Learning to Parse Hierarchical Lists and Outlines using Conditional Random Fields

Ming Ye and Paul Viola
Microsoft Corporation, One Microsoft Way, Redmond, WA 98052
{mingye,viola}@microsoft.com

Abstract

Handwritten notes are complex structures which include blocks of text, drawings, and annotations. The main challenge for the newly emerging tablet computer is to provide high-level tools for editing and authoring handwritten documents using a natural interface. One frequent component of natural notes are lists and hierarchical outlines which correspond directly to the bulleted lists and itemized structures in conventional text editing tools. We present a system which automatically recognizes lists and hierarchical outlines in handwritten notes, and then computes the correct structure. This inferred structure provides the foundation for new user interfaces and facilitates the importation of handwritten notes into conventional editing tools.

1. Introduction

Spontaneous on-line ink notes taken on a tablet PC frequently have hierarchical structures. Users typically write out paragraphs which are composed of lines, lines which are composed of words, and words which are composed of strokes. It is also very common that users create hierarchical structures between paragraphs using different indentation and/or bullet schemes. Automatically interpreting these hierarchical structures allows for complex high-level manipulations such as insertion of a line, moving or collapsing sub-trees, and porting ink into text preparation systems like Microsoft Word with appropriate formatting.

This paper focuses on ***outline parsing***, the problem of segmenting a block of text lines into paragraphs and determining the hierarchical structures between the paragraphs. Each paragraph has certain formatting attributes such as its bullet and indentation styles(see Figure 1)[1]. After parsing, the resulting outline tree has a single invisible ROOT node and a number of paragraph nodes, each containing at least one line and possibly a number of children nodes. The presented outline parser assumes that graphical elements have been filtered out, strokes have been grouped correctly into words, lines and blocks, and annotations have been segmented and removed. These pre-processing modules are beyond the scope of this paper and will be described elsewhere.

Two observations make outline parsing much simpler. The first is that the lines within each block are naturally ordered from top to bottom ***and*** that the nodes in the tree have the same depth first order[2]. The second observation implies that the hierarchical structure can be encoded by assigning each line a label. The labels encode both the depth of the node in the tree and whether the line is a continuation of the same paragraph (see Figure 1). Given these two observations, the inference of the outline tree can be achieved as a line classification problem, where each line is classified into one of N depths and as a continuation or non-continuation. From the classification for each line the tree structure can be computed in one pass.

We take a learning approach to the line classification problem, in particular, we adopt the Markov modeling framework introduced by Collins [2]. The Collins model is a powerful generalization of a Hidden Markov Model [8]. Like a hidden Markov model, a Collins model discovers from a training set of examples the key regularities necessary to label additional examples.

In order to understand the necessity of this model, let us review alternative simpler models. The first approach might be to classify each line based on features computed from that line alone. As described in Section 3, a total of 57 features are computed for each line. Examples include: "left indent", "right indent", "left indent relative to the previous line", "is a bullet

[1] This definition includes a list item as a special case.

[2] Depth first order on a tree is the order of encountered nodes during a depth first search.

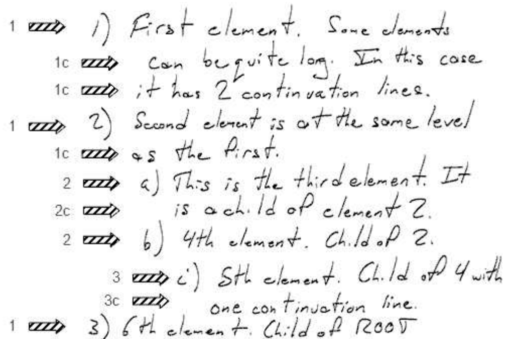

Figure 1. An example outline. Each line is labelled with its depth and if it is a continuation. A line given the label 3c is at depth 3 in the tree and is a continuation of the paragraph. This example has 6 paragraphs (nodes).

present", etc. Using these features one can attempt to learn a function which will correctly classify the line's depth and continuation. This simple scheme, because it is independent from line to line, has a difficult problem in labeling lines because *context* is very important.

One simple extension is called "stacking". In this case the features of the current line ***and the features of the surrounding lines*** are used as input to the classifier for each line (the features are "stacked" into a single input vector). While this improves performance, the dependence between *labels* is not modelled.

The most powerful model is one that ***both*** stacks input features ***and*** propagates label dependencies. Hidden Markov models are potentially appropriate for this process, but one technical assumption is violated, the independence of the observations given the hidden state. Since the input features are stacked, the same feature value appears many times for different input times. This is a serious violation of independence. Recently several types of models have been proposed that do not require observation independence, these include Conditional Random Fields [5], the Collins model, and other non-generative Markov processes [1, 10]. The Collins model is by far the simplest, so we use it for this application.

2. Collins Model for Markov Chains

The formal justifications for the Collins' model and related models are beyond the scope of this paper. Only the details necessary for understanding the operation and training of the model will be summarized.

The model operates as follows: given a sequence of observations s_t we attempt to find a sequence of labels l_t. A Collins model utilizes a set of features $f_i(l', l'', \mathbf{s}, t)$ which are binary functions of a pair of labels, a sequence of observations $\mathbf{s}$, and the time (or position in the sequence). The cost of a label sequence is defined as:

$$C(L, \mathbf{s}) = \sum_t \sum_i \lambda_i f_i(l_t, l_{t-1}, \mathbf{s}, t) \quad (1)$$

where L is a sequence of labels in time $\{l_t\}$, and λ_i are model parameters. Given many labeled training examples $\{L^k, \mathbf{s}^k\}$, the learning process attempts to find a set of weights $\{\lambda_i\}$ such that $\hat{L}^k = \arg\min_L C(L, \mathbf{s}^k)$ is equal to L^k.

Notice that each feature depends only on a pair of adjacent states. As a result terms from the summation may be divided into independent groups. This leads to an efficient minimization using dynamic programming (the algorithm is essentially equivalent to Vitterbi decoding of HMMs).

The features as expressed above are in an abstract form which does not provide much intuition for their operation. Let us consider the following examples in order to build intuition. Without much loss in generality we will consider only binary features which return either 0 or 1. Consider a particular form of feature which ignores the observations and time altogether. These could be rewritten as $f_i(l_t, l_{t-1})$. One particular feature, call it f_{T14}, returns the value 1 if $l_{t-1} = 1$ and $l_t = 4$ and 0 otherwise (i.e. the state has transitioned from a line at depth 1 to depth 4). This transition is impossible due to the nature of the outline tree. To ensure that the model never outputs impossible labels the learning process could assign the corresponding weight λ_{T14} a very large positive value. As a result any hypothetical label sequence including this transition is assigned a high cost. Conversely, f_{T12} (which tests for a transition from depth 1 to 2) is a common occurrence which could be assigned a negative or small positive weight.

Another type of feature can be used to encourage particular states. For example the hypothetical feature $f_{S1_Indent<20}$ returns 1 if the current state is 1 and the left indent of the current line is less than 20 millimeters. This is a fairly common occurrence and it should be assigned a negative weight. The feature $f_{S1_Indent>500}$is an uncommon occurrence and should be assigned a large positive weight.

The most complex type of feature relates two labels given some property of the observation. For example $f_{T11_\mathrm{RelIndent}} < 20$ returns 1 if the current and previous labels are both 1 ***and*** the relative indent between the lines is less than 20 millimeters. This too is a common occurrence and it should be assigned a negative weight.

Of course none of these weights are assigned by hand. Given a large set of features and a large set

of examples, the Collins model is trained iteratively by gradually adjusting the weight vector until convergence.

3. Line Features

The input to the feature extractor is a block of correctly grouped lines which have similar but otherwise arbitrary orientation. The first preprocessing step is to compute the line rotation angles and define the block coordinate. Then we can compensate for the rotation angle and proceed assuming all lines are horizontal and up-right.

There are a set of basic line features from which we derive all other features. We call these features ***raw features***. They include: the left, right, top and bottom line bounds, indent level and bullet type. Calculating the line bounds is straightforward (although care needs to be taken in computing the top and bottom bounds because ink lines are not straight and the ascenders and descenders can be quite irregular). The procedures for indent level estimation and bullet detection are described below.

Indent Level Estimation. Indent levels are quantized left indentations. Although the indent lengths may differ greatly between examples, the indent levels are relatively stable, roughly corresponding to the outline depths (see examples in Figure 4). We use the K-means algorithm for this quantization problem: starting with equal-sized bins, assign the observations to the nearest bin centers, update the bins with the new members and iterate until they no longer change [3]. It is often observed in ink notes that the indent lengths of the same level gradually drift down the page. In such cases, quantizing the absolute indents directly may bury actual levels in spurious detections. To alleviate this problem, we carry out quantization in two passes. The first pass quantizes relative indents and groups neighboring lines which have zero relative indents. The second pass quantizes the average absolute indents of the line groups.

Bullet Detection. Lists are very common structures in ink notes. Bullets signal the start of list items (paragraphs) and their presence can greatly reduce the uncertainty of outline labeling. We have developed a rule-based bullet detector which recognizes a small set of symbols and symbol-embellishment patterns, and exploits consistency between bullets to boost detection confidence. The algorithm comprises four steps. First, for each line we generate several bullet candidates from the stroke clusters at the beginning of the line. Secondly, for each candidate we compute features (such as width, height, aspect ratio, spatial and temporal distances to the rest of the line, etc.), try to recognize it as one of the types such as "dash" or "ending with a parenthesis" (e.g., "1.a)"), and assign it a score in [0,1] indicating the certainty of the candidate being a bullet. Thirdly, a score in [0, 1] is computed for each pair of candidates indicating the degree of similarity between them. The final score of each candidate is a weighted sum of its self-score and all of its pair-scores, reflecting that the more the candidate looks like of a known bullet type *AND* the greater number of other candidates which resemble it, the more likely this candidate is an actual bullet. We then accept a candidate and remove all of its conflicting candidates in a highest-confidence-first fashion, until all candidates have been processed or the highest score falls below a certain value. Preliminary experimental results have shown that this method is effective in detecting common ink bullets such as dashes, dots, alphanumeric-dot combinations and even bullets of unknown types. The features it computes can also be utilized in learning-based bullet recognition which we are currently investigating.

Context	Features	Normalization
($\Delta t = 0$)	Line height	0, 1
	Line width	0, 1
	Left indent	0, 1
	Right indent	0, 1
	Indent level	
	Is the first line in the block	
	Is a bullet present	
	Is the bullet of type X	
	Right indent normalized by the block width	
($\Delta t = 1$)	Relative left indent	0, 1
	Relative right indent	0, 1
	Inter-line distance	0, 1, 2, 3
	Same "is a bullet present" status	
	Same bullet type	
	Relative indent level	
	Is relative indent level positive/zero/negative	
	The pair's line height ratio	
	Relative right indent normalized by the larger line width	
($\Delta t > 1$)	Is the indent level different from its 4 neighbors'	
	Is the line continuation of a list item	
	Ratio between the next and previous inter-line distances	

Table 1. Primitive line features. Normalization schemes: 0 – not normalized, 1 – by average line height, 2 – by the minimum interline distance, 3 – by the median interline distance. All listed schemes for a feature are used.

3.1. Primitive Line Features

Table 1 shows the primitive line features that were used to produce the results reported in this paper. We divide these features into three categories depending on how much context Δt is used in their computation: ($\Delta t = 0$) means only the raw features of line t are used; ($\Delta t = 1$) means the previous or next neighbor's raw features are also used, and so on. Length features can be normalized by various global statistics such as the average line height in the block.

Apparently, there are many meaningful ways of combining raw/derived features and what Table 3 enumerates is merely a small portion. Instead of hand-engineering more features, we take a systematic approach to this problem.

3.2. Combining Primitive Features into Collins Model Features

Recall that the Collins model requires features of the form $f_i(l', l'', \mathbf{s}, t)$ which are dependent both on the current state (or pair of states) and the observation sequence. These features are formed from the primitive features using the training set.

Combination Filters. Based on the initial set of hand constructed filters, a set of *combination* filters are constructed. Each computes a random linear combination of a random subset of the hand constructed filters.

Binary Features. The mean and the variance for each continuous valued filter is estimated from the training set. The range is then portioned into 6 bins each 1 standard deviation in width. A total of 6 binary features are created from each continuous feature. The binary feature takes on the value 1 if the continuous feature falls in the corresponding bin, and zero otherwise.

Observation Features. One feature is generated for each triple $\{s, i, v\}$. The feature returns 1 if the current state is s and binary feature i equals v. Only those features which return 1 for some example in the training set are retained.

Transition Features. One feature is generated for each quadruple $\{s, s', i, v\}$. The feature returns 1 if the current state is s, the previous state is s', and binary feature i equals v. Only those features which return 1 for some example in the training set are retained.

4. Experiments

Our experimental data are a collection of 522 ink files created in Windows Journal® on the TabletPC. All these files contain substantial handwritten script showing interesting outline structures. The median and maximum numbers of lines in a block from this set are 15 and 66 respectively. Each line is labeled with its depth and if it is a continuation: title lines are labeled as 0 or 0c, the rest lines are labeled as 1, 1c, 2, 2c and so on. Five examples are given in Figure 3 and Figure 4.

We partition the data into three sets: 371 for training, 75 for evaluation (observing if accuracy improves with the number of iterations) and 76 for final testing. The following parameters are used in training: learning rate 0.2, decay rate 0.9 and number of iterations 10. The total number of filters used by the Collins model is 6058, which includes 57 raw line features, 228 "stacked" filters, 1135 binarized filters. The remaining filters are equally split between OBSERVATION and TRANSITION. All experiments were carried out on an Intel 3GHz PC with 2GB RAM. Training takes about 28 minutes for 446 examples. Decoding is fast, taking 0.9 seconds for the largest file (66 lines). Note that neither the training nor the decoding program has been optimized for speed, and none of the parameters has been finely tuned.

	Train	Eval.	Test
Number of examples	371	75	76
Avg. %. of misclassified lines	7.1	11.4	10.4
% of files having 0% error	38.4	28.0	34.2
% of files having $\leq$ 20% errors	88.2	80.0	84.2

Table 2. Paragraph segmentation results.

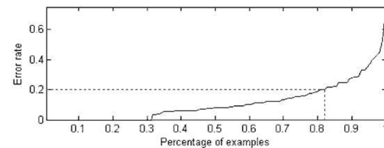

Figure 2. Sorted errors on the evaluation and testing examples. 82% (124 out of 151) examples have 20% or less misclassified lines.

Paragraph Segmentation. We first test out our line labelling system on the paragraph segmentation problem – labelling each line as 1 (paragraph start) or 1c (paragraph continued). Finding paragraphs is a significant problem by itself and paragraph features can be very useful for outline classification. Furthermore, compared to outline inference, paragraph segmentation is a more suitable test bed for our algorithmic framework because there is much less labelling ambiguity (see the Outline Inference section) and we have more

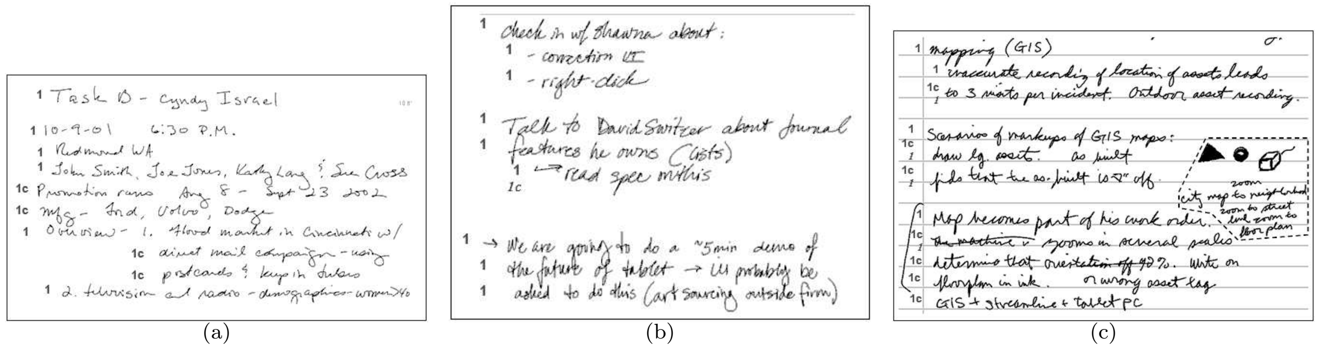

Figure 3. Paragraph segmentation examples. Groundtruth labels are shown in Arial. Predicted labels, if different from the groundtruth, are shown in Italian Times New Roman below the groundtruth.

data relative to the number of classes and hence the results more truthfully represent the algorithm performance.

Our outline classification code can work directly on paragraph segmentation after mapping the groundtruth labels from {0, 0c, 1, 1c, 2, 2c, ...} to {1, 1c}. We measure the error on each example by the percentage of misclassified lines. Three types of error statistics are summarized in Table 2. Figure 2 is a plot of sorted errors on the 151 evaluation and testing examples. The curve remains low and flat except for a sharp rise towards the end, meaning that our algorithm performs fairly well on the majority of notes and errors concentrate on a very small set of cases.

When examining the failure cases, three factors emerged as the major sources of errors. The first is bullet detection error. The only misclassification in Figure 3(b) is due to the arrow bullet not being recognized. We expect to significantly reduce such errors with the introduction of more powerful bullet detection algorithms. The second cause is that currently our system does not consider interaction between the outline structure and the rest of the page. The errors around the middle section in Figure 3(c) are largely due to the removal of the graphical strokes and annotation by the side (in the dashed polygon). One of our future work items is to exploit more features including contextual features. The third cause is ambiguity without full recognition. Our technique mainly relies on geometric features and it can only do as well as one browsing ink notes without carefully reading into the content. For example, Figure 3(a) shows an example with no errors. At a glance, the results look right. However, once we read the sentences, it becomes clear that the 5th and the 6th lines should actually be labeled as 1. Such ambiguous cases are hard to label as well as to infer. Refining features and collecting more training data will help disambiguate some of the cases. Potentially the error patterns can also be incorporated into UI design and user adaptation to improve parsing accuracy and end-to-end inking experience.

Outline Inference. By first running the paragraph segmentation algorithm, some paragraph features can be added to the outline labeling system (Table 1), and then the same training and decoding programs apply. Currently, the only primitive paragraph feature we include is “is this line a paragraph start”. Again, we measure the error on each example by the percentage of misclassified lines and report the error statistics in Table 3.

	Train	Eval.	Test
Number of examples	371	75	76
Avg. %. of misclassified lines	32.7	39.6	48.1
% of files having 0% error	17.5	17.3	9.2
% of files having $\leq$ 20% errors	43.2	34.7	34.2

Table 3. Outline inference results.

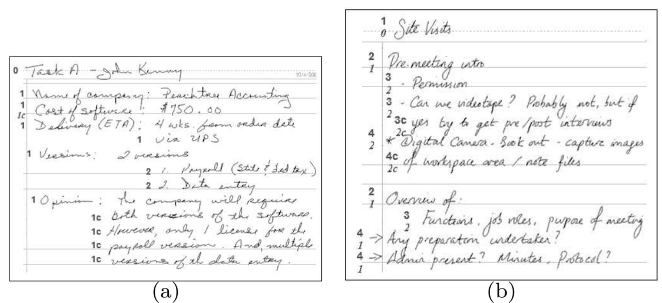

Figure 4. Outline inference examples. See Figure 3 caption.

It is important to point out that there is a considerable amount of ambiguity in outline structures and it makes data labeling, training and performance evalua-

tion much harder than in the paragraph segmentation problem. Our ground truth data is labeled by hand, and is necessarily subjective and includes significant variations. We have frequently observed variation between such alternative decisions for different example in our labeled data. This obscures boundaries between classes and makes training less effective. Figure 4(a) is an example that is almost correct and Figure 4(b) is an example that is almost entirely wrong. However a closer examination reveals that these two files actually have similar structures and similar inference results. The discrepancy in the error rates is largely due to labeling ambiguities: (i) by labeling the title line in Figure 4(b) 1 instead of 0, most results are off by one depth level from the groundtruth; (ii) the 3rd line in Figure 4(a) requires understanding the content to parse right; (iii) the user intention for the 5th and 6th lines is unclear even after careful reading – both the groundtruth and the inferred results seem plausible. When exposed through well designed UIs, many "errors" such as (i) may not even be noticeable by the user. In addition, users' tolerance of errors increases with the amount of ambiguities; errors such as (ii) and (iii) are unlikely to cause much annoyance. The simplistic error metric we used to produce the numbers in Table 3 does not reflect user experience well and should be interpreted with caution.

5. Related Work

Literature on on-line ink layout analysis is scarce and no previous work exists on parsing hierarchical ink notes. Earlier papers in the human computer interaction area are more concerned with user interface design and the developed ink analysis algorithms are usually simple in comparison [7]. Jain et. al. [4] describes an ink notes analysis system which classifies strokes into text and non-text, groups text strokes into lines, and partitions the page into text, diagram and table regions, under some restrictive assumptions such as horizontal baselines. Shilman et. al. describe a parsing component supporting freeform inking which was shipped with Version 1 Microsoft Windows Journal® [9]. It performs writing/drawing classification, word, line and paragraph grouping under arbitrary inking orientations. A seemingly highly related area is document image analysis (DIA) which deciphers the structure in scanned images of printed documents. In particular, the work presented in [6] formulates line/paragraph segmentation as a sequence labeling problem and solves it in an optimization framework. Nonetheless, our optimization framework is drastically different from theirs and in general existing DIA methods cannot be applied to ink parsing because online handwritten data are far more variable and heterogeneous. The work that inspired us to adopt Conditional Random Fields for our problem is [5], which proposes maximum entropy Markov models for text segmentation.

6. Conclusions

Hierarchical outline structure commonly occurs in user notes. Users want a scheme for editing the structure of these outlines, and perhaps for exporting them to word processing programs. We have presented a system which interprets handwritten outline and automatically extracts the correct structure with good reliability.

The described system attempts to label each line in a block of text with its "depth" in the outline tree and flags those lines which are part of the same tree node. A Markov model introduced by Collins is used to classify the lines. This model combines all available line features, such as indentation and bullets, to find a globally consistent assignment of line labels. The parameters of the Collins model are learned from a set of training data. As a result the system is easier to construct *and* extend than a hand engineered system. Computation of the line labels is fast, requiring less than 0.1 seconds on typical ink pages.

We are currently experimenting with more line features and different bullet detection and training methods, which are expected to improve the performance significantly. We are also investigating other applications of Collins' models for ink interpretation. For many of these problems the distribution of ink is inherently two dimensional. Inference and learning on the resulting 2D Markov net is much more difficult. We are investigating schemes for doing this efficiently.

References

[1] Atlun, Y., Hofmann, T. and Johnson, M. (2002). "Discriminative learning for label sequences via boosting", *Proc. Advances in Neural Information Processing Systems (NIPS'15)*.

[2] Collins, Michael (2002). "Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms", *Proc. Empirical Methods in Natural Language Processing*.

[3] Hartigan, J.A. (1975). *Clustering algorithms*. New York: John Wiley & Sons, Inc.

[4] Jain, A.K., Namboodiri, A. M. and Subrahmonia, J. (2001). "Structure in on-line documents", *Proc. International Conference on Document Analysis and Recognition*, pp. 844-848.

[5] Lafferty, J., McCallum, A. and Pereira, F. (2001). "Conditional random fields: probabilistic models for segmenting and labeling sequence data", *Proc. 18th International Conf. on Machine Learning*, pp. 282-289.

[6] Liang, J., Phillips, I.T. and Haralick, R.M. (2001). "An optimization methodology for document structure extraction on Latin character documents", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 23, No. 7, pp. 719-734.

[7] Moran, T. P., Chiu, P., van Melle, W. (1997), "Pen-based interaction techniques for organizing material on an electronic whiteboard", *ACM Symposium on User Interface Software and Technology*, pp. 45-54.

[8] Rabiner, L. (1989) "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. the IEEE*, 77(2), pp. 257-286.

[9] Shilman, M., Wei, Z., Raghupathy, S., Simard, P., and Jones, D. (2003). Discerning structure from freeform handwritten notes. *Proc. International Conference on Document Analysis and Recognition*, pp. 60-65.

[10] Taskar, B., Guestrin, C. and Koller, D. (2003). "Max-margin markov networks", *Proc. Advances in Neural Information Processing Systems*, Editors: Sebastian, T., Saul, L. and Scholkopt, B., MIT Press.