Abstract

Hidden Markov models are frequently used in handwriting-recognition applications. While a large number of methodological variants have been developed to accommodate different use cases, the core concepts have not been changed much. In this paper, we develop a number of datasets to benchmark our own implementation as well as various other tool kits. We introduce a gradual scale of difficulty that allows comparison of datasets in terms of separability of classes. Two experiments are performed to review the basic HMM functions, especially aimed at evaluating the role of the transition probability matrix. We found that the transition matrix may be far less important than the observation probabilities. Furthermore, the traditional training methods are not always able to find the proper (true) topology of the transition matrix. These findings support the view that the quality of the features may require more attention than the aspect of temporal modelling addressed by HMMs.

1 Introduction

In 1989, Rabiner published the seminal work (Rabiner, 1989) on hidden Markov models (HMMs), with applications in speech recognition. Since then, HMMs have been used in other domains as well, such as segmenting gene sequences (Eddy, 1998) and handwriting recognition (Bunke et al., 1995; Plötz and Fink, 2009).
In this paper, we will discuss the applications in this last domain and HMMs in general.

There is a large number of variations of the regular HMMs that Rabiner wrote about, ranging from pseudo 2D-HMMs (Kuo and Agazzi, 1993), to truly 2D-HMMs (Markov random fields) (Park and Lee, 1998) and explicit duration modelling (Benouareth et al., 2008), to nested HMMs (Borkar et al., 2001) and many more. In the core, these variations are still HMMs, usually trained using the Baum-Welch algorithm. When the data is already labelled with hidden states, however, the transition probability matrix can be modelled directly, without using the potentially more unpredictable EM-based approach. This is the case in segmenting gene sequences with profile HMMs (Eddy, 1998) for example, using many pattern heuristics to identify state-transitions in the sequence.

The overall HMM architecture (e.g., determining the number of states, transition matrix topology and integrating it into a larger framework) requires a lot of human effort. However, to our knowledge, no real benchmark has been proposed to test algorithm variants of HMM implementations. In section 2, we will discuss how such a benchmark can be constructed. It will not only provide a way to compare results, but also allow one to determine the difficulty of a particular dataset.

The goal of this paper is to investigate the core of HMMs. HMMs consist of three main components: the initial state probability distribution ($\pi$), the transition probability matrix ($A$) and the observation probability functions ($B$). While the role of the initial state probability distribution is known to be of relatively small importance (especially in left-right topologies such as Bakis, since these models always start in the first state), it is hard to find concrete information on the relative importance of the transition and observation probabilities for optimal performance in the literature.

Artières et al. (2002) mention in passing the importance of the observation probabilities over the transition matrix. However, the study does not provide further information. Therefore, in
section 4 an experiment is presented to gain a better insight in the importance of the transition matrix. It will show that, indeed, the observation probabilities are very important. The implications of this observation and the consequences for using HMMs as a paradigm in handwriting recognition are discussed in the final section.

We will show, using generated data, that it is very difficult for the Baum-Welch algorithm to find the correct topology of the underlying Markovian process. By generating data according to a known Markov process with very specific properties (namely a left-right HMM), we know which properties the ergodic model, initially without any restrictions, should get after training. We can now show that the explicitly coded left-right topology is not found by an ergodic model. See also Figueiredo and Jain (2002) for a discussion of the brittleness of EM algorithms.

Finally, we show that, surprisingly, removing the temporal information from an HMM does not necessarily have a large impact on performance in a real-world problem.

2 BENCHMARK

We will run some experiments using our own implementation as well as other HMM toolkits on a generated data set as a benchmark. It is hard to find a proper HMM benchmark for discrete, one dimensional data that has a gradual scale of increasing difficulty. The dataset that was generated for this purpose has varying degrees of symbol lexicon overlap between classes, making the completely overlapping set most difficult and the dataset with the largest between-class distance least difficult. This is useful for comparing performances between runs on different feature methods, having the ability to attach a ‘difficulty index’ to each.

The generated data contains 100 classes, each class consists of generated transition and observation matrices. The transition
matrix is a randomly\textsuperscript{1} initialised Bakis model with $N_{\text{states}} = 10$ states, which is appropriate for variable duration modeling of left-right symbol sequences. The observation probability functions, with $N_{\text{symbols}} = 20$ symbols, are also instantiated randomly. The topology was chosen as Bakis in this benchmark. Most HMM implementations do not have restrictions on topology, except the dHMM framework (which uses a fixed, hard-coded Bakis structure). See also section 3 for more details on different topologies.

The gradual scale of difficulty is achieved by having multiple data sets with a varying degree of separability in symbol space. Concretely, this means that there is an overlap in lexicons between classes. A separability of $\delta$ of a dataset is defined by the following equation:

\begin{align*}
L_1 &= \{1 \ldots N_s\} \\
L_i &= \{L_{i-1,0} + \delta \ldots L_{i-1,0} + \delta + N_s\} \quad (3.1)
\end{align*}

Where $\delta$ is the separability, $L_i$ is the lexicon, the set of symbols, to be used for class $i$, $L_{i,j}$ is the $j^{\text{th}}$ element of $L_i$ and $N_s$ is the size of the lexicon, i.e., number of symbols per class. A separability of $\delta = 0$ is the most difficult case, because all classes share the same set of symbols: $L_1 = L_2 = \{a, b, c\}$. A separability of $\delta = 1$ means that between classes, one symbol is not re-used in the next class: $L_1 = \{a, b, c\}$ and $L_2 = \{b, c, d\}$, and so on. With more separation than symbols, a gap between the symbols is present: A dataset with $L_1 = \{a, b, c\}$ and $L_2 = \{e, f, g\}$ has a separation of $\delta = 4$.

In this section, we will show the results of running several HMM frameworks on the generated datasets. We test the popular HTK tool kit, which is well known in speech recognition (Young et al., 2006); GHMM, developed mainly for bio-informatics applications (ghm, 2003); a framework developed by Myers and Whitson (Myers and Whitson, 1994), dubbed dHMM here, mainly for discrete Bakis models for automatic speech recognition; and finally our own framework, developed from scratch to review in great detail the algorithmic details of HMMs, dubbed jpHMM.

\textsuperscript{1} Using the default python module random, which uses a Mersenne twister pseudorandom number generator
Table 3.1: Average classification performance (%) of three randomly initialised runs on the same dataset. Please note that the standard deviation for dHMM and GHMM is 0 due to the use of a static random seed, instead of a random seed. HTK-hinit uses the hinit tool to initialise the model with some estimates from the data, which increases performance only slightly on these datasets. The very small difference between a separability of $\delta = 10$ and $\delta = 20$ is not visible in this table. $N_{\text{states}} = 10$, $N_{\text{symbols}} = 20$

<table>
<thead>
<tr>
<th>Separability ($\delta$)</th>
<th>jPHMM</th>
<th>dHMM</th>
<th>GHMM</th>
<th>HTK</th>
<th>HTK-hinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1% (± 0.10)</td>
<td>1%</td>
<td>1%</td>
<td>1% (± 0.12)</td>
<td>1% (± 0.06)</td>
</tr>
<tr>
<td>1</td>
<td>41% (± 0.46)</td>
<td>40%</td>
<td>37%</td>
<td>41% (± 0.12)</td>
<td>41% (± 0.62)</td>
</tr>
<tr>
<td>2</td>
<td>66% (± 0.38)</td>
<td>64%</td>
<td>61%</td>
<td>66% (± 0.10)</td>
<td>66% (± 0.15)</td>
</tr>
<tr>
<td>3</td>
<td>81% (± 0.10)</td>
<td>78%</td>
<td>76%</td>
<td>80% (± 0.10)</td>
<td>80% (± 0.10)</td>
</tr>
<tr>
<td>5</td>
<td>95% (± 0.25)</td>
<td>93%</td>
<td>92%</td>
<td>94% (± 0.17)</td>
<td>94% (± 0.15)</td>
</tr>
<tr>
<td>10</td>
<td>100% (± 0.00)</td>
<td>100%</td>
<td>100%</td>
<td>100% (± 0.00)</td>
<td>100% (± 0.00)</td>
</tr>
<tr>
<td>20</td>
<td>100% (± 0.00)</td>
<td>100%</td>
<td>100%</td>
<td>100% (± 0.00)</td>
<td>100% (± 0.00)</td>
</tr>
</tbody>
</table>

We also use the HTK toolkit together with the hinit tool to have a better initialised model, compared to random initialisation.

The benchmark datasets in this paper are all synthesized and discrete. Also, the duration of the sequences is limited. This means that the results of the current study can not directly be compared to all possible applications. However, there is no fundamental limitation on sequence length, or number of states. This can be addressed in future releases of the benchmark. For some applications and features, continuous observation modelling is beneficial (Chen et al., 1995), while for other applications and feature methods, discrete observation modelling is still very relevant (Rigoll et al., 1996). In order to study the core details of HMMs, using discrete observations is interesting, since its modelling is almost trivial. Common techniques to use discrete models on continuous data are vector quantization (Schenk et al., 2008), $k$-means clustering, or self-organizing maps (see also section 4).

The generated datasets have a separability of $\delta \in \{0, 1, 2, 3, 5, 10, 20\}$. The number of states is $N_{\text{states}} = 10$, $N_{\text{symbols}} = 20$, the length of each sequence is $|\vec{O}| = 10$ observations, yielding effectively an artificial stochastic language with 10-letter words. We have generated 100 classes with 300 se-
quences each. We trained models from each toolkit on all classes, and performed classification based on the most likely model for an instance.

**Results** The classification performances on the seven datasets are reported in Table 3.1, showing that all implementations perform roughly equally well, which is to be expected. However, we can also see the relation between benchmark difficulty, the separability $\delta$ and classification performance for five HMM implementations. From a separability of about $\delta = 5$ onward (for a dataset with $N_{\text{states}} = 10$ and $N_{\text{symbols}} = 20$) classification becomes very accurate.

3 LEARNING THE TOPOLOGY OF A TRANSITION MATRIX

In this section and the next, we describe two experiments to determine the importance of temporal modelling which is effectuated by the transition matrix in the HMM framework. The first experiment is mainly focused on the performance of the Baum-Welch algorithm, while the second shows what happens when the temporal information is removed from an HMM.

The Baum-Welch algorithm, an Expectation-Maximisation (EM) algorithm, works by initialising a model, often by using random probabilities, and then incrementally improving it. The initialisation step is very important due to the possibility of ending up in a local maximum, and the ‘random’ method is therefore very brittle, requiring human supervision.

As a first experiment to examine the transition matrix, we generate artificial data again. This has the advantage that we explicitly know the properties of the transition matrix. The specific property that we are interested in, currently, is the topology of the model. The topology is the shape of the transition matrix and there are a number of topologies possible. The most well-known is the Bakis topology, which is a left-right model that defines for each state two transition probabilities: to the current state and to the next state. Another topology is the Ergodic topology, which puts no a-priory restrictions on the transition probabilities: every
state has a (possible) transition to every state (including itself). See Fig. 3.1 for a graphical representation of these topologies. A variant of Bakis, that has the ability to skip a state by also having a transition probability from $S_i$ to $S_{i+2}$, was left out for brevity.

The experiment is set up as follows: a model is created by randomly initialising a Bakis topology with $N = 20$ states ($L = 20$ symbols). After generating 300 instances of 40 observations long with this topology, a fully Ergodic model is trained on these instances. The resulting transition matrix is examined: has it learned the fact that we used a Bakis topology to generate the training data? To be fair, we shuffle the states in the trained model to have the smallest $\chi^2$ distance to the original model. The found hidden state $S_1$ in the trained model does not have to be state $S_1$ in the generating model, after all.

**Results** The original, generated model and the learned ergodic model can be visually inspected in Fig. 3.2. The transition matrix is converted to an image by taking the state-transition probability and coding it into a grey-scale colour: a probability of 0 is rendered as white, while a probability of 1 is rendered as black. From these figures, we can see that the Bakis topology has a diagonal structure: a probability from state $S_i$ to $S_i$ and to state $S_{i+1}$. The learned, ergodic model does not show a diagonal structure at all, even though we shuffled the matrix to have the smallest $\chi^2$ distance to the generated Bakis model. The learned model is significantly different from the generated Bakis
model \((p \ll 0.0001, \chi^2 = 27863, 19\) degrees of freedom), using a contingency table test on the transition frequencies\(^2\).

From this observation we could conclude that it is difficult to learn the topology of an underlying Markov process. We performed two similar experiments to verify this finding (this time with \(N = 10\) states because of compute time constraints). The first variation was done by averaging over several learned models. This is realised by generating ten Bakis models, generating 300 sequences per model and train an ergodic model on each set of sequences. The models are shuffled and averaged, and visualised in Fig. 3.3. Although we are not aware of this extensive procedure being done in the literature, it appears to be useful to see whether a diagonal pattern can be found, on average, even when it is difficult to see in a single model.

It is well known that one requires a large set of training sequences to estimate the right model. We used this idea in another method of trying to find the underlying Bakis structure using an ergodic model. Instead of averaging over ten models, we now use ten times as much data. This gives the training algorithm more data to learn the structure from. Fig. 3.4(b) shows the results of estimating the topology from 3000 sequences, that were generated using the model shown in Fig. 3.4(a).

Both Fig. 3.3 and 3.4 show that trying really hard to force an ergodic model to find the Bakis structure can result in a slight tendency towards a diagonal structure under highly artificial training conditions. The desired diagonal probabilities are present in the learned ergodic models, but the off-diagonal probabilities are abundant in these models as well. From the diagonals, the self-recurrent state-transition probabilities are most pronounced. This shows that it may be very difficult to find the underlying structure of a Markov process using the Baum-Welch algorithm (given the specific parameters). From this and pilot studies, we conclude that it is less difficult to find a diagonal structure for \(N = 10\) states than for \(N = 20\) (which is more common). We will verify this in a future study.

\(^2\) The Kolmogorov-Smirnov test cannot be used since there is no meaningful univariate axis to integrate the probabilities (Babu and Feigelson, 2006).
4 THE IMPORTANCE OF TEMPORAL MODELLING

We are also interested in what happens when we remove the temporal information from the transition matrix. This means that we create a flat topology: all transition probabilities are equally probable: $a_{ij} = \frac{1}{N}$, where $N$ is the number of states. During training, the transition matrix will continuously be made uniform (i.e., flat) in each iteration. This is necessary because the observation probabilities may no longer be correct when adjusting the transition probabilities after training. The flat topology can be viewed as an orderless “bag of states”. We will now compare how well models with this topology compare to models with a Bakis or ergodic topology.

In this experiment we train an HMM on discrete features, extracted from handwritten word images. The dataset uses a single handwritten book from the collection of the Dutch National Archives (van der Zant et al., 2008a). We use two features: fragmented connected component contours ($FCO^3$) and a sliding window, both quantized using a Kohonen self-organizing feature map (SOFM, see Kohonen, 1987).
Figure 3.3: After generating ten models with the number of states reduced to $N = 10$, per model 300 sequences were generated, otherwise similar to Fig. 3.2. New models were trained on each of these 300 sequences and the models were averaged. Fig. 3.3(a) shows the average model of the generated models, while 3.3(b) shows the average learned model, with a vague tendency towards diagonal state-transitions, mostly the self-recurrent transitions, while the next-state-transitions show a less pronounced pattern. Average $\chi^2$ distance = 16

For the $FCO^3$ feature, the image is broken up into a sequence of fragmented connected component contours (Schomaker et al., 2007). Each of these contours is then quantized into a discrete index of a SOFM of $70 \times 70$ nodes. This means the lexicon consists of 4900 symbols. We have selected 130 classes with at least 51 training instances, with a total of 30,869 labelled instances. Because the average length of the words was 4.4 $FCO^3$ observations, the number of states was chosen to be 3.

The second feature is extracted using a sliding window of 4 by 51 pixels, centered around the centroid of black pixels over the entire word zone. The SOFM for this feature, with $25 \times 25$ nodes, was a lot smaller than the $FCO^3$ feature map, due to time constraints. Centering around the centroid with a height of 51 pixels means that the outstanding sticks and (partial) loops of ascenders and descenders are still preserved, while reducing the size of the image considerably. We limited the number of classes in the experiments with this feature to 20, with a total of 4,928 labelled instances. The average length of observation sequences
Figure 3.4: Instead of averaging over ten models as in Fig. 3.3, we now use ten times as much generated sequences to train a single model (3000 sequences). We see that there is a small tendency towards diagonal (Bakis-like) state transitions, but it is not very strong. $\chi^2$ distance between the two distributions = 14

for the sliding window feature was 65.9 observations, which led us to use $N = 27$ states.

For classification, an HMM $\lambda$ is first trained on the instances of each class, and then classification can be performed using $\arg\max_{\lambda \in \Lambda} \log P(O|\lambda)$, where $\Lambda$ is the set of all trained models and $O$ is the test sequence. To investigate the role of the state-transition probabilities, we perform the experiments with three topologies: Bakis, ergodic and flat, which is the topology where all transition probabilities are equally probable. We perform the experiments on both features using 7-fold cross validation, with our own implementation, JP.HMM.

Results The results are summarised in Table 3.2 and 3.3. We can see that the results of classification with the $FC^3$ feature are very close together (and not statistically significant, with ANOVA, $p > 0.05$). There is a significant difference in the classification performance using the sliding window feature (ANOVA, $p < 0.001$), but the drop in performance is not as dramatic as would

---

3 The increase in number of states is most likely the reason for the increased time necessary for training
Table 3.2: Results of the $FCO^3$ experiment. Performances reported are averages over 7 folds, with 130 classes and at least 51 instances per class in the training set. As can be seen, all topologies perform around 60%. Flat models do not perform significantly worse.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Classification performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bakis</td>
<td>59.9% ± 0.9</td>
</tr>
<tr>
<td>Ergodic</td>
<td>59.5% ± 0.9</td>
</tr>
<tr>
<td>Flat transition probabilities</td>
<td>59.1% ± 0.8</td>
</tr>
</tbody>
</table>

Table 3.3: Results of the sliding window experiment. Performances reported are averages over 7 folds, with 20 classes and at least 51 instances per class in the training set. Differences between the topologies are statistically significant ($p < 0.001$) although the difference between the flat and ergodic topologies is not as dramatic as expected (Please note that in the Flat condition of the transition matrix, temporal information is completely uniform).

<table>
<thead>
<tr>
<th>Topology</th>
<th>Classification performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bakis</td>
<td>75.2% ± 2.0</td>
</tr>
<tr>
<td>Ergodic</td>
<td>78.5% ± 1.2</td>
</tr>
<tr>
<td>Flat transition probabilities</td>
<td>71.1% ± 1.3</td>
</tr>
</tbody>
</table>

be expected from the removal of temporal information in the Markov paradigm.

Please note that the HMMs were used as a measurement tool to find differences between transition models. They have an average performance, avoiding ceiling effects. Also, the $FCO^3$ feature is a feature developed for writer identification, not handwriting recognition per-se. The sliding window feature could be fine-tuned further by changing the size of the Kohonen map, the window, the number of states, etc. In this experiment we are interested in evaluating HMM topologies, not in maximising the recognition performance.
5 DISCUSSION

We set out to reevaluate hidden Markov models, by creating a benchmark for discrete HMMs, and running experiments to investigate the importance of the transition matrix.

Using the benchmark, we found that for discrete observations, all common HMM tools have similar performances. Furthermore, we can now measure the difficulty of discrete data, by comparing the performances of discrete HMMs with the performances of the benchmarks with different degrees of difficulty. In the future we want to extend the current study with continuous density HMMs as well.

While it is barely presented in the literature, the fact that the transition matrix is of a smaller importance than the observation probabilities is well known from personal communications at, e.g., conferences. We have done two experiments to establish the importance of the transition matrix, and found that indeed the observation probabilities have a large impact on recognition performance. The results of these experiments showed that (a) it is hard to learn the correct, known topology of the underlying Markov process and (b) that classification with the temporal information removed from the HMMs can also result in reasonably performant classifiers.

Regarding (a), it appears that the Baum-Welch training method is not very reliable to estimate the underlying transition structure in the data. As noted in (Figueiredo and Jain, 2002), EM is brittle and very sensitive to the initialisation process. We have shown that the Baum-Welch method was unable to find the Bakis topology from generated data when initialised as a full Ergodic model. We have previously studied initialisation of models to prevent local maxima (Bhowmik et al., 2011), but this still requires a lot of human modelling effort, specifically for each problem variant.

Regarding our finding (b), that classification with temporal information removed can result in performant classifiers, we believe that the observation probabilities are very important. This sup-
ports our view that the quality of the features may require more attention than the aspect of temporal modelling. From a more scientific point of view, it is still a challenge to adapt the Baum-Welch estimation algorithm to correctly estimate the Markov parameters of an observed sequential process.

Even though these findings expose limitations of HMM and its training procedure, the fact that recognition performance is not degraded dramatically when removing temporal information from HMMs implies that dynamic programming (i.e., the operational stage) is a strong principle. Also, the Markov assumption remains appealing from a theoretical perspective.

Given these considerations, we feel 1) that it may be misleading to stress the hidden aspect of HMMs, because of the relatively minor role the hidden states play in achieving good performance, 2) the Baum-Welch algorithm should be replaced with a less brittle method, and 3) although the HMM principles mentioned above are strong, there are many tricks of the trade, that are not treated well in literature (see also the Appendix).

POSTSCRIPT

In the preceding two chapters of this dissertation, we have looked at assumptions in HMMs, from a machine learning and a feature-representation perspective. It appears that properly learning the parameters of a Markovian process is not straightforward. It has been hard to replicate state-of-the-art HMM results with the datasets that are available in a large operational system such as Monk. One of the causes of lower performances on these datasets is bootstrapping: Adding new collections to the system frequently means starting from scratch, since no labels are available and the script is very different from other collections. Therefore, the Monk system uses various feature-extraction techniques and machine-learning methods, each with their own strengths.

While representation and learning methods are core components of a search engine for historical documents, they only work properly when there are enough labelled instances in order to train
them. Furthermore, labels are essential in order to assess the quality of the techniques. Researchers frequently use pre-existing labelled datasets to improve existing techniques or develop new methods. Consider for example the many competitions and benchmark datasets that are referred to in handwriting recognition and machine learning literature (Clausner et al., 2018; Strauß et al., 2018). These are useful to be able to compare different approaches, but it is easy to lose sight of the actual use-cases—in the case of Monk, the use-case is a search engine for historical document collections that are sometimes obscure and scarcely labelled.

Bootstrapping and the need of (a lot of) data are fundamental issues that should be addressed in any form of current (deep) machine learning systems. Because certain methods work better in the bootstrapping phase than when there are plenty of labelled instances across all classes, we believe that it is a good idea to apply the methods in an iterated fashion. Furthermore, we believe it is a good idea to let the machine help in the labelling process such that the most relevant instances are considered first. This is more efficient than labelling images from left to right, top to bottom. We therefore consider the handwriting recognition process to be a loop instead of a pipeline: In Monk, labelling is iterative and in each iteration new images are presented.

The iterative application of labelling can be compared to the “harvesting” of labels: By ordering the unlabelled images in such a way that the process can be performed efficiently, a snowball effect can be created. Such an effect can be marked by large jumps in the amount of labels added to the system. The jumps are created by allowing the annotators to quickly label a lot of images in one go. When a suggestion for the labels is provided and the user interface allows you to acknowledge these labels all in one go, the model improves, which in turn improves the quality of new labels, which finally allows for more labels to be added with a few clicks.

In Monk, the user interface for quickly labelling images is a hit list. The images are arranged in a list, usually presented in a table, that match a certain class. This list is ordered in such a way that
the images at the top are good enough to be acknowledged to be of this class. The interface then has two options: accept the first $N$ labels, or manually select all images that should have their labels be accepted. In the bootstrapping phase, it is frequently observed that the latter method is used by the human labellers, due to the classifier not being able to accurately find enough samples, while in later stages, the “snowball” (a critical mass of \{image, label\} tuples) has gained enough traction to provide enough samples to fill the first page with correctly labelled images.

The Monk web-based labelling system has been invaluable to the collection of labels in a number of rare manuscripts. This allowed the researchers to improve their methods in tandem with the improvements of the labelling of the collections. New performance indicators are needed that allow researchers to see how their methods affect the quality of the hit lists. The default method of a mean average precision (MAP) is not sufficient to get an indication of the quality of highest ranked samples. The next chapter will study hit lists and the functions that need to be optimized in further detail.