Offline Handritting Word Recognition

Thijs Kooi, Davide Modolo

January 28, 2011

Table of contents

Overview of the Project

Off-line handwriting recognition

- It involves the automatic conversion of text in an image into letter codes which are usable within computer and text-processing applications
- Off-line handwriting recognition is comparatively difficult, as different people have different handwriting styles

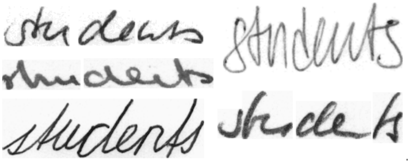


Figure: 'Students' written by different authors

Our AI Project

A lot of research has been done over the past years.

We explored the topic and implemented a full pipeline for the task.
The research touched different fields:

- Data Collection
- Image Processing
- Features extraction
- Machine Learning

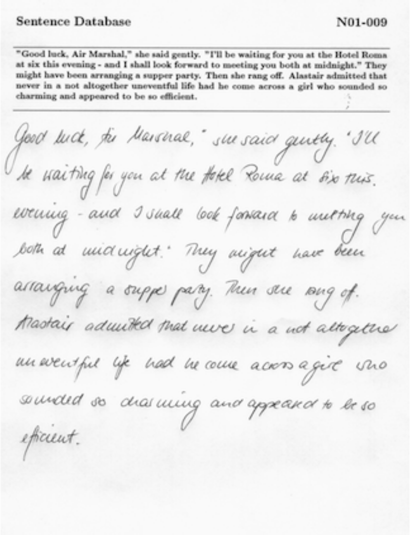
Dataset

The IAM Handwriting Database 3.0[1]

- Unconstrained handwritten text (scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels)
- 1'539 pages of scanned text of 657 writers
- 13'353 isolated and labeled text lines
- 115'320 isolated and labeled words

[1] http://www.iam.unibe.ch/fki/databases/iam-handwriting-database

Example of a page of scanned text



Sentence Database N01-009

"Good luck, Air Marshal," she said gently. "I'll be waiting for you at the Hotel Roma at six this evening - and I shall look forward to meeting you both at midnight." They might have been arranging a supper party. Then she rang off. Alastair admitted that never in a not altogether uneventful life had he come across a girl who sounded so charming and appeared to be so efficient.

"Good luck, Air Marshal," she said gently. "I'll
be waiting for you at the Hotel Roma at six this.
evening - and I shall look forward to meeting you
both at midnight." They might have been
arranging a supper party. Then she rang off.
Alastair admitted that never in a not altogether
uneventful life had he come across a girl who
sounded so charming and appeared to be so
efficient.

Implementation Details

Pipeline

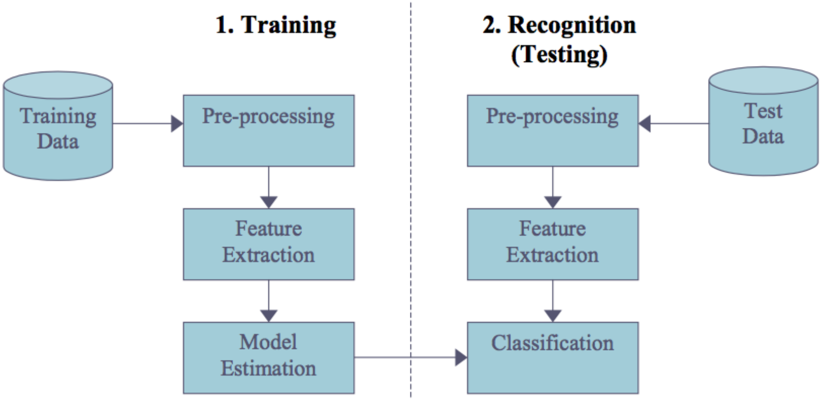


Figure: Pipeline of a word recognition system

Implementation Details

Pre-Processing

Pre-processing

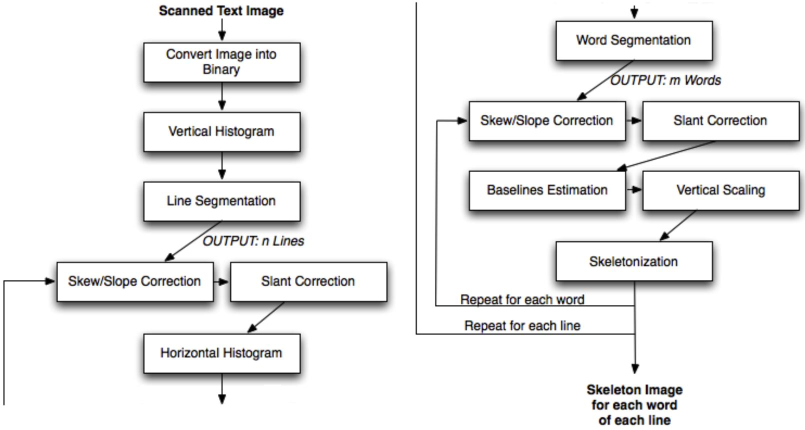
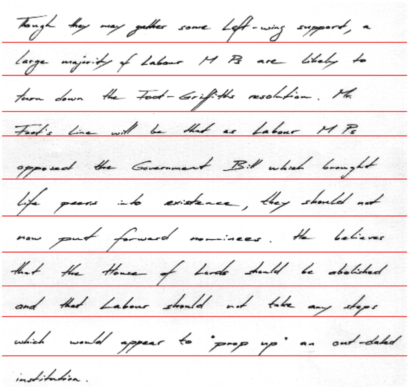
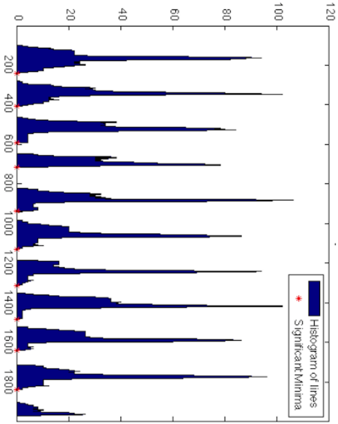


Figure: Pipeline for the pre-processing/normalization step

Line Segmentation



Original image segmented in lines



Vertical histograms and significant minima

Figure: Example of line segmentation

Skew and Slope Correction

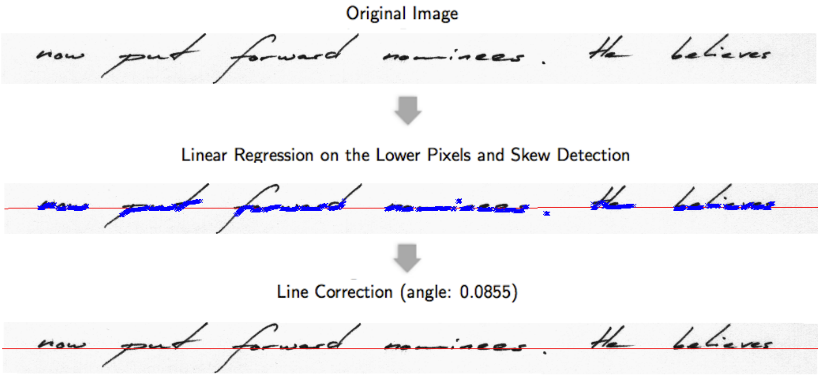


Figure: Skew detection and correction pipeline

Slant Correction

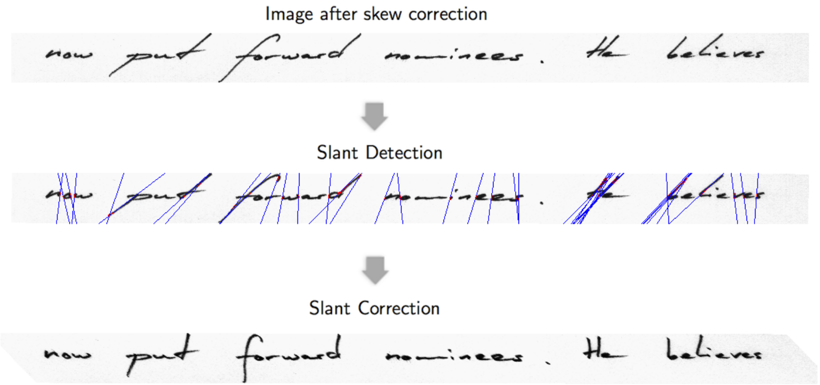
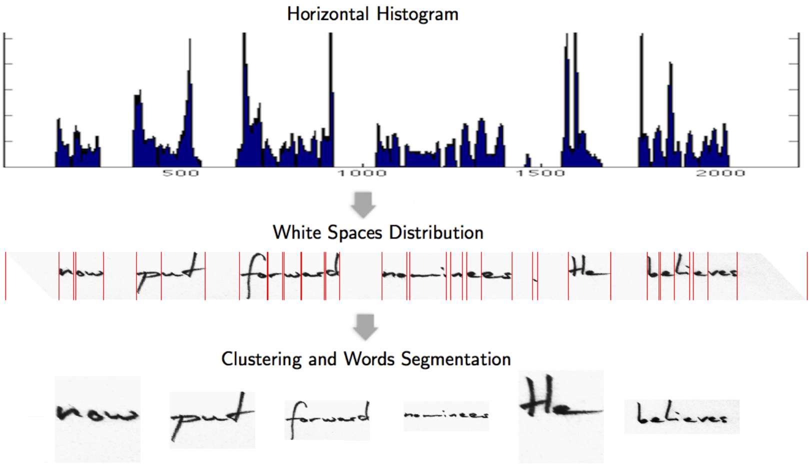
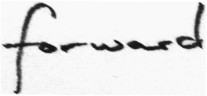


Figure: Slant detection and correction pipeline

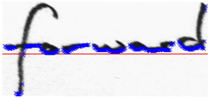
Word segmentation



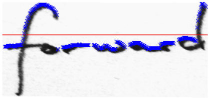
Baseline Estimation



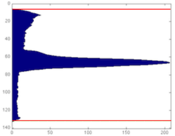
Lower Baseline

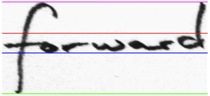


Upper Baseline



Ascender and Descender Baselines





Vertical Scaling

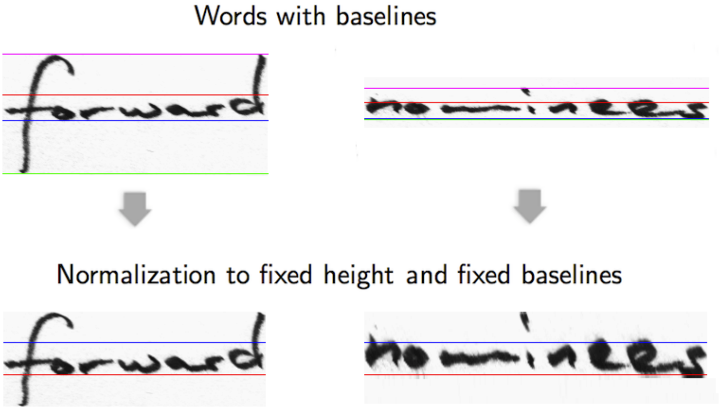


Figure: Examples of vertical scaling process

Skeletonization

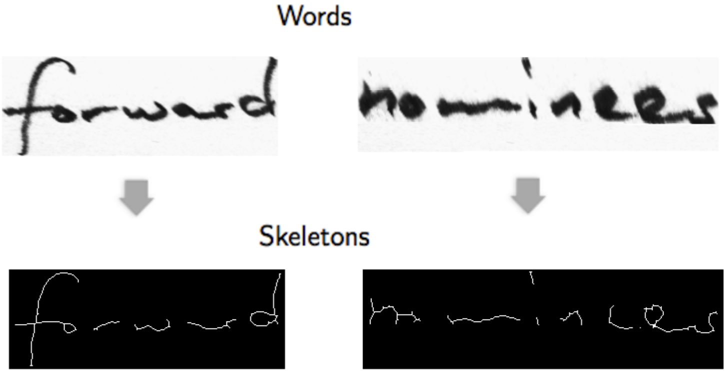
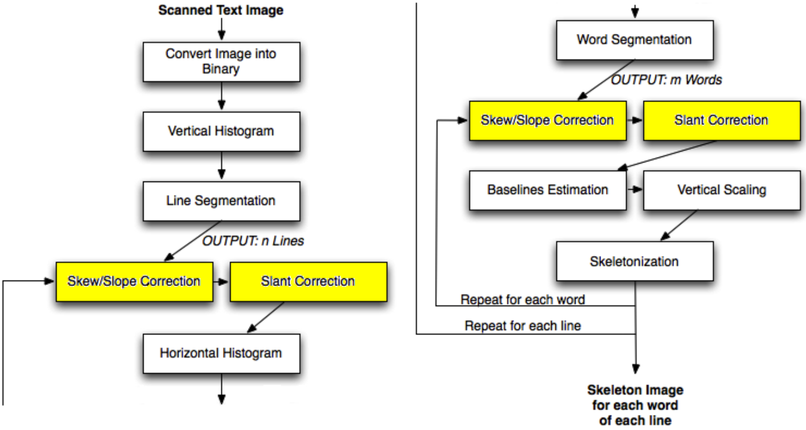


Figure: Skeletonization process

Remembering the entire pipeline....

Why repeating skew and slant correction twice?



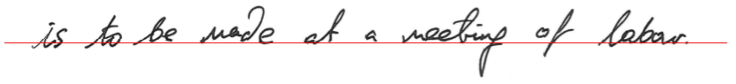
The normalizations are necessary for:

First

Words segmentation

Second

Not all the words have the same slope, slant and lower baseline



Implementation Details

Feature Extraction

Features

Extracted from the skeleton of the words.

Mainly 2 types:

- Statistical
- Morphological

Statistical Features

Percentage of white pixels in the 3 zones of the word:

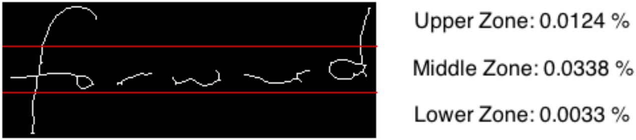


Figure: Example

Morphological Features

Obtained by **connected component analysis**.

- A connected component it is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices.

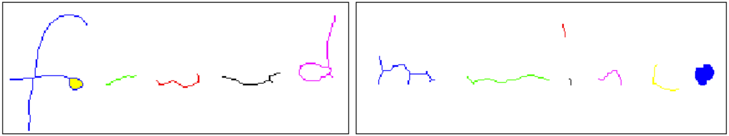
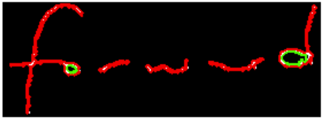


Figure: Example of words divided in different components (colors)

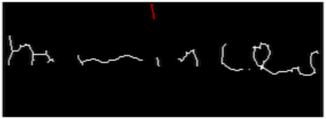
Morphological Features

We extract:

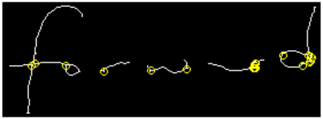
Loops



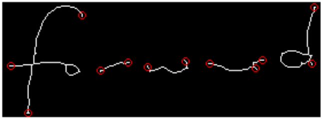
Dots



Junctions



Endpoints

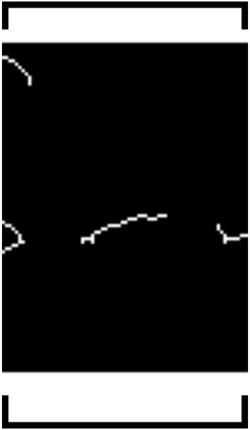


Sliding window

We use a sliding window over the words and at every shift of the window we extract features.

Sliding window

We use a sliding window over the words and at every shift of the window we extract features.



Sliding window

We use a sliding window over the words and at every shift of the window we extract features.

Sliding window

We use a sliding window over the words and at every shift of the window we extract features.

Implementation Details

Hidden-Markov Model for Word Recognition

HMM

- A set of N states $S = (s_1, s_2, \ldots, s_N)$, where the state of the system at time t is denoted q_t
- A set of priors $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$, providing the probability $P(q_1 = s_i)$.
- A transition function $\mathbf{A}$, where $a_{ij} = P(q_{t+1} = s_j | q_t = s_i)$.
- An observation function $\mathbf{B}$, mapping each observation at every state to a probability $b_i(\mathbf{o}_t) = P(\mathbf{o}_t | q_t = s_i, \lambda)$, where λ denotes the model parameters.

The model is trained to estimate the posterior probability $P(\mathbf{O}|\lambda)$ of an observation sequence $\mathbf{O}$, with D-dimensional observation vectors $\mathbf{o}_t = (o_1, o_2, \ldots, o_D)$.

HMM

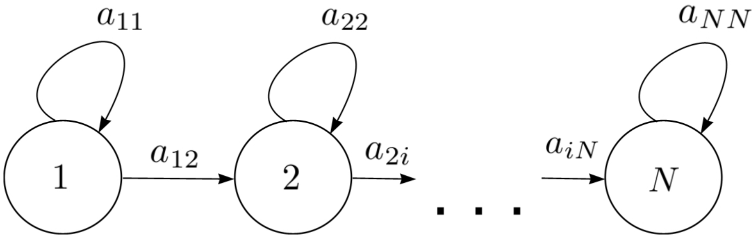


Figure: Left-to-right HMM with N states

Main problems in an HMM

1. The probability of an observation sequence, given the model, $P(\mathbf{O}|\lambda)$.
2. The most likely parameters of the model $\lambda^* = \max P(X|\lambda)$, given a training set of M observation sequences $X = (\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_M)$.
3. The most likely state sequence, underlying a given observation sequence and the model, $Q^* = \max P(Q|\mathbf{O}, \lambda)$.

Main problems in an HMM

1. The probability of an observation sequence, given the model, $P(\mathbf{O}|\lambda)$.

   Sum-product algorithm: forward-backward algorithm

2. The most likely parameters of the model $\lambda^* = \max P(X|\lambda)$, given a training set of M observation sequences $X = (\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_M)$.

   EM-algorithm: Baum-Welch reestimation

3. The most likely state sequence, underlying a given observation sequence and the model, $Q^* = \max P(Q|\mathbf{O}, \lambda)$.

   Dynamic programming: Viterbi algorithm

Updating the parameters

Comparison with GMM:

Model:	**GMM**	**HMM**
Model parameters:	$\lambda = \pi, \mu, \Sigma$	$\lambda = \pi, \mathbf{A}, \mathbf{B}$
Hyper parameters:	Number of components	Topology (states, transitions), observation function
Observed variables:	Data points	Observations
Latent variables:	Priors of a component	State sequence

Updating parameters

Model:	**GMM**	**HMM**
E-step:	Estimate the probability of a component, given the data and current parameters.	Estimate the probability of being in a state at a timestep and the probability of transfering from a state to another state.
M-step:	Maximise π, μ and Σ.	Maximise π, $\mathbf{A}$ and $\mathbf{B}$

Problems we ran into - 1

Consider the following observation:

$$\mathbf{O} = \begin{pmatrix} -1 & 0 \\ 2 & 0 \\ 1 & 0 \\ -2 & 0 \end{pmatrix}$$

Problems we ran into - 1

Singularities. Consider the following observation:

$$\mathbf{O} = \begin{pmatrix} -1 & 0 \\ 2 & 0 \\ 1 & 0 \\ -2 & 0 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 3\frac{1}{3} & 0 \\ 0 & 0 \end{pmatrix}$$

Problems we ran into - 1

Singularities

$$\mathbf{O} = \begin{pmatrix} -1 & 0 \\ 2 & 0 \\ 1 & 0 \\ -2 & 0 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 3\frac{1}{3} & 0 \\ 0 & 0 \end{pmatrix}$$

$$|\Sigma| = 0$$

Possible solution: Add some random noise.

Problems we ran into - 2

Short words may be more difficult to recognise as they are more sensible to inter-writer variations.

Possible solution: Use Mixture of Gaussians

Mixture of Gaussians

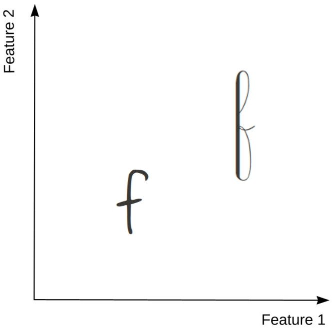


Figure: Two letters plotted in a two dimensional feature space

Mixture of Gaussians

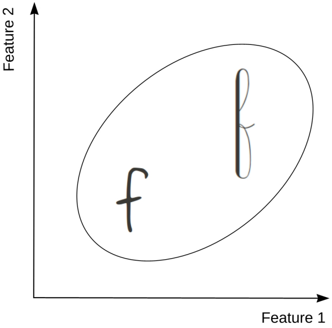


Figure: Two letters plotted in a two dimensional feature space

Mixture of Gaussians

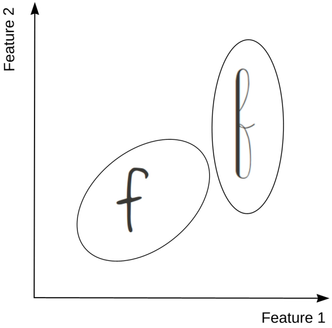


Figure: Two letters plotted in a two dimensional feature space

Experiments and Results

Experiments

We run 2 type of experiments:

- 1 vs 1 word recognition
  - We build models for 2 words and we test both models for new instances of the 2 words from novel authors
  - *GOAL*: Test how much the likelihoods of the 2 words differ
- all vs all words recognition
  - We build models for every word and for every word in the test set we rank all the models by loglikelihood
  - *GOAL*: Compute general accuracy for a small dataset

For both experiments the model of a word is built using 30 samples from 30 authors of the word.

Experiment 1 - Typewritten words

- Done to check the correctness of the pipeline.
- Used words 'Letter' and 'Number' (same length)

Table: Values of likelihoods averaged over 40 runs

Features	States	G.Comp.	Letter	Number	Diff.
Intensity	6	1	121.68	106.36	15.32
All	6	1	795.45	14.08	781.37
All	1	1	602.22	-90.02	692.22
All	6	3	226.96	-820.30	$1.04e^3$
All	6	1, diagonal cov	871.16	531.27	487.75
All	6	1, isotropc cov	-179.00	-272.23	197.35

...many many more parameters. (Interval sliding window, width of sliding window, amount of noise, ... etc.)

Experiment 1 - Handwritten Words (same length)

- Built models for 'Before' and 'People'
- Tested the models on new instances of 'Before'

Table: Values of likelihoods averaged over 40 runs

States	G. Comp.	'Before'	'People'	Difference
6	1	$1.17e+03$	$1.19e+03$	480.16
1	1	699.56	$1.0124e+03$	422.78

Experiment 1 - Handwritten Words (short vs long)

- Built models for 'At' and 'Government'
- Tested the models on new instances of 'At'

Table: Values of likelihoods averaged over 40 runs

States	G. Comp.	'At'	'Government'	Difference
6	1	441.89	$2.80e+03$	$2.36e^3$
1	1	216.47	$1.65e+03$	$1.44e^3$

Experiments 2

- Used a small dataset of 100 words
- Tested on 200 words of novel authors

Num. States	Num. Gaussian Comp.	Accuracy 1[2]	Accuracy 2[3]
lengthWord	1	0.2400	0.3400
1	1	0.2000	0.3600
1	5	0.0800	0.1800
lengthWord	5	0.1200	0.3000

[2] Correct if in 1st position of the ranked list
[3] Correct if in 1st or 2nd position of the ranked list

Issue with recognition

Short words seem to be harder to find.
Possible causes:

- During the testing, the model does not execute a complete sequence, e.g. when looking for 'the' in 'therefore' it might end before 'r'.

Possible solutions: Force the model to end the observations sequence in the final state.

Conclusions

Conclusions about the AI project

- The full pipeline is working, however a lot of improvements are possible:
  - Extract more features
  - Eventually apply PCA to the feature vector
  - Optimize the parameters of the HMM (i.e. using a validation set).
- Built models for letters instead of models for words (it requires letters segmentation)
- Use a language model

Personal Evaluation

- We built a full working pipeline
- We read and learnt a lot about a new topic
- We had the chance to apply a lot of techniques that we had only been studied in theory
- We improved our skills in programming

Questions?