{t=1}^{T} P(\underbrace{w_t \mid w_{t-n+1},\ldots,w_{t-1}}_{n \text{ symbols}})$$

Result: Predicted word w_t and *history* form an n-tuple $\Rightarrow$ n-gram ($\hat{=}$ *event*)

$\Rightarrow$ **n-gram models** (typically: $n = 2 \Rightarrow$ bi-gram, $n = 3 \Rightarrow$ tri-gram)

n-Gram Models: Use Cases

Basic assumption similar to HMM case:

1. Reproduce statistical properties of observed data
2. Derive inferences from the model

Problems to be solved:

Evaluation: *How well does the model represent certain data?*

Basis: Probability of a symbol sequence assigned by the model

Model Creation: *How to create a good model?*

- No hidden state variables
  $\Rightarrow$ No iteratively optimizing techniques required
- Parameters can principally be computed directly
  (by simple counting)

More sophisticated methods necessary in practice! [↗ parameter estimation]

n-Gram Models: Notation

Focus on expressions for computing conditional probabilities

Distinction between predicted word and history important

- Arbitrary individual n-gram: $\mathbf{y}z = y_1, y_2, \ldots y_{n-1}z$
  (predicted word: z, history: $\mathbf{y}$)
- General conditional n-gram probability: $P(z|\mathbf{y})$
  ($P(z|y)$ or $P(z|xy)$ for bi- and tri-gram models)
- Absolute frequency of an n-gram: $c(\mathbf{y}z)$
- Some derived properties of n-gram contexts:

  Count of all n-grams with history $\mathbf{y}$: $c(\mathbf{y}\bullet)$

  Number of n-grams occurring k times in context $\mathbf{y}$: $d_k(\mathbf{y}\bullet)$

n-Gram Models: Evaluation

Basic Principle: Determine descriptive power on *unknown* data

Quality Measure: *Perplexity* $\mathcal{P}$

$$\mathcal{P}(\mathbf{w}) = \frac{1}{\sqrt[|\mathbf{w}|]{P(\mathbf{w})}} = \frac{1}{\sqrt[T]{P(w_1, w_2, \ldots, w_T)}} = P(w_1, w_2, \ldots, w_T)^{-\frac{1}{T}}$$

- Reciprocal of geometric mean of symbol probabilities
- Derived from (cross) entropy definition of a (formal) language

$$H(p|q) = -\sum_i \underbrace{p_i}_{\text{data}} \underbrace{\log_2 q_i}_{\text{model}} \longrightarrow -\underbrace{\sum_t \frac{1}{T}}_{\text{empirical data}} \underbrace{\log_2 P(w_t|\ldots)}_{\text{model}} = -\frac{1}{T} \log_2 \prod_t P(w_t|\ldots)$$

$$\mathcal{P}(\mathbf{w}) = 2^{H(\mathbf{w}|P(\cdot|\ldots))} = 2^{-\frac{1}{T}\log_2 \prod_t P(w_t|\ldots)} = P(w_1, w_2, \ldots, w_T)^{-\frac{1}{T}}$$

Question: *How can perplexity be interpreted?*

n-Gram Models: Interpretation of Perplexity

Intuitive interpretation of perplexity:

- Assume: Text $w_1, w_2, ...w_t, ...w_T$ was produced *statistically* by information source from finite vocabulary V
- Problem: How can that generation be "predicted" as exactly as possible?

  Successful: Only very few symbols likely to continue a sequence

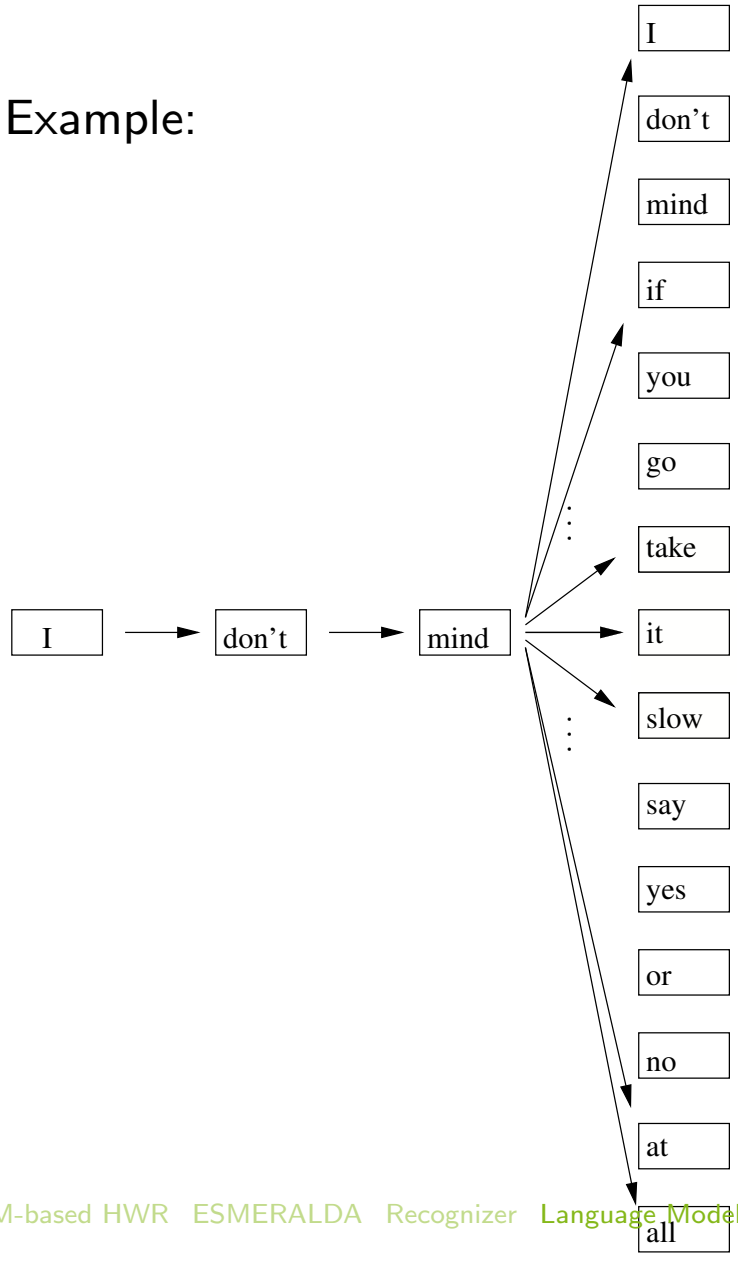
  Unsuccessful: Many symbols have to be taken into account

- Worst case situation: No information

  $\Rightarrow$ No prediction possible

  $\Rightarrow$ All symbols equally likely: $P(w_t|...) = \frac{1}{|V|}$

Example:



n-Gram Models: Interpretation of Perplexity II

- Worst case situation: All symbols equally likely

  $\Rightarrow$ Prediction according to *uniform* distribution $P(w_t|...) = \frac{1}{|V|}$

  - Perplexity of texts generated:

$$\mathcal{P}(\mathbf{w}) = \left\{ \left(\frac{1}{|V|} \right)^T \right\}^{-\frac{1}{T}} = |V|$$

Note: Perplexity equals vocabulary size in absence of restrictions

- In *any* other case: perplexity $\rho < |V|$

  Reason: Entropy (and perplexity) is maximum for uniform distribution!

- Relating this to an "uninformed" source with uniform distribution:
  Prediction is as hard as source with $|V'| = \rho$

  Interpretation: Perplexity gives size of "virtual" lexicon for statistical source!

n-Gram Models: Parameter Estimation

Naive Method:

- Determine number of occurrences
  - $c(w_1, w_2, \ldots w_n)$ for all n-grams and
  - $c(w_1, w_2, \ldots w_{n-1})$ for $n-1$-grams
- Calculate conditional probabilities

$$P(w_n|w_1, w_2, \ldots w_{n-1}) = \frac{c(w_1, w_2, \ldots w_n)}{c(w_1, \ldots w_{n-1})}$$

Example:

$c(\texttt{you say}) = 1$

$c(\texttt{you}) = 3$

$$P(\texttt{say}|\texttt{you}) = \frac{c(\texttt{you say})}{c(\texttt{say})} = \frac{1}{3}$$

Problem: Many n-grams are **not** observed

$\Rightarrow$ *"Unseen events"*

- $c(w_1 \ldots w_n) = 0 \Rightarrow P(w_n|\ldots) = 0$
- $P(\ldots w_1 \cdots w_n \ldots) = 0$!

$c(\texttt{you don't}) = 0$

$P(\texttt{I take it if you don't mind}) = 0$

n-Gram Models: Parameter Estimation II

Parameter estimation in practice

Problem:

- Not *some* but *most* n-gram counts will be **zero**!
- It must be assumed that this is only due to insufficient training data!

$\Rightarrow$ estimate *useful* $P(z|\mathbf{y})$ for $\mathbf{y}z$ with $c(\mathbf{y}z) = 0$

Question: *What estimates are "useful"?*

- small probabilities!, smaller than *seen* events? $\rightarrow$ mostly not guaranteed!
- specific probabilities, not uniform for all unseen events

Solution:

1. Modify n-gram counts and gather "probability mass" for *unseen events*

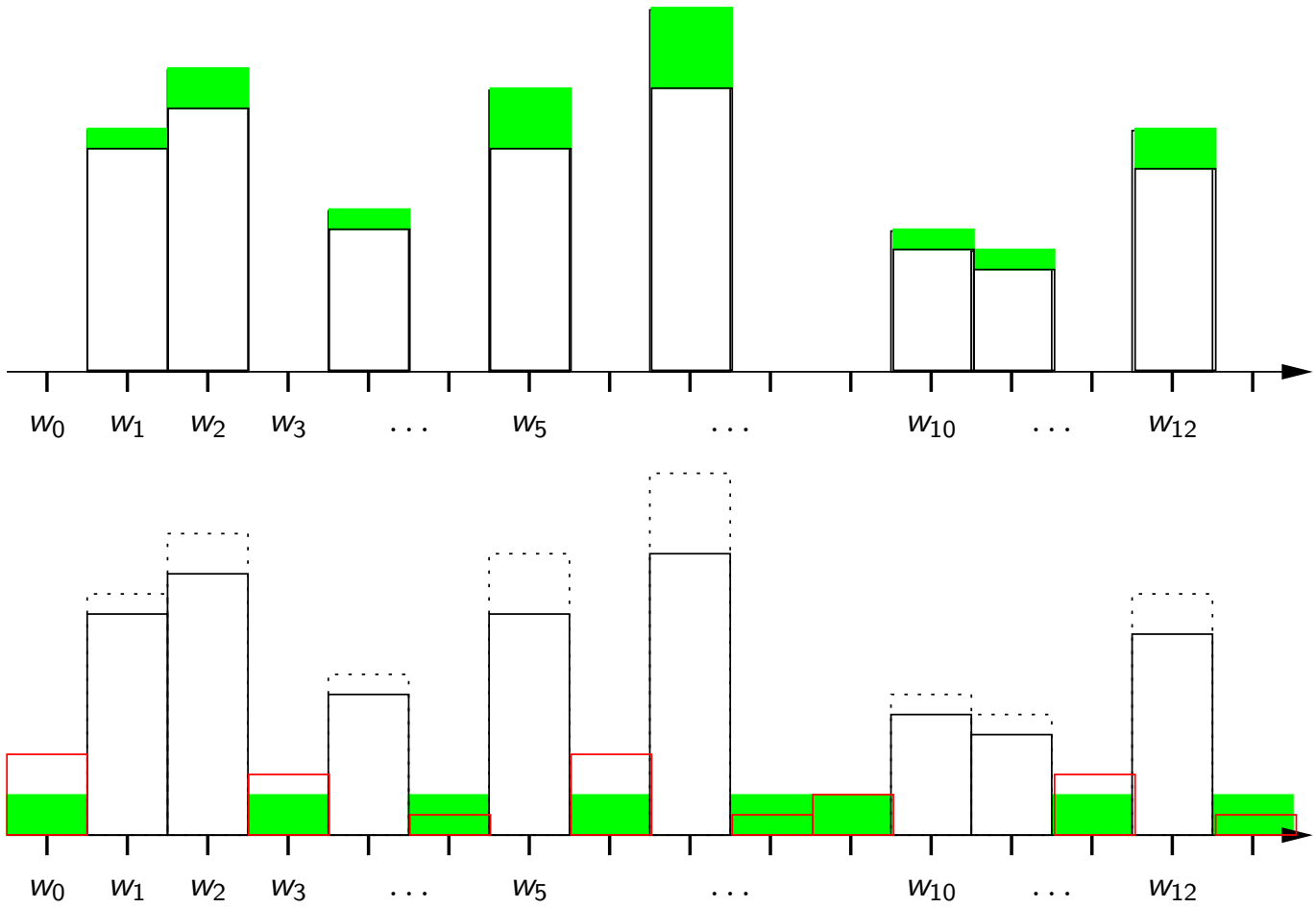
   Note: Keep modification reasonably small for seen events!

2. Redistribute *zero-probability* to *unseen events* according to a more general distribution ($\hat{=}$ *smoothing* of empirical distribution)

   Question: *What distribution is suitable for events we know nothing about?*

n-Gram Models: Parameter Estimation III

Frequency distribution (counts) $\longrightarrow$ Discounting (gathering probability mass)



Zero probability $\longrightarrow$ Incorporate more general distribution

n-Gram Models: Discounting

Gathering of Probability Mass

Calculate modified frequency distribution $f^*(z|\mathbf{y})$ for seen n-grams $\mathbf{y}z$:

$$f^*(z|\mathbf{y}) = \frac{c^*(\mathbf{y}z)}{c(\mathbf{y})} = \frac{c(\mathbf{y}z) - \beta(\mathbf{y}z)}{c(\mathbf{y}\cdot)}$$

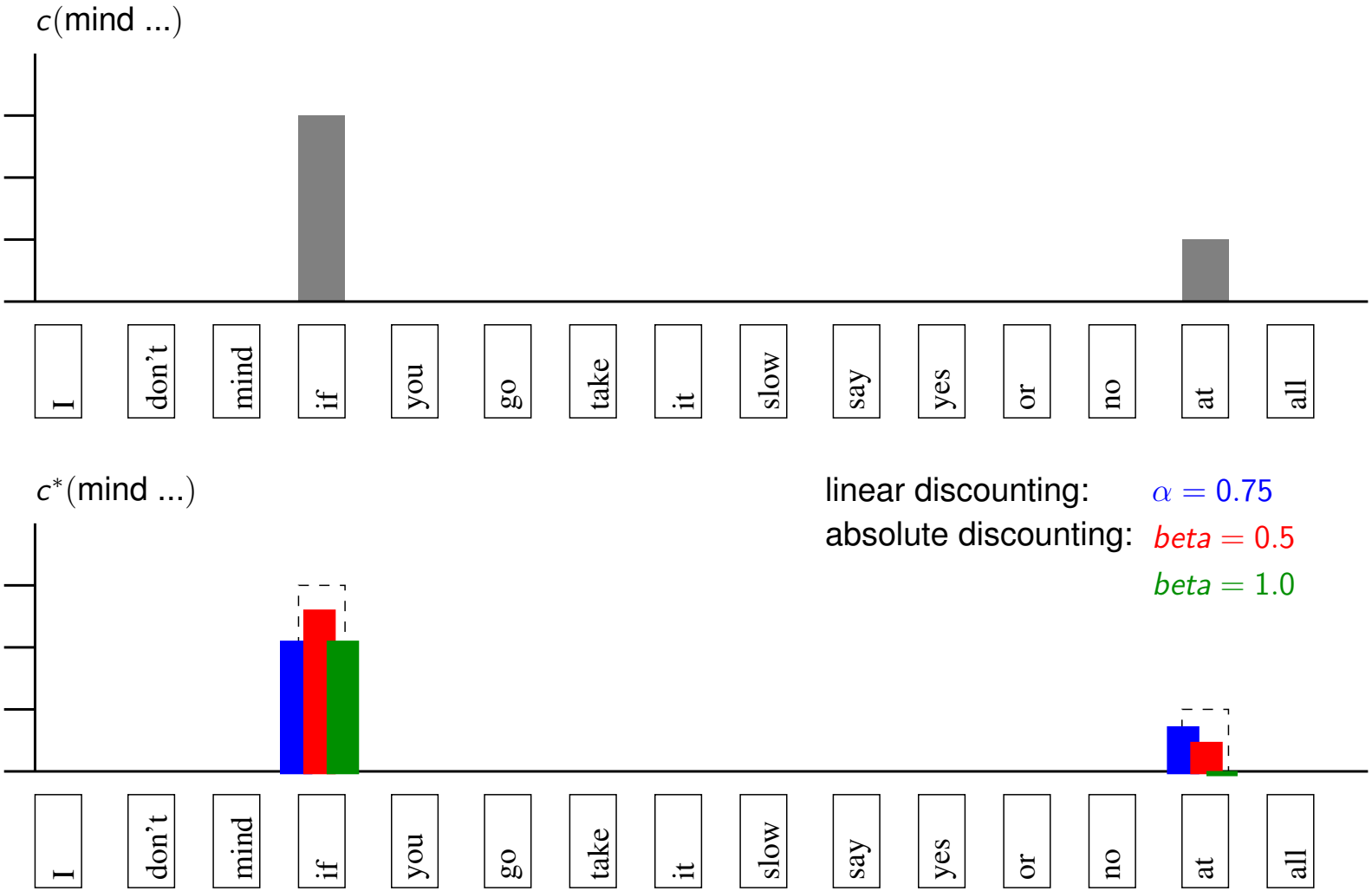
Zero-probability $\lambda(\mathbf{y})$ for history $\mathbf{y}$: Sum of "collected" counts

$$\lambda(\mathbf{y}) = \frac{\sum_{z:c(\mathbf{y}z)>0} \beta(\mathbf{y}z)}{c(\mathbf{y}\cdot)}$$

Choices for discounting factor $\beta()$:

- proportional to n-gram count: $\beta(\mathbf{y}z) = \alpha c(\mathbf{y}z)$ $\Rightarrow$ *linear* discounting
- as some constant $0 < \beta \leq 1$ $\Rightarrow$ *absolute* discounting

n-Gram Models: Discounting - Example



Note: Discounting is performed *individually* for *all* contexts **y**!

n-Gram Models: Smoothing

Redistribution of Probability Mass

Basic methods for incorporating more general distributions:

- Interpolation: Linear combination of (modified) n-gram distribution and (one or more) general distributions
- Backing off: Use more general distribution for unseen events only

Remaining problem: *What is a more general distribution?*

Widely used solution: Corresponding n-1-gram model $P(z|\hat{\mathbf{y}})$ associated with n-gram model $P(z|\mathbf{y})$

- Generalization $\hat{=}$ shortening the context/history

  $\mathbf{y} = y_1, y_2, \ldots y_{n-1} \longrightarrow \hat{\mathbf{y}} = y_2, \ldots y_{n-1}$

- More general distribution obtained:

  $q(z|\mathbf{y}) = q(z|y_1, y_2, \ldots y_{n-1}) \leftarrow P(z|y_2, \ldots y_{n-1}) = P(z|\hat{\mathbf{y}})$

  (i.e. bi-gram for tri-gram model, uni-gram for bi-gram model ...)

n-Gram Language Models: Interpolation

Principle Idea (not considering modified distribution $f^*(\cdot|\cdot)$):

$$P(z|\mathbf{y}) = (1-\alpha)\, f(z|\mathbf{y}) + \alpha\, q(z|\mathbf{y}) \quad 0 \leq \alpha \leq 1$$

Problem: Interpolation weight α needs to be optimized (e.g. on held-out data)

Simplified view with linear discounting: $f^*(z|\mathbf{y}) = (1-\alpha) f(z|\mathbf{y})$

Estimates obtained:

$$P(z|\mathbf{y}) = \begin{cases} f^*(z|\mathbf{y}) + \lambda(\mathbf{y}) q(z|\mathbf{y}) & c^*(\mathbf{y}z) > 0 \\ \lambda(\mathbf{y}) q(z|\mathbf{y}) & c^*(\mathbf{y}z) = 0 \end{cases}$$

Properties:

- Assumes that estimates *always* benefit from smoothing

⇒ All estimates modified

✓ Helpful, if original estimates unreliable

↯ Estimates from large sample counts should be "trusted"

n-Gram Language Models: Backing Off

Basic principle: Back off to general distribution for unseen events

$$P(z|\mathbf{y}) = \begin{cases} f^*(z|\mathbf{y}) & c^*(\mathbf{y}z) > 0 \\ \lambda(\mathbf{y})\, K_{\mathbf{y}}\, q(z|\mathbf{y}) & c^*(\mathbf{y}z) = 0 \end{cases}$$

Normalization factor $K_{\mathbf{y}}$ ensures that: $\sum_z P(z|\mathbf{y}) = 1$

$$K_{\mathbf{y}} = \frac{1}{\sum\limits_{\mathbf{y}z\,:\,c^*(\mathbf{y}z)=0} q(\mathbf{y}z)}$$

Note:

- General distribution used for unseen events only
- Estimates with substantial support unmodified, assumed reliable

n-Gram Language Models: Generalized Smoothing

Observation: With standard solution for $q(z|\mathbf{y})$ more general distribution is again n-gram model $\Rightarrow$ principle can be applied recursively

Example for backing off and tri-gram model:

$$P(z|xy) = \begin{cases} f^*(z|xy) & c^*(xyz) > 0 \\ \lambda(xy)\, K_{xy} \begin{cases} f^*(z|y) & c^*(xyz) = 0 \quad \wedge \quad c^*(yz) > 0 \\ \lambda(y)\, K_y \begin{cases} f^*(z) & c^*(yz) = 0 \quad \wedge \quad c^*(z) > 0 \\ \lambda(\cdot)\, K_\cdot \frac{1}{|V|} & c^*(z) = 0 \end{cases} \end{cases} \end{cases}$$

Note: Combination of absolute discounting and backing off creates powerful n-gram models for a wide range of applications (cf. [4]).

n-Gram Language Models: Representation and Storage

Requirement: *n*-gram models need to define specific probabilities for *all* potential events (i.e. $|V|^n$ scores!)

Observation: Only probabilities of seen events are predefined
(in case of discounting: including context-dependent zero-probability)

$\Rightarrow$ Remaining probabilities can be computed

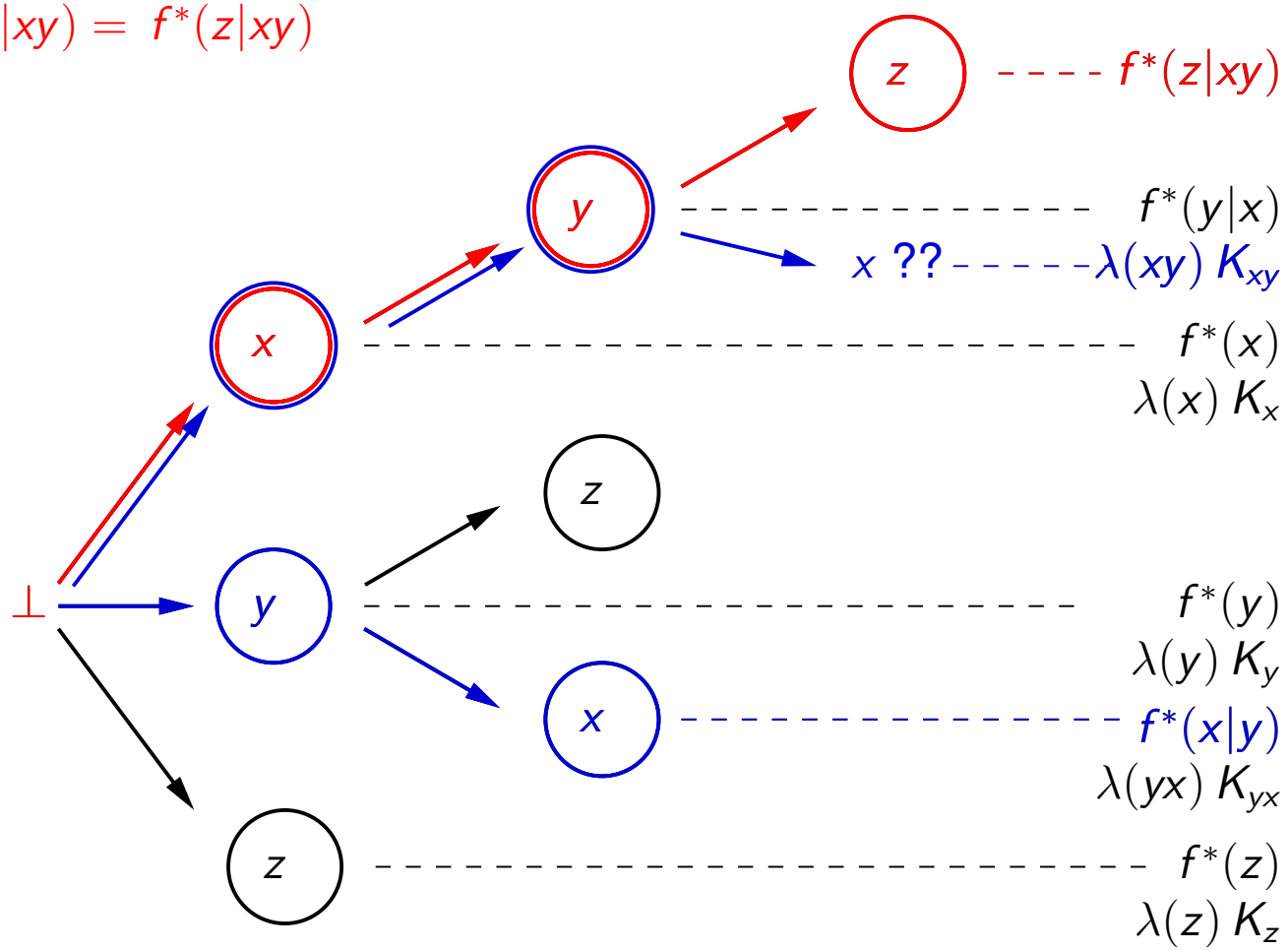
Consequence: Store only probabilities of seen events in memory

$\Rightarrow$ *Huge* savings as *most* events are not observed!

Further Observation: *n*-grams always come in hierarchies
(for representing the respective general distributions)

$\Rightarrow$ Store parameters in prefix-tree for easy access

n-Gram Language Models: Representation and Storage II



$$P(x|xy) = \lambda(xy)\, K_{xy}\, f^*(x|y)$$

n-Gram Language Models: Summary

Pros and Cons:

- ✓ Parameters can be estimated automatically from training texts
- ✓ Models "capture" syntactic, semantic, and pragmatic restrictions of the language fragment considered
- ✓ Can "easily" be combined with statistical recognition systems (e.g. HMMs)
- ↯ Considerable amounts of training data necessary
- ↯ Manageable only for small n (i.e. rather short contexts)

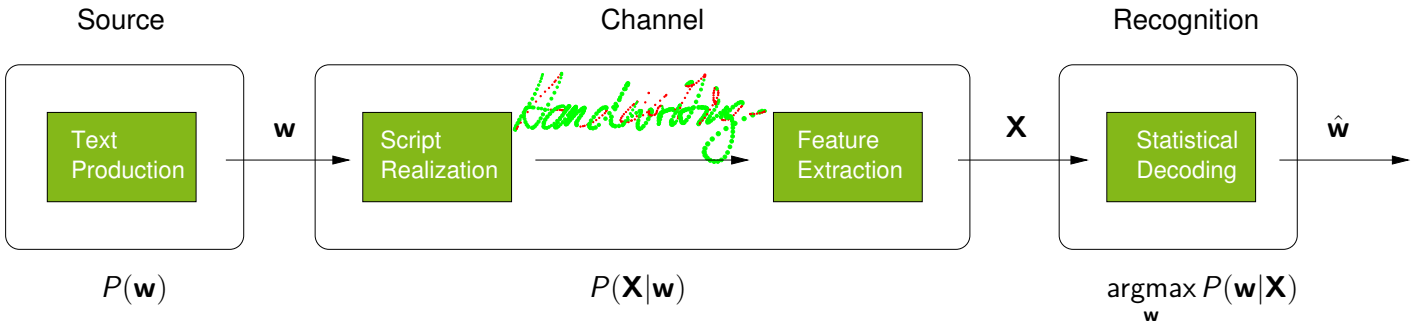
Variants and Extensions of the basic model:

- Category-based language models
  (useful for representing paradigmatic regularities)
- Models for describing long-distance context restrictions
  (useful for languages with discontinuous constitutes, e.g. German)
- Topic-based (i.e. context dependent) models
  (useful, if one global model is too general)

Overview

Integrated Search: Introduction

Remember the channel model:



$\Rightarrow$ HMMs + *n*-gram models *frequently* used in combination!

Problems in practice:

- *How to compute a combined score?* Channel model defines basis only!
- *When to compute the score?* Model valid for *complete* HMM results!
- *How does the language model improve results?*

STOP *Why not use HMMs only to avoid those problems?*

Integrated Search: Basics

Problem 1: Multiplication of $P(\mathbf{X}|O)$ and $P(\mathbf{w})$ does not work in practice!

$\Rightarrow$ Weighted combination using "linguistic matching factor" ρ

$$P(\mathbf{w})^{\rho}\, P(\mathbf{X}|\mathbf{w})$$

Reason: HMM and n-gram scores obtained at largely different time scales and orders of magnitude

- HMM: multi-dimensional density per frame
- n-gram: conditional probability per word

Problem 2: Channel model defines score combination for complete results!

- Can be used in practice only, if ...
  - HMM-based search generates multiple alternative solutions ...
  - n-gram evaluates these *afterward*.
  - $\Rightarrow$ No benefit for HMM search!

$\Rightarrow$ Combination must apply to intermediate results, i.e. path scores $\delta_t(.)$

✓ Achieved by using $P(z|\mathbf{y})$ as "transition probabilities" at word ends.

Integrated Search: Basics II

Question: *How does the language model influence the quality of the results?*

Rule-of-thumb: Error rate decreases proportional to square-root of perplexity

Example for lexicon-free recognition (IAM-DB) with character *n*-grams [15]

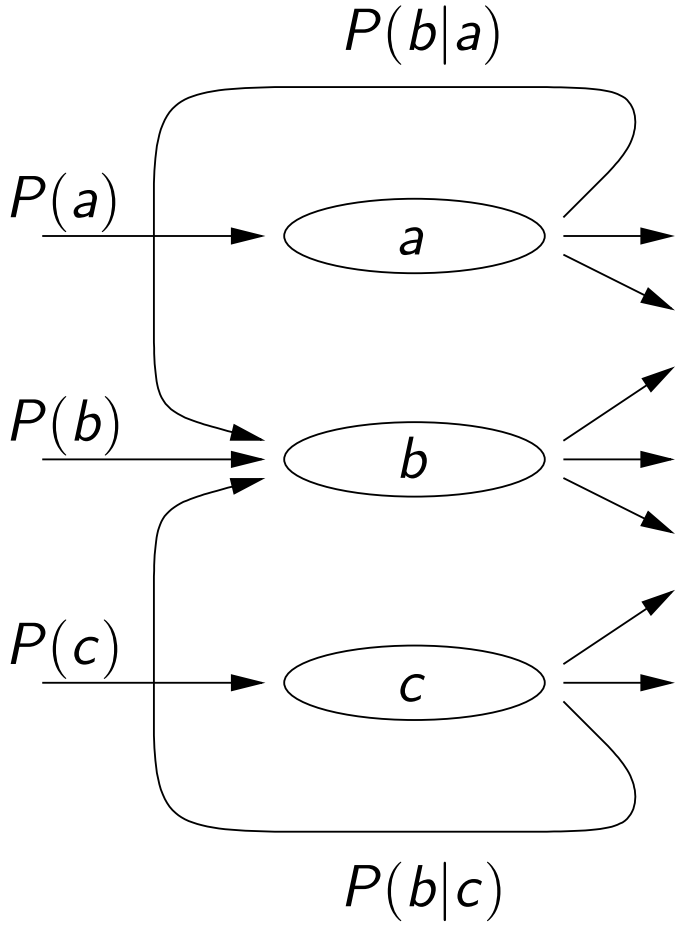
	% CER / perplexity				
	none	2	3	4	5
IAM-DB	29.2 / (75)	22.1 / 12.7	18.3 / 9.3	16.1 7.7	15.6 / 7.3
CER/$\sqrt{\mathcal{P}}$	n.a.	6.2	6.0	6.0	5.8

Note: Important plausibility check: If violated, something *strange* is happening!

Integrated Search: HMM Networks

- Straight-forward extension of HMM-only models
- n-gram scores used as transition probabilities between words
- ↯ HMMs store single-state context only
  $\Rightarrow$ only bi-grams usable!

Question: *How can higher-order models (e.g. tri-grams) be used?*

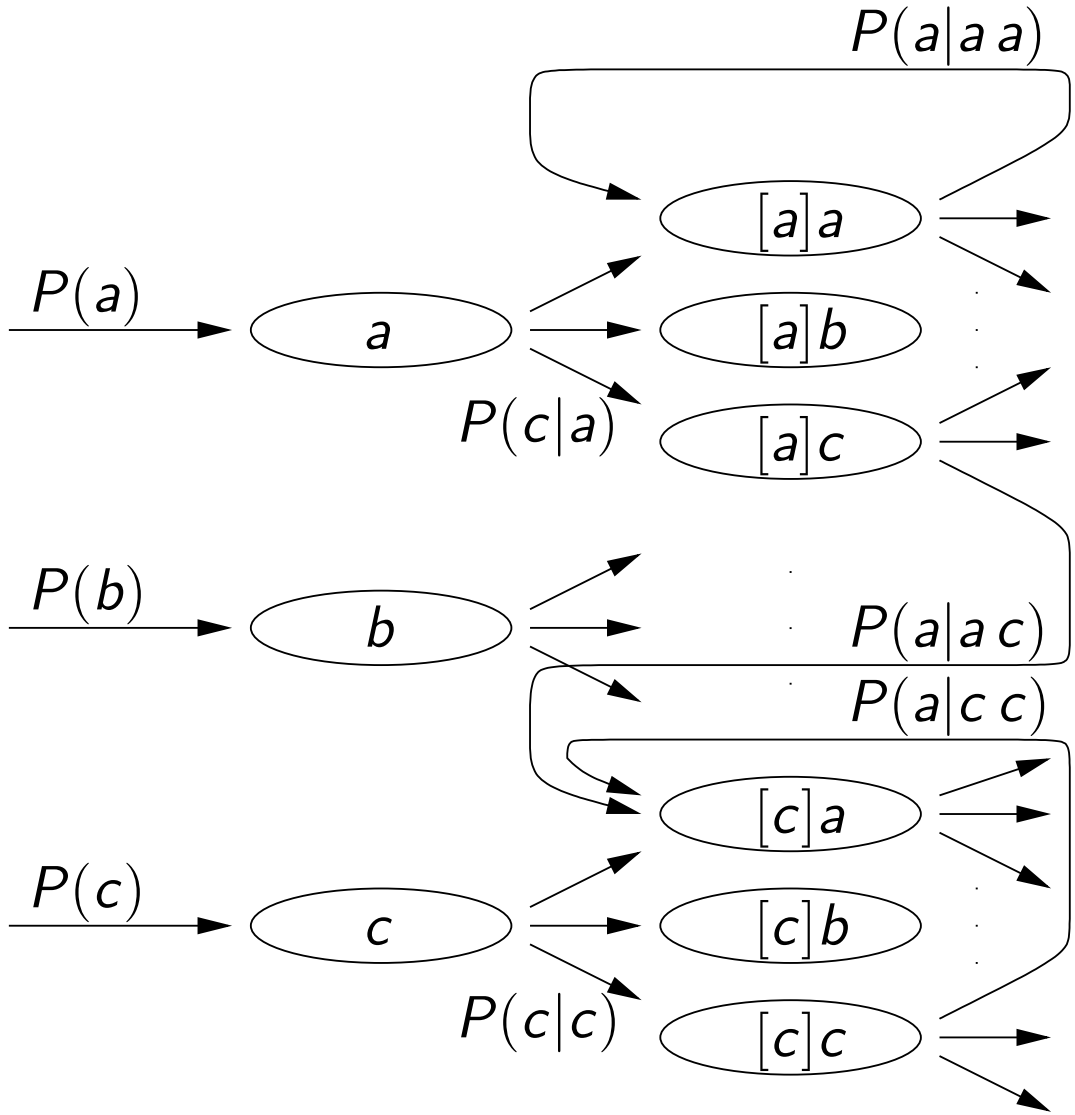


Integrated Search: HMM Networks II

Higher-order *n*-gram models:

⇒ Context dependent copies of word models (i.e. state groups) necessary!

↯ Total model grows exponentially with *n*-gram order!



Overview

ESMERALDA: Creating a Language Model I

... on character level (i.e. with a lexicon of 76 upper and lower-case characters, numerals, and punctuation symbols)

Resource	In theory	In ESMERALDA
Prerequisites:	text corpus	`iam-db.Corpus-a.ctxt`
Tool:	*n*-gram statistics generator	`lm_count`
Parameters:	*n*-gram length	`2`
	(optional) lexicon	`English.clex`
Result:	*n*-gram statistics	`iam-db.count2`

Command line: Creating the *n*-gram statistics

```
cut          -f2 iam-db.Corpus-a.ctxt | sed s/;$/@/ \
| lm_count   -b @ -l English.clex 2 \
             > iam-db.count2 2>iam-db.count2.err
```

ESMERALDA: Creating a Language Model II

Resource	In theory	In ESMERALDA
Prerequisites:	*n*-gram statistics	`iam-db.count2`
Tool:	*n*-gram parameter generator	`lm_param`
Parameters:	discounting method (absolute discounting, $\beta = 1$)	`S1`
Result:	*n*-gram model	`iam-db-S1.clm2`

Command line: Creating the *n*-gram model

```
lm_param   S1 < iam-db.count2 \
           > iam-db-S1.clm2 2> iam-db-S1.clm2.err
```

ESMERALDA: Creating a Language Model III

What else can lm_param do? Just ask: 'lm_param -h'

```
usage: lm_param [<option> ...] <reduce>
        reads   n-gram count data from standard input
        writes  the n-gram parameters to standard output
    where
        <reduce> is the method used for count reduction
                (available methods are: 'S<beta>', 'Sd1', 'Sd3+')
    valid options are
        -f <x>  set smallest probability to <x> (compute logarithm)
        -g [rec|kn] use recursive definitions for general distributions
                ('rec', default) or Kneser-Ney method ('kn')
        -n <n>  use a maximum n-gram length of <n> (default: from count data)
        -m <c>  use only events with a minimum count of <c>
        -s [bo|ip] use backing off ('bo', default) or interpolation ('ip')
                for smoothing probability distributions
        -C <file> dump count data to <file>
    general options are
        -h      display usage information
        -v      be more verbose (can be used multiple times)
        -V      display version information
```

ESMERALDA: Evaluating a Language Model

Resource	In theory	In ESMERALDA
Prerequisites:	*n*-gram model	`iam-db-S1.clm2`
Method:	compute preplexity	`lm_perp`
Input:	(text) text corpus	`iam-db.Xeval-a.chdr`
Result:	empirical perplexity	

Command line: Evaluating the *n*-gram model

```
cut        -f2 iam-db.Xeval-a.chdr \
| lm_perp   -is iam-db-S1.clm2
```

Evaluating the *n*-gram model: Example

```
lm_perp version 2.20 running on murnau under Linux 3.0.1-030001-generic.
lm_perp: 2-gram read.
lm_perp: 15349 words (0 unknown, 0 percent) processed.
lm_perp: 10 cpu ms processing 15349 events (0.000651508 on average).
lm_perp: 15349 score requests with avg. length 2 processed.
lm_perp: score hits up to depth 2 with [ __   0.7 99.3 ] percent.
11.4286
```

ESMERALDA: Using a Language Model in Decoding

Resource	In theory	In ESMERALDA
Prerequisites:	feature representation	`iam-db.Xeval-a.lst`
Method:	Viterbi beam-search	`mm_align`
Parameters:	specification of task model	`<CHARACTERS> +`
	definition of required compound models	`iam-db.swu.def, English.clex.def`
	bi-gram language model	`iam-db-S1.clm2`
	beam witdh	`-b 75`
	language model weight	`-W 7`
Input:	model to be decoded	...
Result:	segmentation of data	

Command line: Decoding the model

```
sed       's/$/ <CHARACTERS> + ;/' ../../iam-db.Xeval-a.lst | \
mm_align  -b 75 -W 7 -L ../../lm/char/iam-db-S1.clm2 \
          iam-db.9.cl iam-db.9.state iam-db.model \
          ../../iam-db.swu.def ../../English.clex.def
```

References I

[1] Issam Bazzi, Richard Schwartz, and John Makhoul.
An omnifont open-vocabulary OCR system for English and Arabic.
IEEE Trans. on Pattern Analysis and Machine Intelligence, 21(6):495–504, 1999.

[2] Radmilo M. Bozinovic and Sargur N. Srihari.
Off-line cursive script word recognition.
IEEE Trans. on Pattern Analysis and Machine Intelligence, 11(1):69–83, 1989.

[3] T. Caesar, J. M. Gloger, and E. Mandler.
Preprocessing and feature extraction for a handwriting recognition system.
In *Proc. Int. Conf. on Document Analysis and Recognition*, pages 408–411, Tsukuba Science City, Japan, 1993.

[4] Stanley F. Chen and Joshua Goodman.
An empirical study of smoothing techniques for language modeling.
Computer Speech & Language, 13:359–394, 1999.

References II

[5] J. G. A. Dolfing and R. Haeb-Umbach.
Signal representations for Hidden Markov Model based on-line handwriting recognition.
In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, volume IV, pages 3385–3388, München, 1997.

[6] Gernot A. Fink.
Developing HMM-based recognizers with ESMERALDA.
In Václav Matoušek, Pavel Mautner, Jana Ocelíková, and Petr Sojka, editors, *Text, Speech and Dialogue*, volume 1692 of *Lecture Notes in Artificial Intelligence*, pages 229–234. Springer, Berlin Heidelberg, 1999.

[7] Gernot A. Fink.
Markov Models for Pattern Recognition.
Springer, Berlin Heidelberg, 2008.

References III

[8] Gernot A. Fink and Thomas Plötz.
ESMERALDA: A development environment for HMM-based pattern recognition systems.
In *7th Open German/Russian Workshop on Pattern Recognition and Image Understanding*, Ettlingen, Germany, 2007.

[9] S. Madhvanath, G. Kim, and V. Govindaraju.
Chaincode contour processing for handwritten word recognition.
IEEE Trans. on Pattern Analysis and Machine Intelligence, 21(9):928–932, 1999.

[10] Thomas Plötz and Gernot A. Fink.
Markov models for offline handwriting recognition: A survey.
Int. Journal on Document Analysis and Recognition, 12(4):269–298, 2009.

[11] Thomas Plötz and Gernot A. Fink.
Markov Models for Handwriting Recognition.
SpringerBriefs in Computer Science. Springer, 2011.

References IV

[12] M. Schenkel, I. Guyon, and D. Henderson.
On-line cursive script recognition using time delay neural networks and hidden Markov models.
In *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, volume 2, pages 637–640, Adelaide, Australia, April 1994.

[13] Richard Schwartz, Christopher LaPre, John Makhoul, Christopher Raphael, and Ying Zhao.
Language-independent OCR using a continuous speech recognition system.
In *Proc. Int. Conf. on Pattern Recognition*, volume 3, pages 99–103, Vienna, Austria, 1996.

[14] M. Wienecke, G. A. Fink, and G. Sagerer.
Experiments in unconstrained offline handwritten text recognition.
In *Proc. 8th Int. Workshop on Frontiers in Handwriting Recognition*, Niagara on the Lake, Canada, August 2002. IEEE.

References V

[15] M. Wienecke, G. A. Fink, and G. Sagerer.
Toward automatic video-based whiteboard reading.
Int. Journal on Document Analysis and Recognition, 7(2–3):188–200, 2005.