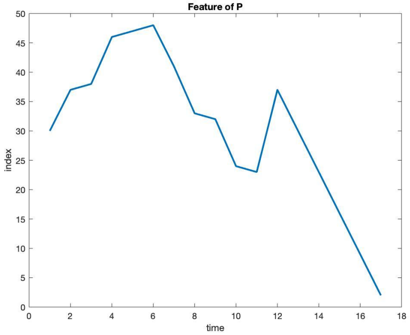
Character Recognition with HMM

Alfred Krister Ulvog + Mar Balibrea Rull

Application

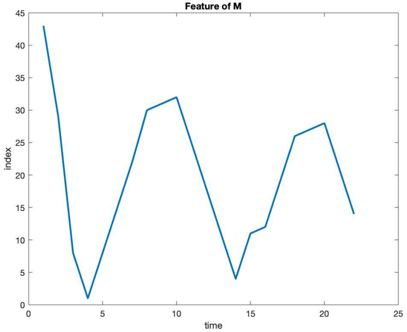
- Password system
- Correct characters + correct handwriting
- Classifiers would be trained for every individual separately
- Characters used: lowercase letters

`td_m_label{1, 1}`



`td_m_ft{1, 1}`

`td_p_label{1, 1}`



`td_p_ft{1, 1}`

Feature extraction scheme

2D integers | 2D integers | 2D integers | Scalar integer | **Scalar integer**

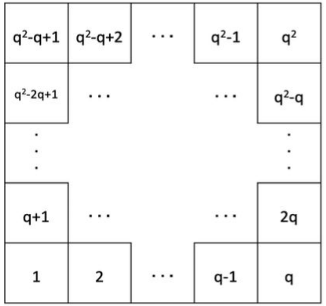
Redundancy removal
Removing not drawing coordinates.

Centering
Subtracting the center in x and y axis.
Robustness to position.

Normalization
Dividing my maximum range.
Robustness to size.

Quantization / pixelation
Assigning index based on position (unidimensional quantization).

Redundancy removal
Removing index repetitions.
Robustness to mouse speed.



Quantization Scheme (q = 7)

Training phase

- 25 realizations of all characters
  (20 training data + 5 test data for each character)
- No distinction because passwords are random
- One HMM for each class (character)
- Baum-Welch for each HMM
  - Minimum of 10 Iterations
  - Iterate until 1% KL improvement

Our HMM Design and Motivation

- LeftRightHMM for each character:
  We wanted a state to have some meaning.
  By forcing LeftRightHMM, each state will represent certain parts of letters/strokes.
- # of States:
  Varies with characters. Average length of sequence divided by 3.6 and round up (we will explain why).
- Output distribution: `DiscreteD(ones(1, 49))`
  The possible output from our feature vector is between 1 and 49.

Initial Model

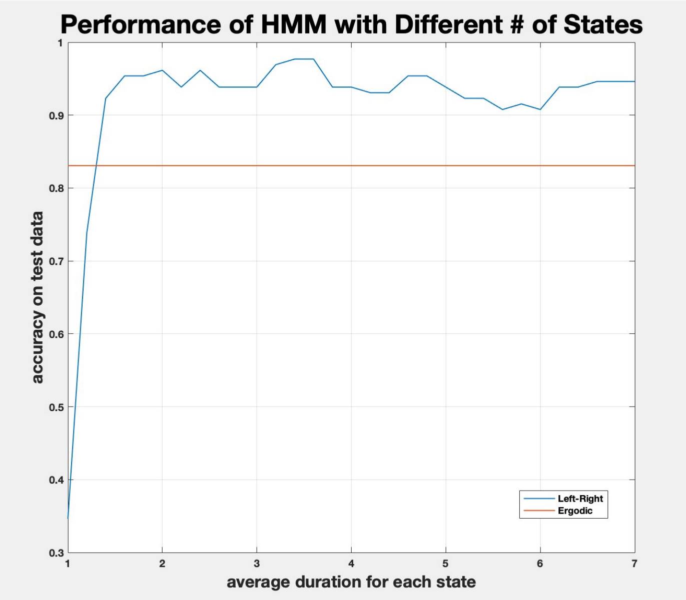
- Left-Right HMM for each class
- Fixed # of States for each character: 7
- Performance on the test data: 93.08% (121 / 130)

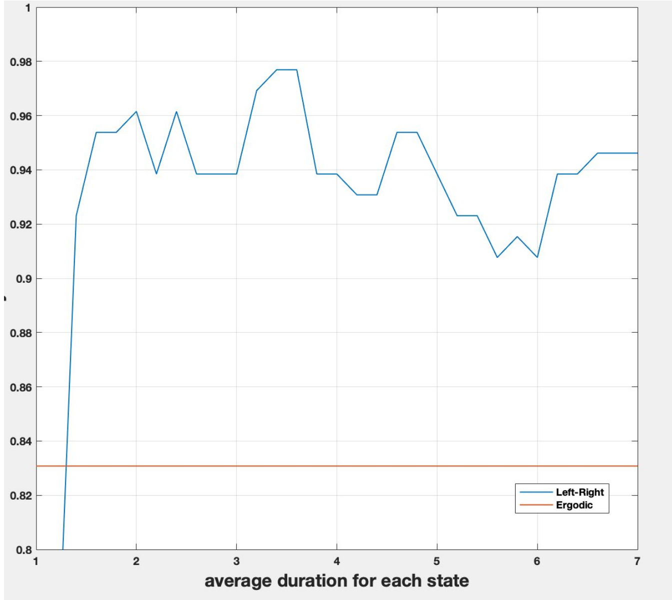
- Can we improve?

  One character is more complex than the other... (e.g., “g” and “i”)

$$\text{\# of States} = \frac{\text{Average Length of Feature Vector}}{\text{Average Duration of Each State}}$$

Ergodic or Left-Right? and # of States?





- Left-Right: Peak Rate at 97.69% (127 / 130)
- Ergodic: Flat Rate at 83.08% (108 / 130)

  Ergodic Markov Chain’s performance did not vary with number of states.

Key points we noticed during training

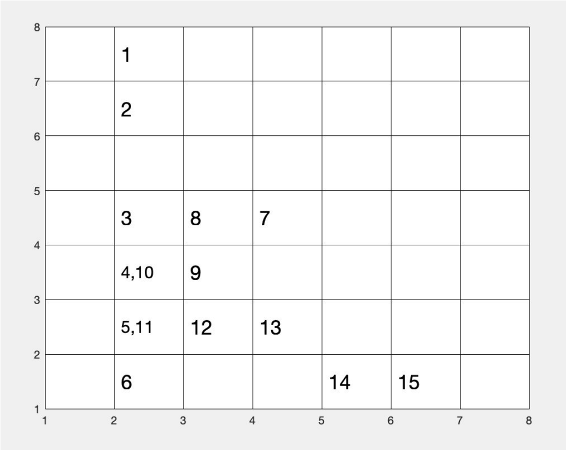
- Forward / backward algorithm had to be revised so that the the variables are not divided by 0 (once one of the variable becomes “NaN”, the model cannot learn anymore).
- Accuracy became much higher when training data had diversely distorted characters.

Demo

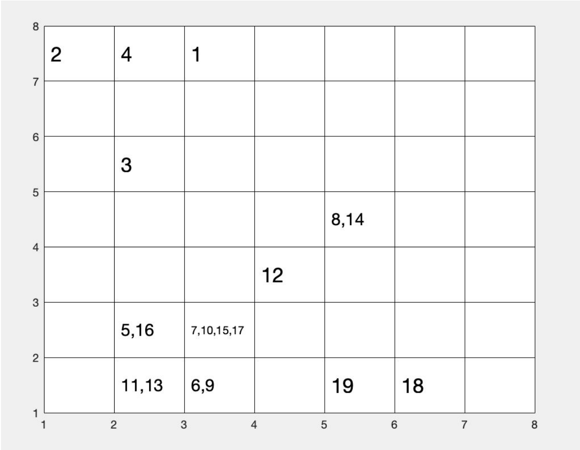
Visual results of learning

Random Output from HMM Trained for letter “k”

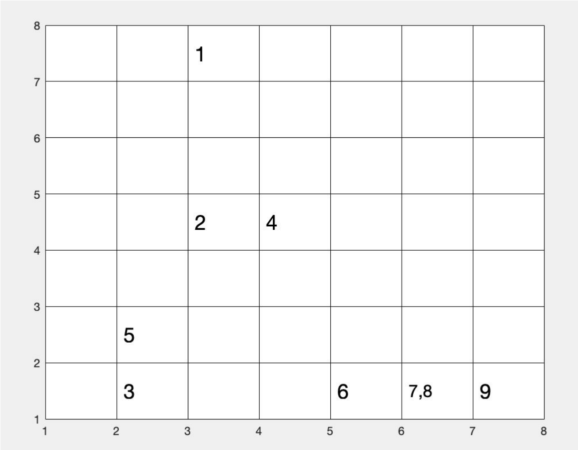
Feature Vector from Training Data



Random Output 1



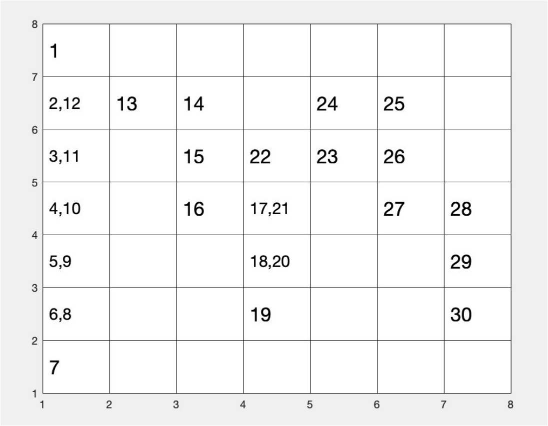
Random Output 2



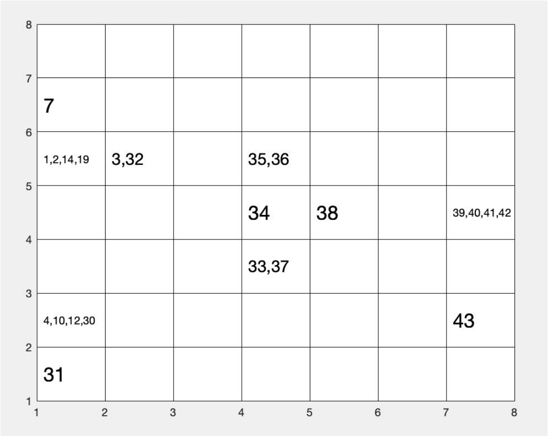
Visual results of learning (cont.)

Random Output from HMM Trained for letter “m”

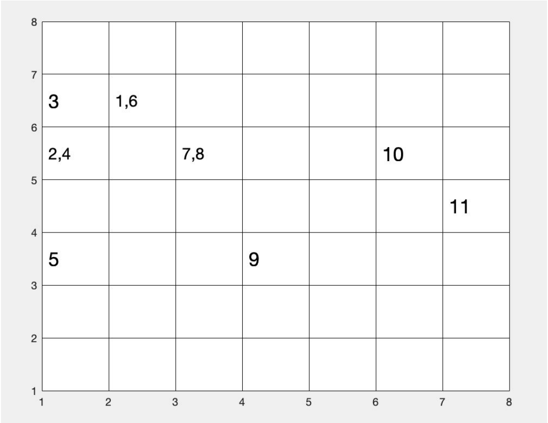
Feature Vector from Training Data



Random Output 1



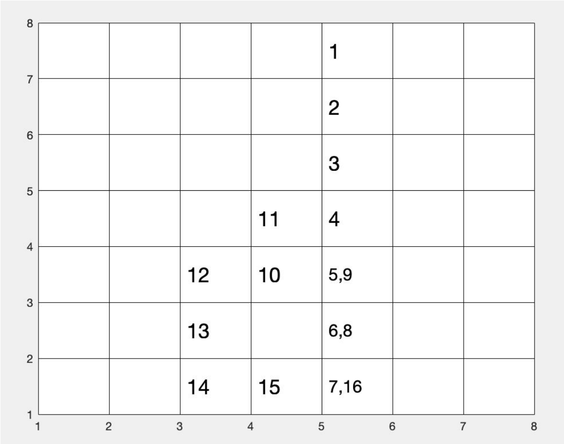
Random Output 2



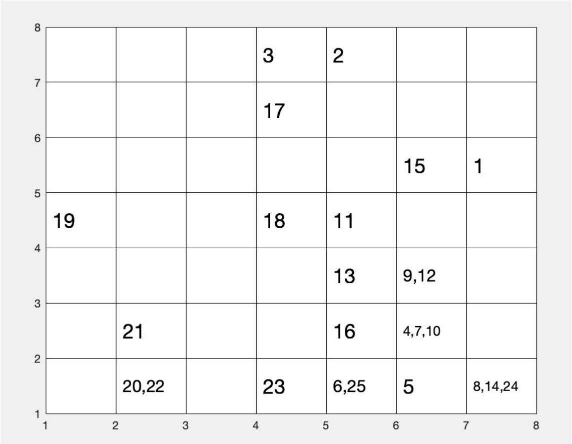
Visual results of learning (cont.)

Random Output from HMM Trained for letter “d”

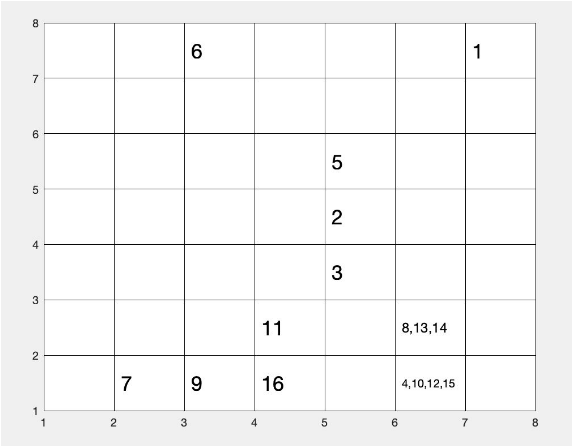
Feature Vector from Training Data



Random Output 1



Random Output 2



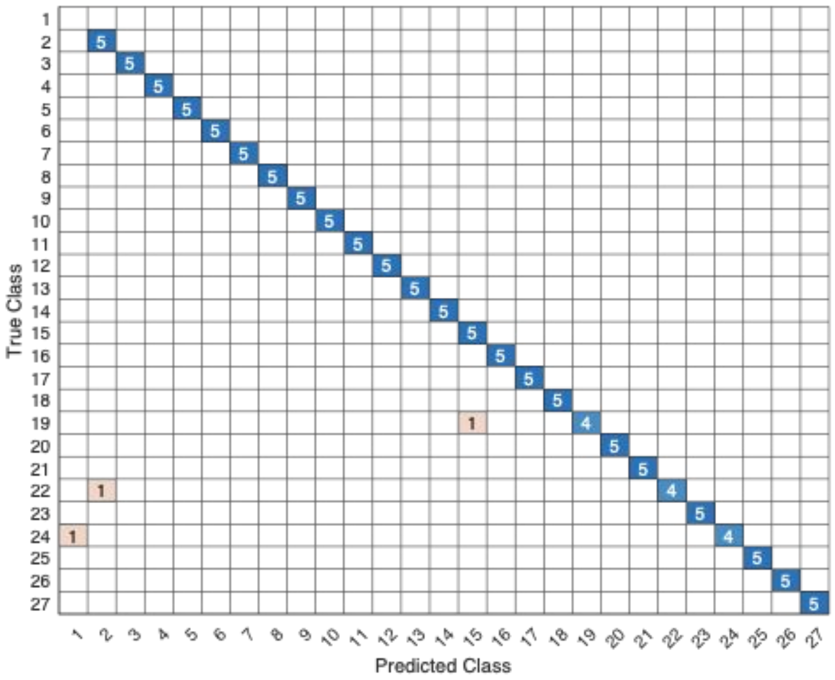
Classification accuracy metrics

- Accuracy over the test set: **97.69% (127 / 130)**
- Confusion matrix shows there's just 3 errors:

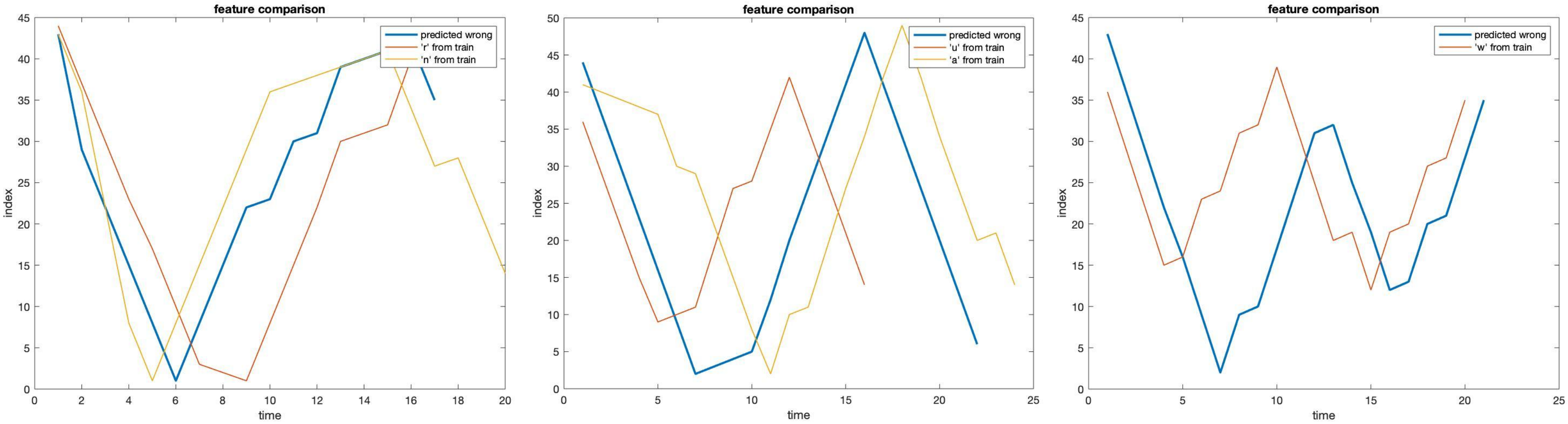
Misclassified as 'n'

Misclassified as 'a'

Misclassified as none



Examples of misclassified instances



Graphs with the wrongly predicted feature (test), a feature from the correct label (train) and a feature from the wrongly predicted label (train).

Conclusions

- With the right kind of structure for HMMs, high accuracy for character recognition were achieved despite we had limited training data.
- Increasing in training data may improve our model to have higher accuracy.

- We must make a careful decision when constructing the models (number of states, ergodic/left-right, etc.).
- Make sure the training data is diverse.

Any questions?

Alfred Krister Ulvog + Mar Balibrea Rull

This presentation and the code will be submitted in the assignment on Canvas.