Build Your Own Handwriting Recognizer
— ICDAR 2011 Tutorial, Beijing, 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

---

Fink & Vajda
Build Your Own Handwriting Recognizer

[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
  - Example: The effect of bottom congestion due to the story, and the marriage of the central Throughout in terms of the cinema, and again and again
  - Stroke width and quality
  - Size of the writing
  - Variation even for single writer!
- Segmentation of *cursive* script problematic
  ⇒ “Merging” of adjacent characters
General Architecture

Online handwriting recognition

*President Kennedy’s Rejection of it is a painful blow to the West German Government. And, since this is election year in West Germany, Dr. Adenauer is in a tough spot waiting.*

**Captured pen trajectory**

- Imaging device / Digitizer tablet
- Text detection
- Line extraction
- Baseline correction, slant and size normalization
- Serialization and feature extraction
- Model Decoding
  - Writing model (HMM)
  - Language model (n−gram)

**Feature vector sequences**

- Normalized text lines

**Recognition hypotheses**

**Localized text area**

- Text line images

**Offline handwriting recognition**

*President Kennedy’s Rejection of it is a painful blow to the West German Government. And, since this is election year in West Germany, Dr. Adenauer is in a tough spot waiting.*
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.
  ⇒ 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

▶ Introduction

▶ MM-based Handwriting Recognition
  ▶ Motivation
  ▶ Data Preparation
  ▶ Hidden Markov Models
  ▶ Language Models
  ▶ Summary

▶ ESMERALDA

▶ Building the Recognizer

▶ Adding a Language Model

Handwriting Recognition Basics

Fundamentals

Why MM-based HWR?

Preprocessing and Feature Extraction

Definition, Use Cases, Algorithms

Executive Summary

and Further Reading

The Development Environment

References
“Traditional” Recognition Paradigm

Segmentation + Classification:

Potential elementary segments, strokes, ...

✓ 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)

\[
\hat{w} = \arg \max_w P(w\mid X) = \arg \max_w \frac{P(w)P(X\mid w)}{P(X)} = \arg \max_w P(w)P(X\mid w)
\]

Wanted: Sequence of words/characters \(\hat{w}\), which is most probable for given signal/features \(X\)
The Channel Model II

\[
\hat{w} = \arg\max_w P(w|X) = \arg\max_w \frac{P(w)P(X|w)}{P(X)} = \arg\max_w P(w)P(X|w)
\]

Two aspects of modeling:

- **Script (appearance) model:** \( P(X|w) \) \( \Rightarrow \) Representation of words/characters
  
  *Hidden-Markov-Models*

- **Language model:** \( P(w) \) \( \Rightarrow \) 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

▶ Introduction

▶ MM-based Handwriting Recognition
  ▶ Motivation
  ▶ Data Preparation
  ▶ Hidden Markov Models
  ▶ Language Models
  ▶ Summary

▶ ESMERALDA

▶ Building the Recognizer

▶ Adding a Language Model

... Handwriting Recognition Basics

... MM-based HWR Fundamentals

... Why MM-based HWR?

... Preprocessing and Feature Extraction

... Definition, Use Cases, Algorithms

... Executive Summary

... and Further Reading

... The Development Environment

Fink & Vajda
Build Your Own Handwriting Recognizer
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)
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)

for the curtain to rise on the Commonwealth

what, in fact, can the other Commonwealth countries

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

▶ Introduction

▶ MM-based Handwriting Recognition
  ▶ Motivation
  ▶ Data Preparation
  ▶ Hidden Markov Models
  ▶ Language Models
  ▶ Summary

▶ ESMERALDA

▶ Building the Recognizer

▶ Adding a Language Model

... Handwriting Recognition Basics

... MM-based HWR Fundamentals

... Why MM-based HWR?

... Preprocessing and Feature Extraction

... Definition, Use Cases, Algorithms

... Executive Summary

... The Development Environment

... and Further Reading
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 ($\Rightarrow$ first order)

   $$P(s_t|s_{t-1}) = P(s_t|s_{t-1})$$

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

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

Note: Only outputs can be observed $\Rightarrow$ hidden Markov model
Hidden-Markov-Models: Formal Definition

A Hidden-Markov-Model $\lambda$ of first order is defined as:

- a finite set of states:
  \[ \{ s \mid 1 \leq s \leq N \} \]

- a matrix of state transition probabilities:
  \[ A = \{ a_{ij} \mid a_{ij} = P(s_t = j \mid s_{t-1} = i) \} \]

- a vector of start probabilities:
  \[ \pi = \{ \pi_i \mid \pi_i = P(s_1 = i) \} \]

- state specific output probability distributions:
  \[ B = \{ b_{jk} \mid b_{jk} = P(O_t = o_k \mid s_t = j) \} \) (discrete case) \]

  or

  \[ \{ b_j(O_t) \mid b_j(O_t) = p(O_t \mid s_t = j) \} \) (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

\[
p(x) \triangleq \sum_{k=1}^{\infty} c_k \mathcal{N}(x|\mu_k, C_k) \\
\approx \sum_{k=1}^{M} c_k \mathcal{N}(x|\mu_k, C_k)
\]
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?
→ Computation of the production probability $P(O|\lambda)$

Decoding: What is the “internal structure” of the model? (≡ “Recognition”)
→ Computation of the optimal state sequence $s^* = \arg\max_s P(O, s|\lambda)$

Training: How to determine the “optimal” model?
→ Improvement of a given model $\lambda$ with $P(O|\hat{\lambda}) \geq P(O|\lambda)$
The Production Probability

**Wanted:** Assessment of HMMs’ quality for describing statistical properties of data

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

$\Rightarrow P(O|\lambda)$

---

**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. \( P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \)

✓ 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 $P(O|\lambda)$ not sufficient for analysis if individual states are associated to meaningful segments of data

⇒ (Probabilistic) Inference of optimal state sequence $s^*$ necessary

Maximization of posterior probability:

$$s^* = \arg\max_s P(s|O, \lambda)$$

Bayes’ rule:

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

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

$$s^* = \arg\max_s P(s|O, \lambda) = \arg\max_s P(O, s|\lambda)$$

Computation of $s^*$: Viterbi-Algorithm
The Viterbi Algorithm

...inductive procedure for efficient computation of $s^*$ 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. $\delta_1(i) := \pi_i b_i(O_1)$
2. $\delta_{t+1}(j) := \max_i (\delta_t(i) a_{ij}) b_j(O_{t+1})$
3. $P^*(O | \lambda) = P(O, s^* | \lambda) = \max_i \delta_T(i)$
   $s_T^* := \arg\max_j \delta_T(j)$
4. Back-tracking of optimal path:
   $s_t^* = \psi_{t+1}(s_{t+1}^*)$

✓ Implicit segmentation
✓ Linear complexity in time
☒ 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 $\lambda$

$\Rightarrow$ Optimized model $\hat{\lambda}$ better suited for given sample data

General procedure: Parameters of $\lambda$ subject to growth transformation such that

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

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

⇒ *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
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

- Introduction
- MM-based Handwriting Recognition
  - Motivation
  - Data Preparation
  - Hidden Markov Models
  - Language Models
  - Summary
- ESMERALDA
  - The Development Environment
- Building the Recognizer
- Adding a Language Model

Introduction to Handwriting Recognition Basics

Fundamentals

- Why MM-based HWR?
- Preprocessing and Feature Extraction
- Definition, Use Cases, Algorithms
- Executive Summary
- and Further Reading

Overview of the project

- ESMERALDA: Development Environment

References
**n-Gram Models: Definition**

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

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

**Method:** Factorization of $P(w)$ applying Bayes’ rule according to

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

**Problem:** Context dependency increases arbitrarily with length of symbol sequence

$\Rightarrow$ Limit length of the “history”

$$P(w) \approx \prod_{t=1}^{T} P(w_t | w_{t-n+1}, \ldots, w_{t-1})$$

$\underbrace{\quad \quad \quad \quad \quad \quad \quad \quad n \text{ symbols}\quad \quad \quad \quad \quad \quad \quad \quad}$

**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 \( c(yz) \))

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

\[ \Rightarrow \text{estimate useful } P(z|y) \text{ for } yz \text{ with } c(yz) = 0 \]

**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 (\( \hat{=} \text{smoothing of empirical distribution} \))

**Note:** Usually “more general” \( \hat{=} (n-1) \)-gram model
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 $\cong$ HMM) and language model ($\cong$ 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


✓ (Electronic) Inspection copy available!

✓ Conference discount: 20%!


✓ Open access publication!

Overview

- Introduction
- MM-based Handwriting Recognition Basics
- ESMERALDA
  - Software Structure
  - Model Representation
  - The Development Cycle
- The Development Environment
  - Modules etc.
  - HMMs in ESMERALDA
  - From Training to Decoding
- Building the Recognizer
- Adding a Language Model
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

- Introduction
- MM-based Handwriting Recognition
- ESMERALDA
  - Software Structure
  - Model Representation
  - The Development Cycle
- Building the Recognizer
- Adding a Language Model

...Handwriting Recognition Basics

Fundamentals

...The Development Environment

...Modules etc.

...HMMs in ESMERALDA

...From Training to Decoding
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

![Diagram of ESMERALDA System Architecture](image-url)
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 (\( \ast \).cl)

- contains (multiple) codebooks
- Gaussian densities
  - mean vectors \( \vec{\mu}_i \)
  - covariance matrices \( C_i \)
  - 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, \ldots)
  - “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**

\[ P(s_t = S_1 | s_{t-1} = S_1) \]
\[ P(s_t = S_2 | s_{t-1} = S_2) \]
\[ P(s_t = S_3 | s_{t-1} = S_3) \]

\[ P(s_1 = S_1) \]
\[ P(o_t = O_1 | s_t = S_1) \]
\[ P(o_t = O_2 | s_t = S_1) \]
\[ \vdots \]
\[ P(o_t = O_N | s_t = S_1) \]

\[ P(s_2 = S_2 | s_{t-1} = S_2) \]
\[ P(o_t = O_1 | s_t = S_2) \]
\[ P(o_t = O_2 | s_t = S_2) \]
\[ \vdots \]
\[ P(o_t = O_N | s_t = S_2) \]

\[ P(s_3 = S_3 | s_{t-1} = S_3) \]
\[ P(o_t = O_1 | s_t = S_3) \]
\[ P(o_t = O_2 | s_t = S_3) \]
\[ \vdots \]
\[ P(o_t = O_N | s_t = S_3) \]

\[ P(s_3 = S_3 | s_{t-1} = S_2) \]
\[ P(s_2 = S_2 | s_{t-1} = S_1) \]

**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 \[\uparrow \text{md.k.means}\]
- 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 \[\uparrow \text{codebooks}\] and states \[\uparrow \text{mm.param}\])
**Hidden Markov Models – Tools (2)**

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

- existing HMM without code library (states, models, concepts)
- training statistics (*.accu – output from [\(\uparrow\) mm_train])
- output: updated state and model definitions

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

- existing HMM (code library, 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

similar to [\(\uparrow\) mm_align]
**n-gram Models – Overview**

\[
P(z|xy) = f^*(z|xy) \]

**lm:** 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)

_Fink & Vajda_  
Build Your Own Handwriting Recognizer

**References:**
External representations:

- **Statistics (relative frequencies / counts, *.count)**
  - contain counts \( c(\vec{yz}) \) 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 [\( \rightarrow \) lm_count]
- output: language model

**lm_perp**: language model evaluation on sample data – calculation of perplexity
parameters:
- language model (*.lm) [\( \rightarrow \) lm_param]
- sample data (ASCII text)
- output: perplexity
Auxiliary Modules – (Very) Brief Overview

**Required** for practical applications

- *Integrated Development Environment*

  **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

- Introduction
- Handwriting Recognition Basics
- MM-based Handwriting Recognition Fundamentals
- ESMERALDA
  - Software Structure
  - Model Representation
  - The Development Cycle
- The Development Cycle
  - From Training to Decoding
- Building the Recognizer
- Adding a Language Model

References
Model Representation I

(Semi-)Continuous HMMs / 4 Components:

1. Set of densities (component densities for mixtures \( \hat{\text{codebook}} \))
   - \( id, type, \text{mean vector}, \text{covariance matrix} \)
   - \( \Rightarrow \).cl, e.g.: icdar2011.0.cl

2. Set of states
   - \( id, \text{transition probabilities}, \text{mixture weights} \)
   - \( \Rightarrow \).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 [...]
```

- 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 $\sim$ # 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

<table>
<thead>
<tr>
<th>Model</th>
<th>Linear 4:</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>[0 1 2 3 ]</td>
<td>[-] Linear 4: [0 1 2 3 ]</td>
</tr>
<tr>
<td>/0/</td>
<td>[4 5 6 7 ]</td>
<td>/0/ Linear 4: [4 5 6 7 ]</td>
</tr>
<tr>
<td>/1/</td>
<td>[8 9 10 11 ]</td>
<td>/1/ Linear 4: [8 9 10 11 ]</td>
</tr>
<tr>
<td>/2/</td>
<td>[12 13 14 15 ]</td>
<td>/2/ Linear 4: [12 13 14 15 ]</td>
</tr>
<tr>
<td>[...]</td>
<td>[...]</td>
<td>[...]</td>
</tr>
<tr>
<td>/a/</td>
<td>[44 45 46 47 ]</td>
<td>/a/ Linear 4: [44 45 46 47 ]</td>
</tr>
<tr>
<td>/A/</td>
<td>[48 49 50 51 ]</td>
<td>/A/ Linear 4: [48 49 50 51 ]</td>
</tr>
<tr>
<td>[...]</td>
<td>[...]</td>
<td>[...]</td>
</tr>
<tr>
<td>/Z/</td>
<td>[316 317 318 319 ]</td>
<td>/Z/ Linear 4: [316 317 318 319 ]</td>
</tr>
</tbody>
</table>

- 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
Example: Word and task models

```c
/* 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:** `<part₁>` `<part₂>` `[<part₃> ...]`
- “series connection” of at least two models (or concepts)

**alternative:** `<part₁> | <part₂> [| <part₃> ...]`
- “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

▶ Introduction

▶ MM-based Handwriting Recognition

▶ ESMERALDA
  ▶ Software Structure
  ▶ Model Representation
  ▶ The Development Cycle

▶ Building the Recognizer

▶ Adding a Language Model

... Handwriting Recognition Basics

... MM-based HWR Fundamentals

... The Development Environment

... Modules etc.

... HMMs in ESMERALDA

... From Training to Decoding
Development Cycle I

0. Data & Task Definition:
   - Preparation of the data
     - Text extraction, line separation, preprocessing & normalization, serialization, feature extraction
     ⇒ 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

- **desired number of densities**
  - md_k_means
  - md_param
  - mm_init

- **number of training iterations**
  - mm_train
  - mm_param
  - update
  - accu
  - codelibrary
  - model
  - state

- **initial segmentation**
  - initial segmentation
  - labelled training data
  - configuration of basic models
  - definition of compound models

- **evaluation/application**
  - recognition task
  - recognition hypotheses
  - test samples
  - training data
  - external data
  - configuration of basic models
  - definition of compound models
  - pattern recognition hypotheses
  - test samples

- **parameters**
  - parameters
  - basic models
  - internal data
  - external data

- **iterative training**
  - iterative training
  - training
  - validation
  - model
  - state
  - codelibrary
  - model
  - state

- **initialization**
  - initialization
  - initialization
  - initialization
Overview

Introduction

MM-based Handwriting Recognition

ESMERALDA

Building the Recognizer

- Initializing the Codebook
- Initializing the HMM
- Re-Estimating HMM Parameters
- Decoding the Model
- Evaluation

Adding a Language Model
## Initializing the Codebook I

<table>
<thead>
<tr>
<th>Resource</th>
<th>In theory</th>
<th>In ESMERALDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prerequisites:</strong></td>
<td>feature representation</td>
<td>iam-db.Corpus-a.lst</td>
</tr>
<tr>
<td><strong>Method:</strong></td>
<td>$k$-means</td>
<td>md_k_means</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
<td>dimension of feature vectors</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>desired number of densities $N$</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>type of densities: Gaussians w. diagonal covariances</td>
<td>-g</td>
</tr>
<tr>
<td><strong>Result:</strong></td>
<td>set of densities ($N$ or a few less)</td>
<td>iam-db.0.cl</td>
</tr>
</tbody>
</table>

### 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** ⇒ Sufficient for our purposes!
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

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!

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

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 [options] ... 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 ... [options] ... [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>` occurrences of `<t>` samples each were available (for semi-continuous models only)
## Re-Estimating HMM Parameters I

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

### Command line: Re-estimating model parameters

```bash
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.
- Actual parameters still need to be created!
- Performs one re-estimation step only: Needs to be applied multiple times.
- By default uses Viterbi pre-alignment of training data not suitable with a "flat start": Switch to beam-pruned Baum-Welch training (`-a none`).
- Reports "model quality" on stderr: `mm_train: average P(O|lambda) is 11.6556.`
### Re-Estimating HMM Parameters II

<table>
<thead>
<tr>
<th>Resource</th>
<th>In theory</th>
<th>In ESMERALDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>parameter estimation statistics</td>
<td>iam-db.1.accu, iam-db.1.update</td>
</tr>
<tr>
<td><strong>Tools:</strong></td>
<td></td>
<td>mm_param, md_param</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
<td>previous parameter set</td>
<td>iam-db.0.cl, iam-db.0.state, iam-db.0.model</td>
</tr>
<tr>
<td><strong>Result:</strong></td>
<td>re-estimated parameter sets</td>
<td>iam-db.1.state, iam-db.1.cl</td>
</tr>
</tbody>
</table>

**Command lines: Creating parameter estimates from statistics**

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

md_param iam-db.0.cl \
<i>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.
- `md_param` creates code library parameters from statistics preserving original structure.
Decoding the Model I

<table>
<thead>
<tr>
<th>Resource</th>
<th>In theory</th>
<th>In ESMERALDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prerequisites:</strong></td>
<td>feature representation</td>
<td>iam-db.Xeval-a.lst</td>
</tr>
<tr>
<td><strong>Method:</strong></td>
<td>Viterbi beam-search</td>
<td>mm_align</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
<td>specification of task model</td>
<td>&lt;CHARACTERS&gt; +</td>
</tr>
<tr>
<td></td>
<td>definition of required compound models</td>
<td>iam-db.swu.def, English.clex.def</td>
</tr>
<tr>
<td></td>
<td>beam width</td>
<td>-b 75</td>
</tr>
<tr>
<td></td>
<td>word penalty</td>
<td>-w 15</td>
</tr>
<tr>
<td><strong>Input:</strong></td>
<td>model to be decoded</td>
<td>iam-db.0.cl, iam-db.0.state,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>iam-db.0.model</td>
</tr>
<tr>
<td><strong>Result:</strong></td>
<td>segmentation of data</td>
<td>iam-db.Xeval-a-9-...chars</td>
</tr>
</tbody>
</table>

Command line: Decoding the model

```bash
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.
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

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

### 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
76 ms (40 cpu ms) processing time -- 331 lines processed.
```
Overview

▶ Introduction

▶ MM-based Handwriting Recognition Fundamentals

▶ ESMERALDA The Development Environment

▶ Building the Recognizer

▶ Adding a Language Model
  ▶ Language Model: Foundations
  ▶ Integrated Search
  ▶ Building and Using an LM in ESMERALDA

Handwriting Recognition Basics

The Development Environment

Language Model
Overview

- Introduction
  … Handwriting Recognition Basics

- MM-based Handwriting Recognition Fundamentals

- ESMERALDA … The Development Environment

- Building the Recognizer

- Adding a Language Model
  - Language Model: Foundations
  - Integrated Search
  - Building and Using an LM in ESMERALDA
**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
  
  ⇒ Not widely used!
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 $P(w)$ for given word sequence $w = w_1, w_2, \ldots, w_k$

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

**Method:** Factorization of $P(w)$ applying Bayes’ rule according to

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

**Problem:** Context dependency increases arbitrarily with length of symbol sequence

$\Rightarrow$ Limit length of the “history”

$$P(w) \approx \prod_{t=1}^{T} P(w_t | w_{t-n+1}, \ldots, w_{t-1})$$

$n$ symbols

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

$\Rightarrow$ $n$-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
  - 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: \( yz = y_1, y_2, \ldots, y_{n-1}z \)
  (predicted word: \( z \), history: \( y \))

- General conditional \( n \)-gram probability:
  \( P(z|y) \) or \( P(z|xy) \) for bi- and tri-gram models

- Absolute frequency of an \( n \)-gram:
  \( c(yz) \)

- Some derived properties of \( n \)-gram contexts:
  - Count of all \( n \)-grams with history \( y \):
    \( c(y\cdot) \)
  - Number of \( n \)-grams occurring \( k \) times in context \( y \):
    \( d_k(y\cdot) \)
\[ P(w) = \frac{1}{|w|} \sqrt{P(w)} = \frac{1}{\sqrt[\tau]{P(w_1, w_2, \ldots, w_T)}} = P(w_1, w_2, \ldots, w_T)^{-\frac{1}{\tau}} \]

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

\[ H(p|q) = -\sum_i p_i \log_2 q_i \rightarrow -\sum_t \frac{1}{T} \log_2 P(w_t|\ldots) = -\frac{1}{T} \log_2 \prod_t P(w_t|\ldots) \]

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

**Question:** How can perplexity be interpreted?
Intuitive interpretation of perplexity:

- Assume: Text $w_1, w_2, \ldots w_t, \ldots w_T$ 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
  - No prediction possible
  - All symbols equally likely: $P(w_t|\ldots) = \frac{1}{|V|}$
Worst case situation: All symbols equally likely

⇒ Prediction according to uniform distribution

\[ P(w_t|...) = \frac{1}{|V|} \]

Perplexity of texts generated:

\[ P(w) = \left\{ \left( \frac{1}{|V|} \right)^T \right\}^{-1/|V|} = |V| \]

Note: Perplexity equals vocabulary size in absence of restrictions

\[ \text{In any other case: perplexity } \rho < |V| \]

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
  - \( c(w_1, w_2, ... w_n) \) for all \( n \)-grams and
  - \( c(w_1, w_2, ... w_{n-1}) \) for \( n - 1 \)-grams
- Calculate conditional probabilities

\[
P(w_n|w_1, w_2, ... w_{n-1}) = \frac{c(w_1, w_2, ... w_n)}{c(w_1, ... w_{n-1})}
\]

**Problem:** Many \( n \)-grams are not observed
⇒ "Unseen events"

- \( c(w_1 ... w_n) = 0 \) ⇒ \( P(w_n|...) = 0 \)
- \( c(you \ don’t) = 0 \)
- \( P(I \ take \ it \ if \ you \ don’t \ mind) = 0 \)

**Example:**
- \( c(you \ say) = 1 \)
- \( c(you) = 3 \)
- \( P(say|you) = \frac{c(you \ say)}{c(say)} = \frac{1}{3} \)
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!

⇒ estimate useful $P(z|y)$ for $yz$ with $c(yz) = 0$

Question: What estimates are “useful”? (Note: small probabilities!, smaller than *seen* events? → 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) $\rightarrow$ Discounting (gathering probability mass)

Zero probability $\rightarrow$ Incorporate more general distribution
\(n\)-Gram Models: Discounting

Gathering of Probability Mass

Calculate modified frequency distribution \(f^*(z|y)\) for seen \(n\)-grams \(yz\):

\[
f^*(z|y) = \frac{c^*(yz)}{c(y)} = \frac{c(yz) - \beta(yz)}{c(y\cdot)}
\]

Zero-probability \(\lambda(y)\) for history \(y\): Sum of “collected” counts

\[
\lambda(y) = \sum_{z: c(yz) > 0} \beta(yz) \frac{\beta(yz)}{c(y\cdot)}
\]

Choices for discounting factor \(\beta()\):

- proportional to \(n\)-gram count: \(\beta(yz) = \alpha c(yz)\) \(\Rightarrow\) linear discounting
- as some constant \(0 < \beta \leq 1\) \(\Rightarrow\) absolute discounting
$n$-Gram Models: Discounting - Example

Note: Discounting is performed \emph{individually} for all contexts $y$!
\textit{n-Gram Models: Smoothing}

\textbf{Redistribution of Probability Mass}

\textbf{Basic methods} for incorporating more general distributions:

\textit{Interpolation:} Linear combination of (modified) \textit{n}-gram distribution and (one or more) general distributions

\textit{Backing off:} Use more general distribution for unseen events only

\textbf{Remaining problem:} What is a more general distribution?

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

\begin{itemize}
  \item Generalization \( \hat{\cdots} \) shortening the context/history
    \[ y = y_1, y_2, \ldots y_{n-1} \longrightarrow \hat{y} = y_2, \ldots y_{n-1} \]
  \item More general distribution obtained:
    \[ q(z|y) = q(z|y_1, y_2, \ldots y_{n-1}) \leftarrow P(z|y_2, \ldots y_{n-1}) = P(z|\hat{y}) \] (i.e. bi-gram for tri-gram model, uni-gram for bi-gram model ...)\end{itemize}
n-Gram Language Models: Interpolation

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

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

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

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

Estimates obtained:

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

Properties:
- Assumes that estimates always benefit from smoothing
- All estimates modified
- Helpful, if original estimates unreliable
- Estimates from large sample counts should be “trusted”
**Basic principle:** Back off to general distribution for unseen events

\[
P(z|y) = \begin{cases} 
  f^*(z|y) & c^*(yz) > 0 \\
  \lambda(y) K_y q(z|y) & c^*(yz) = 0
\end{cases}
\]

Normalization factor \( K_y \) ensures that: \( \sum_z P(z|y) = 1 \)

\[
K_y = \frac{1}{\sum_{yz : c^*(yz)=0} q(yz)}
\]

**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 $q(z|y)$ more general distribution is again $n$-gram model $\Rightarrow$ principle can be applied recursively

**Example** for backing off and tri-gram model:

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

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

⇒ Remaining probabilities can be computed

**Consequence:** Store only probabilities of seen events in memory

⇒ *Huge* savings as *most* events are not observed!

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

⇒ Store parameters in prefix-tree for easy access
\[ P(z|xy) = f^*(z|xy) \]

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

- Introduction
- Handwriting Recognition Basics
- MM-based Handwriting Recognition Fundamentals
- ESMERALDA The Development Environment
- Building the Recognizer
- Adding a Language Model
  - Language Model: Foundations
  - Integrated Search
  - Building and Using an LM in ESMERALDA
Integrated Search: Introduction

Remember the channel model:

\[
\begin{align*}
P(w) & \quad \text{Source} & \quad \text{Text Production} \\
& \quad \text{Channel} & \quad \text{Script Realization} \quad \text{Feature Extraction} \quad \text{Statistical Decoding} \\
& \quad \text{Recognition} & \quad \text{X} \quad \arg\max_w P(w|X) \\
\end{align*}
\]

⇒ 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?**

Why not use HMMs only to avoid those problems?
**Integrated Search: Basics**

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

⇒ Weighted combination using “linguistic matching factor” $\rho$

\[
P(w)^{\rho} P(X|w)
\]

**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*.
⇒ No benefit for HMM search!
⇒ Combination must apply to intermediate results, i.e. path scores $\delta_t(.)$

✓ Achieved by using $P(z|y)$ 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]

<table>
<thead>
<tr>
<th></th>
<th>none</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAM-DB</td>
<td>29.2</td>
<td>22.1</td>
<td>18.3</td>
<td>16.1</td>
<td>15.6</td>
</tr>
<tr>
<td>CER/\sqrt{P}</td>
<td>n.a.</td>
<td>6.2</td>
<td>6.0</td>
<td>6.0</td>
<td>5.8</td>
</tr>
</tbody>
</table>

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

\[ P(b|a) \]
\[ P(a) \]
\[ P(b) \]
\[ P(c) \]
\[ P(b|c) \]

\( \Rightarrow \) HMMs store single-state context only

Question: How can higher-order models (e.g. tri-grams) be used?
Higher-order $n$-gram models:

$\Rightarrow$ Context dependent copies of word models (i.e. state groups) necessary!

$\Leftrightarrow$ Total model grows exponentially with $n$-gram order!
Overview

▶ Introduction

▶ Handwriting Recognition Basics

▶ MM-based Handwriting Recognition Fundamentals

▶ ESMERALDA The Development Environment

▶ Building the Recognizer

▶ Adding a Language Model
  ▶ Language Model: Foundations
  ▶ Integrated Search
  ▶ Building and Using an LM in ESMERALDA
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)

<table>
<thead>
<tr>
<th>Resource</th>
<th>In theory</th>
<th>In ESMERALDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prerequisites:</strong></td>
<td>text corpus</td>
<td>iam-db.Corpus-a.ctxt</td>
</tr>
<tr>
<td><strong>Tool:</strong></td>
<td>n-gram statistics generator</td>
<td>lm_count</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
<td>n-gram length</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>(optional) lexicon</td>
<td>English.clex</td>
</tr>
<tr>
<td><strong>Result:</strong></td>
<td>n-gram statistics</td>
<td>iam-db.count2</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Resource</th>
<th>In theory</th>
<th>In ESMERALDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prerequisites</td>
<td>n-gram statistics</td>
<td>iam-db.count2</td>
</tr>
<tr>
<td>Tool:</td>
<td>n-gram parameter generator</td>
<td>lm_param</td>
</tr>
<tr>
<td>Parameters:</td>
<td>discounting method</td>
<td>S1</td>
</tr>
<tr>
<td></td>
<td>(absolute discounting, $\beta = 1$)</td>
<td></td>
</tr>
<tr>
<td>Result:</td>
<td>n-gram model</td>
<td>iam-db-S1.clm2</td>
</tr>
</tbody>
</table>

Command line: Creating the n-gram model

```
lm_param S1 < iam-db.count2 \
> iam-db-S1.clm2 2> iam-db-S1.clm2.err
```
What else can \texttt{lm\_param} do? Just ask: \texttt{\textquotesingle lm\_param \ -h\textquotesingle}

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

<table>
<thead>
<tr>
<th>Resource</th>
<th>In theory</th>
<th>In ESMERALDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Prerequisites:</em></td>
<td><em>n</em>-gram model</td>
<td>iam-db-S1.clm2</td>
</tr>
<tr>
<td><em>Method:</em></td>
<td>compute perplexity</td>
<td>lm_perp</td>
</tr>
<tr>
<td><em>Input:</em></td>
<td>(text) text corpus</td>
<td>iam-db.Xeval-a.chdr</td>
</tr>
<tr>
<td><em>Result:</em></td>
<td>empirical perplexity</td>
<td></td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Resource</th>
<th>In theory</th>
<th>In ESMERALDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prerequisites:</strong></td>
<td>feature representation</td>
<td>iam-db.Xeval-a.lst</td>
</tr>
<tr>
<td><strong>Method:</strong></td>
<td>Viterbi beam-search</td>
<td>mm_align</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
<td>specification of task model</td>
<td>&lt;CHARACTER&gt; +</td>
</tr>
<tr>
<td></td>
<td>definition of required compound models</td>
<td>iam-db.swu.def, English.clex.def</td>
</tr>
<tr>
<td></td>
<td>bi-gram language model</td>
<td>iam-db-S1.clm2</td>
</tr>
<tr>
<td></td>
<td>beam width</td>
<td>-b 75</td>
</tr>
<tr>
<td></td>
<td>language model weight</td>
<td>-W 7</td>
</tr>
<tr>
<td><strong>Input:</strong></td>
<td>model to be decoded</td>
<td>...</td>
</tr>
<tr>
<td><strong>Result:</strong></td>
<td>segmentation of data</td>
<td></td>
</tr>
</tbody>
</table>

**Command line: Decoding the model**

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


Developing HMM-based recognizers with ESMERALDA. 

*Markov Models for Pattern Recognition*. 
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.

Chaincode contour processing for handwritten word recognition.

Markov models for offline handwriting recognition: A survey.

*Markov Models for Handwriting Recognition*.
On-line cursive script recognition using time delay neural networks and hidden Markov models.

Language-independent OCR using a continuous speech recognition system.

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.