ROBUST NAMED ENTITY DETECTION IN VIDEOTEXT USING CHARACTER LATTICES

Krishna Subramanian, Rohit Prasad, Ehry MacRostie, Prem Natarajan

BBN Technologies, 10 Moulton Street, Cambridge, MA 02138, USA

ABSTRACT

Text in video sequences can provide key indexing information. In particular, videotext is rich in named entities (NEs) and detection of such entities is critical for search applications. Traditional approaches for detecting NEs in OCR output look for these NEs in the single-best recognition results. Due to inevitable presence of recognition errors in the single-best output, such approaches usually result in low recall. Given that a lattice is more likely to contain the correct answer, we explore NE detection from character lattices produced by our videotext OCR system. Furthermore, we use an *approximate* match criterion that allows insertion of punctuations during lookup. Experimental results show a 50% relative improvement in NE recall using lattices over exact lookup in the 1-best hypothesis. Since the improvement in recall is accompanied by a large number of false positives, we present techniques for reducing false alarms. In addition, we describe efficient techniques for reducing the time for detecting NEs.

Index Terms— Optical Character Recognition, Hidden Markov Models, Videotext, Named Entities, Character Lattices

1. INTRODUCTION

In the modern world, video is an increasingly important source of information, and the volume of collected multimedia data is expanding at a tremendous rate. Video is a content-rich medium, with content descriptors including speech transcripts, overlaid and scene text, closed captions, and embedded faces. In particular, text in video sequences provides key indexing and interpretation information in several application areas [1,2].

The first step in indexing video based on text is to recognize it. In [2], we had presented a hidden Markov model (HMM) based system for recognizing overlaid text in broadcast news videos. In this paper, we extend the work presented in [2] to extract relevant metadata for searching videos. Specifically, we focus on extraction of a pre-defined set of named entities (NEs) in videotext. Named entities have been widely used to index audio[3,4], however, to the best of our knowledge, NE extraction from videotext has never been explored.

Many attributes of videotext, such as low resolution, interlace shear, compression artifacts, and motion blur make the recognition of videotext a challenging task. As a result, the 1-best recognition output usually contains a significant amount of errors. Consequently, a mere lookup of the exact NE string in the 1-best hypothesis fails to detect NEs in a large fraction of videotext.

In this paper, we explore use of character lattices from the videotext OCR system for improving detection of NEs. We also compare exact string lookup with a softer match criterion that allows insertion of punctuation characters while looking up the NE in the 1-best or the lattice. Although approximate match on character lattices results in significant improvement in recall, it comes with an associated cost of increased number of false positives. Therefore, we investigate novel rejection techniques for reducing false alarm rates. We conclude with initial exploration of speed-ups for reducing the time for detecting named entities.

2. OVERVIEW OF HMM BASED VIDEOTEXT OCR

Our videotext OCR system is a customized version of the HMM based BBN Byblos OCR [2] system developed for recognizing text in printed documents. The BBN Byblos OCR system can be subdivided into two basic functional components: training and recognition. Both training and recognition share a common pre-processing and feature extraction stage. The pre-processing and feature extraction stage starts off by first deskewing the scanned image and then locating the positions of the text lines on the deskewed image. Next, the feature extraction program computes a feature vector, which is a function of the horizontal position within the line. First, each line of text is horizontally segmented into a sequence of thin, overlapping steps. For each frame we then compute a script-independent, feature vector that is a numerical representation of the frame.

Each character in the OCR lexicon is modeled using a multi-state, left-to-right HMMs. Each state has an associated output probability distribution over the features. The character HMMs are trained on transcribed text lines using the Expectation Maximization (EM) algorithm. Note that the model topology including number of states, and allowable transitions is typically optimized for each script.

The language model (LM) used in the BBN Byblos OCR engine is a character or word n-gram LM estimated from the character HMM training data and other available sources of text. The recognition engine performs a two-pass search. The first pass uses a bigram LM to generate a lattice of characters or words. The second pass uses a trigram LM and optionally more detailed character HMMs to generate a 1-best hypothesis, N-best hypotheses, or a lattice.

The key modification to the BBN Byblos OCR engine for recognizing videotext is in the pre-processing of videotext images. The pre-processing of videotext involves two key steps. The first step is to upsample the videotext region by a fixed factor. The upsampling is performed to mitigate the effect of low resolution of videotext. The second step is to binarize the color text images into black text on white background or vice-versa, depending on the text and background characteristics.

Following binarization, we extract the same set of features from videotext as for machine-printed OCR [4]. Next, the two-pass recognition strategy described earlier is used to recognize *all* I-frames for text regions in a development set. Then, we empirically determine the I-frame that results in lowest character error rate (CER). On a validation set as well as for the runtime system, the

recognition result from the empirically determined lowest CER I-frame is used for evaluating performance.

3. VIDEOTEXT OCR CORPUS

The results reported in this paper are performed on overlaid videotext data collected from English Broadcast News videos. For our experiments we used the TDT-2 corpus of CNN and ABC news broadcasts recorded in 1998. We annotated text region boundaries and frame spans manually. Each text region consisted of a single line of text with possibly multiple words. A single transcription ground truth value was assigned to each text region. Approximately 7 hours of video each from CNN and ABC was manually annotated. All text was annotated except for the moving text crawler in the CNN videos.

The text density in CNN was significantly higher for CNN than for ABC: 6.6 text regions per frame versus 2.1 text regions per frame. The corpus therefore contained significantly more CNN text data. Specifically, for CNN we annotated 16,719 text regions and for ABC 5,567 text regions were annotated. We held out a fair development set of 871 regions for CNN and 475 regions for ABC – none of the regions in the development set were included in the training set.

4. NAMED ENTITY LOOKUP USING LATTICES

Spotting the occurrence of *known* or pre-defined named entities in electronic text data is a straightforward task. In contrast, spotting such occurrences in automatically generated transcriptions of speech data or videotext data is complicated due to the inevitable presence of errors in the transcriptions. While N-best lists can increase the accuracy of name spotting, they suffer from the drawback that if one portion of a multi-word named entity is correctly recognized in one of the N hypotheses and the remainder is correctly recognized in a separate hypothesis, then the spotting algorithm does not have access to the entire correct hypothesis even though the required information is contained within the N best. Character lattices provide a way to overcome this limitation without limiting the vocabulary size.

While looking up a NE in a character lattice, we are essentially looking for a path in the lattice with the sequence of characters comprising of the NE of interest. We used the depth-first traversal technique to lookup the NE in the lattice. Our first experiments for lookup used an exact string match criterion. However, analysis of the recognition errors revealed that a significant fraction of errors occur due to punctuation insertions. If we were to perform exact search on a lattice with too many such insertions, the lookup is likely to fail in detecting NEs for a large fraction of text region. Therefore, to improve the recall, we implemented a soft match criterion (referred to as "approximate" match). In approximate match, we ignore a pre-defined set of "do not care" characters while searching for a NE in the lattice.

In Table 1, we compare the NE recall performance using the exact and the approximate search for three different sources of data: 1-Best hypothesis, 100-Best hypotheses, and decoder lattice. We manually identified a list of 306 named entities of interest for comparing performance. There were 415 instances of these named entities in 1336 text regions.

The 1-best, 100-best, and character lattices were generated using a character tied-mixture HMMs trained on 22K text regions in the training data. For each text region in the training corpus, we included 5 uniformly selected instances for character HMM estimation. This was done to increase the coverage of different types of distortions that manifest themselves over the lifetime of a text region. All training images were binarized using a threshold on pixel intensity. This threshold was chosen to be 80th percentile for high intensity text and 20th percentile for low intensity text. A trigram character LM was estimated from the same training data. Including the punctuations and numerals, the recognition lexicon consisted of 86 characters. Each character HMM had an associated 512 Gaussian mixtures for modeling the output feature distribution at each state. Decoding the 5th (or last) I-frame results in the best performance on the test data. The character error rate (CER) of the 1-best hypothesis for the 5th I-frame instance was 16.6%.

Search	Source	#Recall	Recall (%)	# FA
Exact	1-Best	215	51.7	34
	100-Best	290	69.7	86
	Lattice	308	74.0	916
Approx	1-Best	245	58.9	4
	100-Best	306	73.6	131
	Lattice	322	77.4	1135

Table 1: NE recall performance from using exact and approximate search in the 1-Best, 100-Best and decoder lattice

As shown in Table 1, the %Recall using character lattices improves to 50% using approximate match on the lattice compared to the exact match using 1-Best.

5. REDUCING FALSE ALARMS

From Table 1, we observe that as we search for NEs in larger sets of hypotheses as in character lattices, we not only increase the number of recalled NEs but also increase the number of false accepts (FA). The number of FAs grow exponentially with a linear increase in the number of recalled NEs in going from 1-Best to 100-Best and finally to lattices. Therefore, the task of NE rejection is critical for any NE based application.

In speech recognition, posterior probabilities have been shown to be effective in rejecting out-of-vocabulary (OOV) words. We use the same principle for reducing false alarms in NE lookup. In our approach, we used character posteriors as features for classifying recalled NE from falsely accepted NE. The classifier is based on comparing an objective function against a configurable threshold for trading-off false alarms versus false rejects. Any NE hypothesis whose objective function evaluates to value higher than the threshold is accepted or else rejected.

We explored three different objective functions for rejecting a NE based on set of posterior features $\{p_i\}$. The set of posterior features are the character/arc posteriors of NE derived from the character lattice:

1. Minimum, i.e., $o(\{p_i\}) = \min(\{p_i\})$
2. Average, i.e., $o(\{p_i\}) = \text{average}(\{p_i\})$
3. Median, i.e., $o(\{p_i\}) = \text{median}(\{p_i\})$

The receiver operating characteristics (ROC) curves in Figure 1 illustrate the rejection performance of the three objective functions using lattice based lookup with approximate match criterion. From the ROC curves in Figure 1, we see that overall, the performance of the Average objective function is better than the Median and the Minimum objective functions. However, the Minimum objective function is significantly better than the Median in regions of low-FA but worse than the Median objective function for high-FA. For

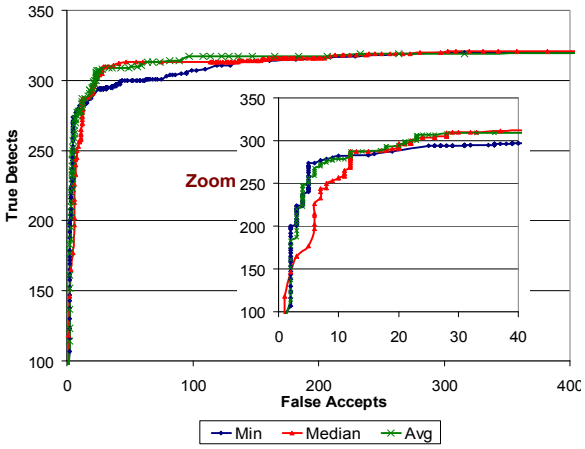

Figure 1: ROC curves comparing the performance of the three objective functions, Min, Median and Average.

very high-FA, the performance of all the three objective functions is comparable.

In Table 2, we show operating points on the ROC curve generated using the Average objective function. We compare NE lookup in 1-best and lattices. For each sample point, we list the threshold, τ, the true detects, and the false accepts. From this table, we see that by increasing the threshold, we can significantly reduce the number of false accepts while retaining most of the recalled

Source	τ	#TD	TD (%)	# FA
Lattice	0.00	322	79.3	1135
Lattice	0.10	317	76.2	129
Lattice	0.50	287	66.7	12
Lattice	0.73	248	62.0	4
1-Best	0.00	245	58.9	4

Table 2: Sample operating points on the ROC curve generated using the Average objective function.

NEs. Also, for the same number of false accepts, we see that using lattices for NE lookup results in more number of correctly detected NEs compared to using 1-best hypotheses.

6. SPEED-UPS FOR LATTICE BASED NE LOOKUP

Searching for NEs in lattices is computationally intensive and hence time consuming. In this section, we discuss some of the techniques that we have explored to speed up search for NEs in significantly dense lattices. Unless otherwise mentioned, these speed-ups are primarily focused towards NE search using approximate match criterion. We also present experimental results comparing the trade-offs in search speed versus detection accuracy using these techniques.

In the following, we describe three techniques for speeding up the detection process.

6.1. Node clustering

As mentioned in Section 4, our HMM based videotext OCR system uses a trigram character LM while generating character lattices. Consequently, a lattice with trigram LM probabilities is required to have many copies of nodes and arcs than would be necessary if lattice had unigram LM probabilities. The trigram probabilities are useful in computing *accurate* posterior probabilities. Once we have computed character/arc posteriors using the acoustic and language model scores with the forward-backward algorithm, we can collapse multiple copies of arcs and nodes for the purposes of looking up NEs. In doing so, we essentially perform three sequential operations on the lattices: merge nodes, merge arcs, and prune arcs that contain a character which can be ignored during search. The first two operations can be used to speed up the lookup with both exact and approximate match criteria, whereas the third stage is effective for approximate match only. Note that the output of the last stage is a compact lattice with significantly smaller number of arcs and nodes. We referred to this lattice as the node clustered (NC) lattice.

6.1.1 Merging nodes

The first step involves merging nodes with the same spatial position. The algorithm to merge nodes is as follows. Let $\{n_c\}$ be a set of nodes in the lattice with the same position stamp. We then perform the following operations for these set of nodes:

1. Create a new node n_p.
2. For each node n in $\{n_c\}$, let $\{a_i\}$ = set of incoming arcs to node n, and $\{a_o\}$ = set of outgoing arcs from node n.
3. Re-route each $\{a_i\}$ through n_p so that it is an incoming arc of node n_p.
4. Re-route each $\{a_o\}$ through n_p so that it is an outgoing arc of node n_p.
5. Delete all nodes n in $\{n_c\}$.

6.1.2 Merging arcs

The merging of nodes with the same position stamp can create multiple arcs between pairs of nodes with the same word label. In this step, we merge the multiple arcs with the same word label. The algorithm used to merge arcs is as follows:

1. For each pair of connected nodes i and j:
   a. Let $\{a_{ij}\}$ denote the set of arcs between any two connected nodes i and j.
   b. For each set $\{b_{ij}\}$ in $\{a_{ij}\}$ such that, for any two arcs p,q in $\{b_{ij}\}$, word(p) == word(q)
      i. We create a new arc, r, between nodes i and j such that:
         1. word(r) = word(p), p in $\{b_{ij}\}$
         2. posterior(r) = $\sum$ {posterior(p) $\mid p$ in $\{b_{ij}\}$}
      ii. Delete all arcs p in $\{b_{ij}\}$
   c. End-for in Step b
2. End-for in Step 1

6.1.3 Pruning "Do Not Care" arcs

In our description of approximate match criterion in Section 4, we noted that all arcs with ignore characters are treated as Φ (null) arcs. As a result, we can further simplify the lattice resulting from the transformations described in Section 6.1.2 by replacing multiple "ignore" character arcs between pairs of nodes with a single ignore arc using the following algorithm:

1. For each pair of connected nodes i and j
   a. Let $\{a_{ij}\}$ denote the set of arcs between any two connected nodes i and j.
   b. Let $\{b_{ij}\}$ in $\{a_{ij}\}$ be such that, for any arc p in $\{b_{ij}\}$, word(p) in ignore_list.
   c. Find arc q = argmax({posterior(p) $\mid p$ in $\{b_{ij}\}$})
   d. Delete all arcs p in $\{b_{ij}\} \backslash q$
2. End-for in Step 1

6.2. Node caching

We mentioned in Section 4 that we use depth-first lattice traversal for our NE search. In performing depth-first traversal with approximate match criterion, it is possible to traverse a node with the same partial match (of a particular NE) multiple times. However, the partial match at any given node is going to remain the same, irrespective of whether we reached that node in one single hop from a predecessor node or multiple hops which included a sequence of "do not care" characters. Therefore, we maintain a record of the partial match at any given node in the lattice. Before traversing to a new node, we check to see if the partial match thus far has already been cached in the new node. We propagate to the new node only if the node has not been cached. After entering the node, we update the cache for future traversals.

6.3. Lattice pruning

The size of the lattice produced by the videotext OCR decoder directly impacts the lookup speed. Therefore, we also studied the impact of the size of the lattice produced by the decoder on speed and accuracy of NE lookup. The lattice size can be pruned during creation or after computing the posteriors using the forward-backward algorithm.

6.4. Experimental results

In Table 3, we list the %true detects at different false alarms rates and the associated lookup time for NE detection using approximate match criterion with different configuration. All the three stages of node clustering described in Section 6.1 were performed for the systems that used node clustering.

In Table 3, we show the number of recalled NEs obtained by sampling the ROC curve at points with false accepts (FA) equal to 4, 10, and 131, for various speed-up configurations. FA=4 and FA=131 were chosen since they are sample operating points in terms of FAs for the *oracle* NE detection results in Table 1. In general, just performing node clustering slightly degrades recall performance for high-FA regions (-1.7%), improves recall performance in low-FA regions (+4%), and significantly (87%) improves overall detection speed. Its performance (both speed and recall) is sustained irrespective of the size of the lattice used. The possible reason for the improvement in low-FA recall performance could be one or both of the following:

1. Increase in the total number of *unique* lattice paths (due to node clustering).
2. Consolidation of the character (arc) posterior scores across lattice paths which aids in better separation between correctly detected named entities and false ones.

Node caching, when used on its own, results in a very small degradation in recall in both low-FA and higher-FA regions, and results in a small improvement (20%) in overall detection speed. But when we perform node caching along with node clustering, the recall performance drops considerably in low-FA regions (-10%) and high-FA regions (-4.3%). The improvement in overall detection speed when using node clustering and node caching over just using node clustering is very small (17%) when used without pruning. On the whole, the benefits of using node caching are overshadowed by the disadvantages of its use, especially along with node clustering.

Lattice pruning provides a lot of control over the tradeoffs between recall performance and overall detection speed. Aggressively pruning the lattice tends to slightly improve low-FA recall performance (1%), degrade recall performance in higher-FA regions (-4.7%), while improving overall detection speed (50%-87% depending on how aggressively the lattice is pruned). But the real gains from using lattice pruning are seen when it is combined with node clustering – there is an overall improvement in low-FA recall (4%) while maintaining the earlier degradation in higher-FA recall (-4.7%), but providing a significant improvement in overall detection speed (94.4%).

NCl	NCa	Pr	Recall (%)			*f*(%)
			FA=4	FA=10	FA=131	
No	No	Dense	59.6	66.8	76.2	0.0
Yes	No		62.0	65.6	76.2	87.6
No	Yes		58.9	65.4	76.2	20.5
Yes	Yes		53.8	63.9	76.2	89.7
No	No	Sparse	60.1	67.8	72.6	50.8
Yes	No		62.0	67.5	72.6	94.4
Yes	Yes		57.2	65.9	72.6	94.8

Table 3: Table comparing the speed performance and detection performance of different system configurations. NCl: Node clustering, NCa: Node Caching, Pr: Lattice size, FA: false accept, *f*: Speed improvement

6. CONCLUSIONS AND FUTURE WORK

In this paper, we demonstrated significant improvements in recall of named entities by using character lattices instead of 1-best hypotheses. We also compared different objective functions based on character posteriors for rejecting false positives. Recognizing the need for fast detection, we developed several techniques for improving detection speed. In particular, the combination of node clustering and moderate lattice density resulted in best trade-off between speed and accuracy.

The focus on this paper was on detecting pre-defined or *known* entities. Future work will address detecting new or *unknown* entities. Given videotext regions tend to be short, algorithms based on long-term context for detecting new NEs are unlikely to result in high recall. Therefore, we will explore linguistics cues to detect unknown named entities.

7. REFERENCES

[1] T. Sato et al., "Video OCR: Indexing Digital News Libraries by Recognition of Superimposed Caption", *ACM Multimedia Systems Special Issue on Video Libraries*, 7(5): 385-395, 1999.

[2] P. Natarajan, B. Elmieh, R. Schwartz, and J. Makhoul, "Videotext OCR using Hidden Markov Models," *Proceedings Sixth International Conference on Document Analysis and Recognition*, pp. 947 – 951, Seattle, WA, 2001.

[3] F Kubala, R Schwartz, R Stone and R Weishedel *Named Entity Extraction from Speech* Proc. Broadcast News Transcription and Understanding Workshop 1998.

[4] F. Bechet, A. L. Gorin, J. H. Wright, and D. H. Tur, Detecting and extracting named entities from spontaneous speech in a mixed-initiative spoken dialogue context: How May I Help You? *Speech Communication, Volume 42, Issue 2*, February 2004,

[5] P. Natarajan, Z. Lu, I. Bazzi, R. Schwartz, and J. Makhoul, "Multilingual Machine Printed OCR," *International Journal Pattern Recognition and Artificial Intelligence, Special Issue on Hidden Markov Models in Vision*, pp. 43 – 63, 2001.