Random Fields Feature Induction for Chinese Characters

draft

November 23, 1999

1. **Chinese Character Random Field**

   Most Chinese characters can be constructed from a few hundred basic components called radicals. Some radicals are simple Chinese characters by themselves; others are parts of a character. Ordinary Chinese dictionaries usually list around 200 radicals for indexing. However more radicals are needed to compose Chinese characters. In this experiment we define 353 radicals, as listed in Appendix A. Figure 1 shows the five radicals that build the character 'bear'.

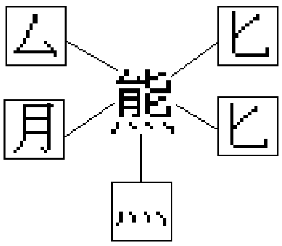

Figure 1: Radicals for 'bear'

   Chinese characters are not randomly piled up from radicals. In stead, there are three compositions to combine two parts:

   (a) Place them side by side, one on the left and the other on the right. We denote this Left-Right composition as LR.
   (b) Place one on top of the other. We denote this Top-Down composition as TD.
   (c) Place one outside of the other, so the outside one surrounds (or partially surrounds) the inside one. We denote this Outside-Inside composition as OI.

   Figure 2 shows how the character 'silk fabric' is composed from radicals in these three ways.

   Formally, we define a Chinese character as a binary tree T. The leaves of the tree are radicals, the internal nodes are LR, TD, or OI. The tree is defined as follows:

$$\begin{aligned} T &::= radical \mid composition\ T\ T \\ radical &::= r_1 \mid r_2 \mid \ldots \mid r_{353} \\ composition &::= LR \mid TD \mid OI \end{aligned}$$

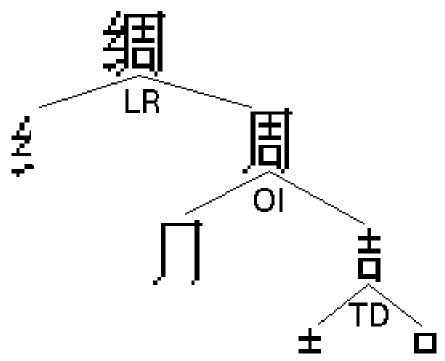

Figure 2: The construction of the character 'silk fabric'

where the r_i's are different radicals. In addition, we insist the following canonical form: the left subtree of a LR, TD, or OI node is the left, top, or outside component, respectively. The majority of common Chinese characters can be represented as such binary trees. However there are some characters that can not easily be decomposed into radicals. In this experiment we simply exclude these characters.

Our Chinese character data set consists of 5496 different Chinese characters, their binary tree representations, and their counts obtained from a large corpus. Appendix B shows a random portion of the data set. The smallest trees have only 1 node, i.e. those characters that by itself are radicals. The largest tree is the character 'insane', which has 19 nodes, as shown in Figure 3. The empirical distribution of tree size is given in table 1.

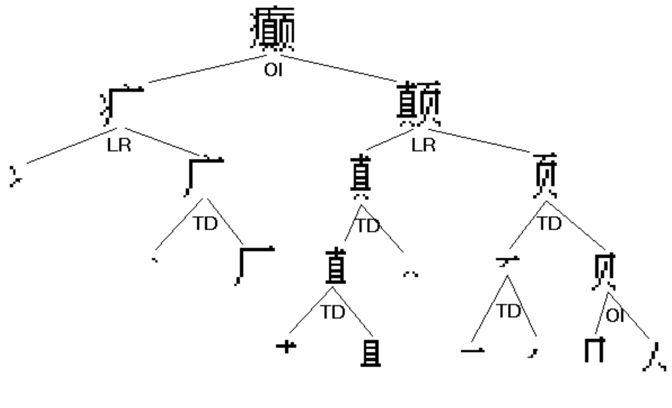

Figure 3: The tree for the character 'insane'

The data set is regarded as empirical samples drawn from an underlying mixture of random fields by the following process: First the size of the tree is determined according to the distribution in Table 1. Then a tree is generated from an underlying Chinese character random field for that size. There are 10 random fields, one for each tree size between 1 and 19. The (Markovian) random field for tree size n is a fully connected graph with $n+1$ vertices $V_1, \ldots, V_{n+1}$. Vertices $V_1, \ldots, V_{(n+1)/2-1}$ take values in *composition* and can be interpreted as internal nodes of a character tree.

nodes	1	3	5	7	9
$P(nodes)$	0.209	0.317	0.272	0.155	0.0381
nodes	11	13	15	17	19
$P(nodes)$	4.67E-3	2.82E-3	1.21E-3	4.60E-4	2.37E-6

Table 1: Tree size distribution

Vertices $V_{(n+1)/2}$, ..., V_n take values in *radical* and can be interpreted as leaf nodes of a character tree. Vertex V_{n+1} is an index of all tree structures reachable from some arbitrary initial structure by tree rotation operation, which is described below. As an example, Figure 4 shows the random field for character trees of size 7. For this field, vertex V_8 can take 5 values, each stands for a different topology of the character trees.

The tree rotation operation is shown in Figure 5. Let y be an internal node, except the root, of a tree. The rotation operation applied on node y changes the structure of the tree as follows: Let x be the parent of y. W.L.G. let's assume y is x's left child (the other case is symmetric). After rotation, x becomes y's right child, and y's original right child tree β becomes x's left child tree. Note in general before rotation x may not be the root node of the tree. In such case after rotation y's parent will be x's original parent (not shown in the Figure 5). The tree rotation operation has the property that by applying an appropriate sequence of such operations, the topology of a tree can be arbitrarily changed.

2. **Feature Induction**

We denote the Chinese Character data set as $\mathcal{D}$, and the empirical distribution as $\tilde{p}$. Our goal is to build a series of fields with the form

$$p^{<t>}(c) \quad = \quad \frac{1}{Z} e^{\sum_i \lambda_i * f_i(c)}, t = 0, 1, 2, \ldots \tag{1}$$

to increasingly approximate $\tilde{p}$. (1) is known as a Maximum Entropy model [1]. The f_i's are $features$, which maps a Chinese character to a number. The λ's are parameters of the field to be trained. Z is a normalization factor to make it a probability distribution. In other words, we want to build an increasingly accurate field $p^{<t>}$ to model the distribution of Chinese character trees. The features we find will reflect the salient structure of Chinese characters. Note by using (1) we implicitly ties features across fields for different tree sizes, so that a feature has the same parameter independent of the size of the tree it appears in.

The series of fields (1) is developed by iterating the following steps [2]:

(a) Given the current random field $p^{<t>}$, search for the best feature f^* within some candidate feature set by investigating the heuristic gain of adding each feature to the model. The best feature f^* is the one that has the largest heuristic gain.

(b) Add f^* to $p^{<t>}$ and train the λ's to generate $p^{<t+1>}$. The λ's are trained to minimize the Kullback-Leibler divergence between the empirical sample distribution $\tilde{p}$ and the random field $p^{<t+1>}$.

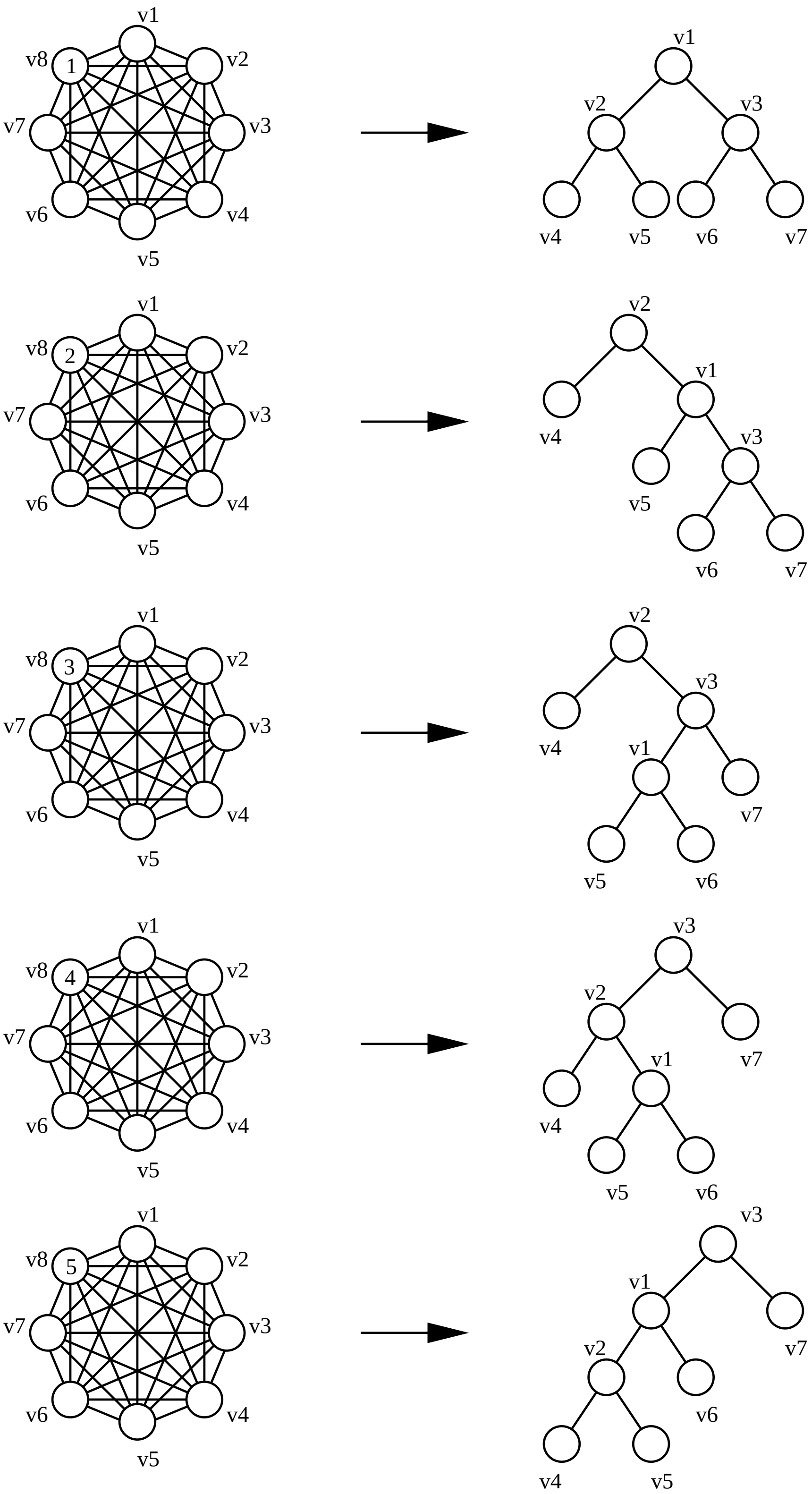

Figure 4: Interpretion of V_8

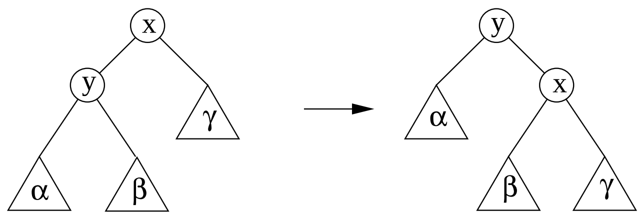

Figure 5: The tree rotation operation on tree node y

This is in fact a greedy search algorithm. The algorithm starts from a random field $p^{<0>}$ with no feature. Since it has no feature, $p^{<0>}$ is actually a uniform distribution on Chinese character trees, with each tree structure, leaf radical, and internal node composition equally probable (however, we take the distribution of *tree size* as given, as in Table 1). With more and more features added, the field $p^{<t>}$ gradually approximates $\tilde{p}$. Since in each iteration the added feature is the one with the largest heuristic gain, the algorithm is greedy.

To implement the feature induction algorithm, we need to solve the following problems:

- We need to specify the form of features, and to specify the candidate feature set for a given field $p^{<t>}$.
- We must be able to evaluate the heuristic gain of a candidate feature f. It is known that the heuristic gain can be expressed as the Kullback-Leibler divergence $D(B_{\tilde{p}}||B_{p^{<t>}})$, where $B_{\tilde{p}}$ and $B_{p^{<t>}}$ are Bernoulli random variables with

$$\begin{array}{ll} B_{\tilde{p}}(1) = E_{\tilde{p}}[f] & B_{\tilde{p}}(0) = 1 - E_{\tilde{p}}[f] \\ B_{p^{<t>}}(1) = E_{p^{<t>}}[f] & B_{p^{<t>}}(0) = 1 - E_{p^{<t>}}[f] \end{array}$$

  Thus the question becomes evaluating $E_{\tilde{p}}[f]$, the expectation of feature f under the empirical distribution $\tilde{p}$, and $E_{p^{<t>}}[f]$, the expectation of f under the current field $p^{<t>}$. $E_{\tilde{p}}[f]$ is straight forward to obtain from the empirical data set $\mathcal{D}$. We will use Markov Chain Monte Carlo (MCMC) method to estimate $E_{p^{<t>}}[f]$.
- We must be able to train the λ's. There are some standard algorithms to train a Maximum Entropy model, e.g. Generalized Iterative Scaling (GIS) [3] and Improved Iterative Scaling (IIS) [2]. In this experiment, we use IIS to do the training. Again, we need to estimate $E_{p^{<t>}}[f]$.

We will present in detail how we define features, and how to estimate feature expectations with MCMC in the following sections.

3. Form of Features

A feature is a function that maps a character tree to a number. All the features in our experiment are binary features, i.e. they take the value 0 or 1. We also refer to the values as 'inactive' or 'active' respectively. Each feature is defined by a unique template. A template is a tree by itself, with each node takes value in *radical* $\cup$ *composition* $\cup$ $\{*\}$. A '*' template node (called a wildcard) matches a character tree node with arbitrary value, other template node matches a character tree node with the same value. A template matches a character tree T iff there exists part of T that has the same topological structure as the template, and all the nodes in this part matches the template's. A feature f is active in a character tree T iff f's template matches T:

$$f(T) = \begin{cases} 1 & \text{if } f\text{'s template matches } T \\ 0 & \text{otherwise} \end{cases}$$

Since there is a 1-to-1 mapping between feature and template, we will no longer distinguish a feature and its template in the rest of the paper.

The allowable features in our experiment have the following forms:

- The atomic features have only 1 node, and the value of the node is either a radical or a composition, as in Figure 6.

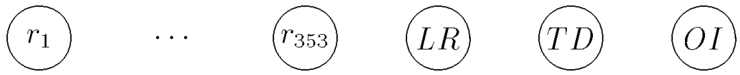

Figure 6: Atomic Features

- Increasingly complex features are 'grown' from existing features. There are several ways to grow a feature f:

  (a) For each wildcard node x in f, we can fill a value v in x to generate a new feature. Note if x has children in f, then v can only take values in $composition$, otherwise v can take values in $composition \cup radical$. We define

  $$GrowFill(f) = \{g | g = f \text{ } with \text{ } one \text{ } wildcard \text{ } node \text{ } filled \text{ } in\}$$

  The number of new features grown in this way, $|GrowFill(f)|$, is given by

  $$\#wi * |composition| + \#wl * (|composition| + |radical|)$$

  where $\#wi$ is the number of internal wildcard nodes in f, $\#wl$ is the number of wildcard leaves in f.

  (b) If the root node r of f is not a wildcard node, we may create a new node p as r's parent, where $p \in composition$. We also create a new wildcard node as r's sibling. We define

  $$GrowParent(f) = \{g | g = f \text{ } with \text{ } parent \text{ } p\}$$

  Figure 7 shows the $GrowParent()$ operation. Note r can be p's left child or right child. This gives us $|GrowParent(f)| = 2 * |composition|$.

  (c) If the root node r of f is not a wildcard node, we may create a new node s as r's sibling, where $s \in composition \cup radical$. We also create a new wildcard node as r's parent. We define

  $$GrowSibling(f) = \{g | g = f \text{ } with \text{ } sibling \text{ } s\}$$

  Figure 8 shows the $GrowSibling()$ operation. Note r can be s's left sibling or right sibling. $|GrowSibling(f)| = 2 * (|composition| + |radical|)$.

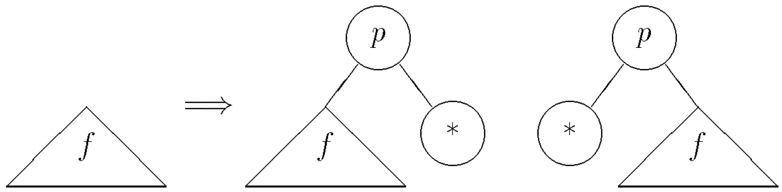

Figure 7: $GrowParent(f)$

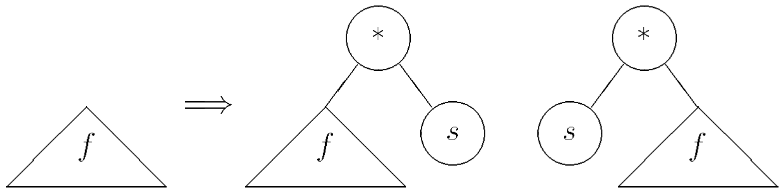

Figure 8: $GrowSibling(f)$

(d) If a leaf node l of f is not a wildcard node, and $l \in composition$, we may create a new node c as one of l's child, where $c \in composition \cup radical$. We also create a new wildcard node as c's other child. We define

$$GrowChild(f) = \{g | g = f \ with \ one \ leaf \ having \ a \ child \ c\}$$

Figure 9 shows the $GrowChild()$ operation. Since c can be l's left or right child, we have

$$|GrowChild(f)| = \#cl * (|composition| + |radical|)$$

where $\#cl$ is the number of leaf nodes in f with composition values.

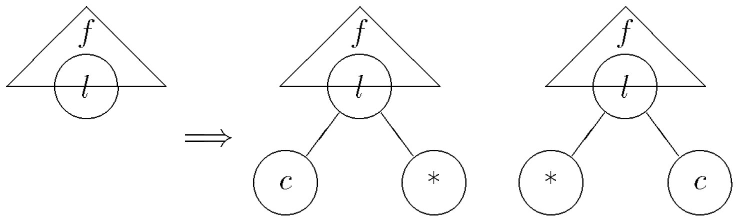

Figure 9: $GrowChild(f)$

We are now ready to define the candidate feature set for a field $p^{<t>}$. Denote the set of features in $p^{<t>}$ as $F^{<t>}$. The candidate features will include all atomic features, and all features that can be grown out of $F^{<t>}$ with a single operation. Formally, the candidate feature set for $p^{<t>}$ is defined as

$$CandidateFeature(p^{<t>}) = \{atomic\ features\} \cup \{g | g \in Grow(f), \forall f \in F^{<t>}\} - F^{<t>}$$

where

$$Grow(f) = GrowFill(f) \cup GrowParent(f) \cup GrowSibling(f) \cup GrowChild(f)$$

4. **Estimating feature expectations with Markov Chain Monte Carlo method**

Given a field $p^{<t>}$ and a feature f, we want to estimate $E_{p^{<t>}}[f]$ for both candidate feature selection and Improved Iterative Scaling training. By definition

$$E_{p^{<t>}}[f] = \sum_{all\ \ c} p^{<t>}(c) * f(c)$$

The expression sums over all character trees c and is infeasible computationally. A standard way to solve the problem is to generate a set of character samples $\mathcal{S}$ from the distribution of the field $p^{<t>}$, and estimate the feature expectation with

$$\hat{E}_{p^{<t>}}[f] = \frac{1}{|\mathcal{S}|} \sum_{c \in \mathcal{S}} f(c)$$

We use Markov Chain Monte Carlo (MCMC) methods to generate samples from $p^{<t>}$ [4] [5]. Since the distribution over tree size is fixed (as in Table 1), we will run multiply sampling chains independently, one for each particular tree size. At the end we will merge samples from all the chains together and weight them properly to maintain the tree size distribution.

As mentioned earlier, a character tree c with n nodes can be viewed equivalently as a particular configuration of a random field with $n + 1$ vertices. That is, the tree c can be regarded as a $n + 1$ dimensional vector $(v_1, \ldots, v_{n+1})$, where $v_1, \ldots, v_{n+1}$ are the values of the vertices $V_1, \ldots, V_{n+1}$ in the random field respectively. In this section we will adopt the view that c is a vector, which is more natural for MCMC.

Formally, the sampling problem for a particular tree size is stated as follows. Given vector size $n + 1$, the state space consists of all the valid vectors $c = (v_1, \ldots, v_{n+1})$, where $v_1, \ldots, v_{(n+1)/2-1} \in composition$, $v_{(n+1)/2}, \ldots, v_n \in radical$, and v_{n+1} is a topology index. We can compute $p^{<t>}(c)$ up to a normalization factor, the unnormalized probability we denote as $p^{*<t>}(c)$. Our goal is to use MCMC to generate a set of samples $\{c_1, c_2, \ldots\}$ according to the distribution $p^{<t>}(c)$.

We define a base transition $\mathcal{B}_k$ of the Markov chain for each dimension $k = 1, \ldots, n+1$, which only updates the kth dimention. One complete step of MCMC walk is achieved by applying $\mathcal{B}_1, \ldots, \mathcal{B}_{n+1}$ in sequence. This in effect defines a Markov chain with transition matrix $\mathcal{T} = \mathcal{B}_1 \cdots \mathcal{B}_{n+1}$. We define all the $\mathcal{B}_k$ such that the distribution $p^{<t>}(c)$ is invariant with respect to any one of them. This guarantees $p^{<t>}(c)$ is also invariant with respect to $\mathcal{T}$. As will be shown below, the base transitions $\mathcal{B}_1, \ldots, \mathcal{B}_n$ have a flavor of the Gibbs sampling algorithm, while $\mathcal{B}_{n+1}$ has a flavor of the Metropolis algorithm. Since both the Gibbs sampler and the Metropolis sampler are a special case of the Metropolis-Hastings algorithm, our solution can be viewed as a kind of Metropolis-Hasting algorithm. The set of samples is generated in this way:

(a) Randomly pick a valid vector c as a starting point.

(b) Generate a new vector c^1 by applying $\mathcal{B}_1$ to c.

(c) Generate another new vector c^2 by applying $\mathcal{B}_2$ to c^1.

(d) Repeat for each of the remaining dimensions, until we generate a new vector c^{n+1} by applying $\mathcal{B}_{n+1}$ to c^n.

(e) c^{n+1} is our first sample. Let $c = c^{n+1}$ and repeat from step 4b to produce subsequent samples.

Now we discuss the base transitions in detail.

- $\mathcal{B}_1, \ldots, \mathcal{B}_{(n+1)/2-1}$

  Let $c = (v_1, \ldots, v_k, \ldots, v_{n+1})$ be a valid vector. Applying a base transition $\mathcal{B}_k$, $k = 1, \ldots, (n+1)/2 - 1$, to c will generate a new vector $c' = (v_1, \ldots, v'_k, \ldots, v_{n+1})$ with the following probabilities:

  $$P(v'_k) = \frac{p^{*<t>}((v_1, \ldots, v'_k, \ldots, v_{n+1}))}{\sum_{v''_k \in \ composition} p^{*<t>}((v_1, \ldots, v''_k, \ldots, v_{n+1}))}$$

  If we view c again as a character tree, then intuitively these base transitions change the value of an internal tree node according to the posterior probability distribution of the node, conditioned on the current values of other nodes and the tree topology. This is indeed a Gibbs sampling step.

- $\mathcal{B}_{(n+1)/2}, \ldots, \mathcal{B}_n$

  Let $c = (v_1, \ldots, v_k, \ldots, v_{n+1})$ be a valid vector. Applying a base transition $\mathcal{B}_k$, $k = (n+1)/2, \ldots, n$, to c will generate a new vector $c' = (v_1, \ldots, v'_k, \ldots, v_{n+1})$ with the following probabilities:

  $$P(v'_k) = \frac{p^{*<t>}((v_1, \ldots, v'_k, \ldots, v_{n+1}))}{\sum_{v''_k \in \ radical} p^{*<t>}((v_1, \ldots, v''_k, \ldots, v_{n+1}))}$$

  The only difference between this group of base transitions and the previous group is the new values are taken in *radical* instead of *composition*. Intuitively this group changes the value of a leaf node, also in a Gibbs sampling step.

- $\mathcal{B}_{n+1}$

  Let $c = (v_1, \ldots, v_{n+1})$ be a valid vector. As defined, v_{n+1} is an index of tree topology. The base transition $\mathcal{B}_{n+1}$ changes v_{n+1} that corresponds to a single tree rotation operation. To understand this base transition, let's look at the specific example of Figure 4 again. In Figure 4, V_8 can take 5 values, which correspond to 5 different tree topologies. Three observations can be made:

  (a) In general, a single rotation operation can not change V_8 to all its possible values. For example, if initially $V_8 = 1$, a rotation operation can only be applied on tree node V_2 or V_3, which will only give us new $V_8 = 2$ or $V_8 = 5$ respectively. In other words, V_8 cannot be changed from 1 to 3 or 4 with one rotation operation.

  (b) Rotation is symmetric. For example, V_8 can change from 1 to 2, and it can change from 2 to 1.

  (c) With a sequence of rotation operations, V_8 can change from any value to any other value in 1,2,3,4,5.

  The operation of base transition $\mathcal{B}_{n+1}$ consists of the following steps:

(a) Let $R(v_{n+1})$ be the set of values that can be reached from v_{n+1} with a single rotation operation. Uniformly choose a value $v'_{n+1} \in R(v_{n+1}) \cup \{v_{n+1}\}$.

(b) Let
$$r = \frac{p^{*<t>}((v_1, \ldots, v'_{n+1}))}{p^{*<t>}((v_1, \ldots, v_{n+1}))}$$

(c) With probability $min(1, r)$ we accept the change, which means the new vector is $c' = (v_1, \ldots, v'_{n+1})$. Otherwise the new vector is the same as the the old one, $c' = c$.

The uniform selection of new value in the first step is in fact the proposal probability in the Metropolis algorithm. Since the rotation operation is symmetric, it is easy to verify that the proposal probability is also symmetric. In addition, since a sequence of rotation operations can arbitrarily change the value of V_{n+1}, $\mathcal{B}_{n+1}$ is ergodic with respect to dimension $n+1$.

Because we use an exponential model for $p^{<t>}$, it is easy to see that $\mathcal{B}_1, \ldots, \mathcal{B}_n$ are all ergodic with respect to their individual dimensions. We have also shown that $\mathcal{B}_{n+1}$ is ergodic with respect to dimension $n+1$. Therefore $\mathcal{T}$, our Markov chain, is also ergodic.

5. Experimental Results

We need to generate samples with MCMC methods in two situations: when we evaluate the candidate features, and during each iteration of IIS model training. We train each model $p^{<t>}$ with 3 iterations of IIS. Therefore for each new feature, we need to generate 4 sets of samples. We generate 100,000 samples as one sample set, with sample size distributed according to Table 1.

In practice we don't estimate the normalization factor Z in (1). This makes performance evaluation hard, since we can't compute the actual likelihood of data. However, a good model will give higher and higher likelihood to the empirical data $\mathcal{D}$ than to some noisy data $\mathcal{U}$ generated from a uniform model (overfitting problem is not considered in this paper). This implies we can use the ratio of unnormalized probabilities between $\mathcal{D}$ and $\mathcal{U}$ as a measure of model goodness. We generated $\mathcal{U}$ with 139,467 samples from $p^{<0>}$, which sample size also distributed according to Table 1. We define a loglikelihood ratio

$$LLR = \frac{1}{|\mathcal{D}|} \log p^{*<t>}(\mathcal{D}) - \frac{1}{|\mathcal{U}|} \log p^{*<t>}(\mathcal{U})$$

which is 0 with $p^{<0>}$, and should increase with better models.

The experiment was run on a Pentium II Linux machine. It took 7 days to generate 150 features. Figure 10 shows that LLR increases with the number of features in the model.

The initial model $p^{<0>}$ is a uniform model. All radicals, compositions, topologies are equally probable, only the distribution on tree size is fixed. Figure 11 are 100 random samples drawn with the MCMC method from $p^{<0>}$. The generated characters violate many rules of Chinese morphology, especially on where to put radicals. They look erratic to native readers.

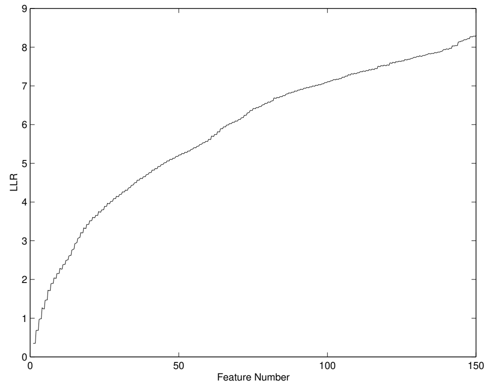

Figure 10: LLR v.s. Number of Features in Model

Figure 11: Characters generated by $p^{<0>}$

The first feature f_1 found is radical $(bfda)$ (see Appendix A), which is known as the character 'Mouth' in Chinese. The shape of this radical is a rectangle box. It is a very common component in many characters. The parameter λ_1 for this feature is approximately 3.02, which means a character with a 'Mouth' radical is $e^{3.02} \approx 20.5$ times more likely than a character without the radical. The field constructed is thus

$$p^{<1>}(c) = \frac{1}{Z^{<1>}} e^{3.02 * f_{Mouth}(c)}$$

The LLR for this field is 0.352, which means on average a sample in the empirical data set $\mathcal{D}$ is approximately $e^{0.352} \approx 1.42$ times more likely than a sample in the uniformly random data set $\mathcal{U}$. Figure 12 are 100 random samples drawn from $p^{<1>}$. Note the increased occurance of 'Mouth', both as individual characters and as component radicals. But other than that, the samples are as chaotic as those drawn from $p^{<0>}$.

Figure 12: Characters generated by $p^{<1>}$

The second feature f_2 is radical $(d2bb)$, which is a horizontal stroke, known as the number 'One'. It is also a very common radical. The parameter for it is $\lambda_2 \approx 3.00$. Meanwhile λ_1 changes slightly to 3.04. The field becomes

$$p^{<2>}(c) = \frac{1}{Z^{<2>}} e^{3.04 * f_{Mouth}(c) + 3.00 * f_{One}(c)}$$

The LLR for $p^{<2>}$ is 0.687.

The third feature found is $f_3 = (d8af)$, which itself is not a character, but a top-right to bottom-left stroke called 'Pie'. The radical is also fairly common. The parameters are now $\lambda_1 = 3.09, \lambda_2 = 3.04, \lambda_3 = 2.91$. The LLR increases to 0.983.

	Feature	λ
f_1	(bfda)	3.25
f_2	(d2bb)	3.02
f_3	(d8af)	2.18
f_4	(* d8af c8d5)	7.08
f_5	(d8bc)	2.24
f_6	(* d9e8 d8bc)	6.96
f_7	(c8cb)	2.37
f_8	(* d8af c4be)	6.43
f_9	(* d8e7 c8cb)	6.19
f_{10}	(* a2b2 d2bb)	5.64

Table 2: The 10th field $p^{<10>}$

The fourth feature is our first composite feature: $f_4 = (* \ d8af \ c8d5)$. It is active when there are a radical 'Pie' and a radical 'Sun' occuring in any of the three compositions. The reason for this feature to emerge is the character 'De', which is a function word and is the most frequent character in Chinese. In fact, 'De' accounts for 4% of all the character occurances. The character tree for 'De' is $(LR\ (TD\ 'Pie'\ 'Sun'\ )(OI\ 'Spoon-outline'\ 'Dot'\ ))$. It is the left subtree of 'De' that makes our program think f_4 is an important feature. The parameters are now $\lambda_1 = 3.13, \lambda_2 = 3.07, \lambda_3 = 2.53, \lambda_4 = 6.48$. Note λ_4 is very high, since in $p^{<3>}$ the chance of co-occurance of 'Pie' and 'Sun' is very low compared to that in the empirical data. Also notice λ_3 decreases, a phenomenon we will discuss later. The LLR now increases further to 1.238.

Table 2 shows the features and their parameters in $p^{<10>}$, the 10th field. For readers familiar with Chinese, these features are all common parts of characters. The LLR for $p^{<10>}$ is 2.268. Figure 13 are 100 random samples drawn from $p^{<10>}$. The samples still do not resemble Chinese characters.

With 100 features in the model, $p^{<100>}$ has an LLR of 7.108. Figure 14 are 100 random samples drawn from $p^{<100>}$. Compared with previous samples, they look much more like true Chinese characters.

An interesting phenomenon is observed during the experiment. In Chinese there is a commonly used radical ($d8e9$), which means 'People' (several appear in Figure 14). This radical appears only to the left of other radicals. Our program discovered $f_{11} = (d8e9)$ for $p^{<11>}$ and at that time, gave it a large parameter $\lambda_{11} = 2.36$. In the subsequent fields, the parameter slightly increases. However, for the 20th field, our program decided that $f_{20} = (LR\ d8e9\ *)$ is a good feature and added it in. Due to the restrictive composition form of the radical 'People', f_{11} and f_{20} is completely correlated in the empirical data such that when one is active, so is the other. Moreover, f_{20} is a more detailed version of f_{11}. As a result, λ_{11} and λ_{20} begin to diverge: λ_{11} stops increasing and drops towards $-\infty$, while λ_{20} increases towards $+\infty$, as shown in Figure 15.

The divergence can be understood by considering artificial samples in which 'People'

Figure 13: Characters generated by $p^{<10>}$

Figure 14: Characters generated by $p^{<100>}$

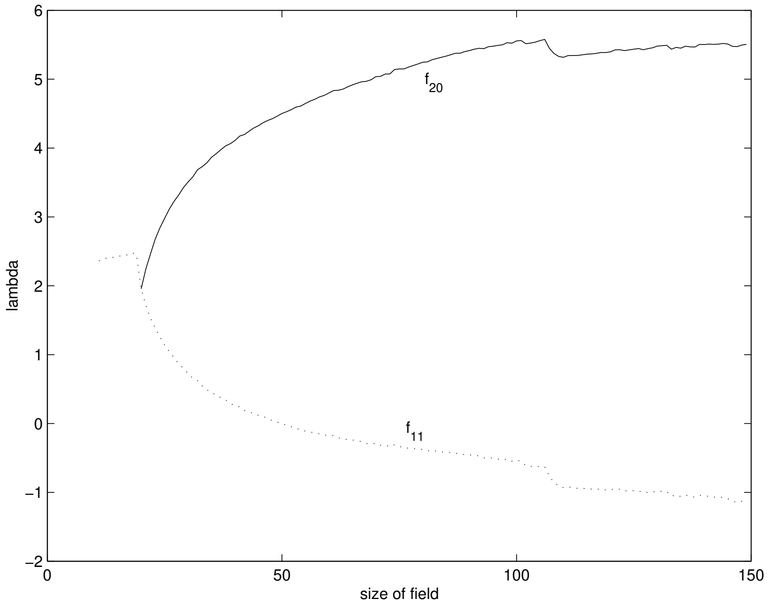

Figure 15: λ diverges when f_{20} is a detailed version of f_{11}

appears alone, or on the right, top, or bottom of some other radicals. f_{11} will be active in these artificial samples, but not f_{20}. Since in the empirical data none of these artificial samples appear, and our program tries to build a field that resembles the empirical data, it will decrease λ_{11} as much as possible to prevent these artifical samples from being generated. In the limit it will drive λ_{11} to $-\infty$. Meanwhile it will try to maintain $\lambda_{11} + \lambda_{20}$ at some positive constant, so that a correct amount of samples containing $(LR\ d8e9\ *)$ can be generated. Therefore λ_{11} will goes to $+\infty$. This divergence in practice is not a big problem, since the program will not run forever. However, it does show the weakness of exponential models of the form (1) in describing zero probability events.

6. **Future Work**

   Add features in BATCH [6]. Forthcoming...

7. Appendix A: The Radicals

These radical are hand picked by the authors, and are not necessarily the correct or best radicals for Chinese.

赤	前	函	坚	局	寿	肖	译	应	足	共	质	钹
a2b1	a2b2	a2b4	a2b5	a2b6	a2b7	a2b8	a2b9	a2ba	a2bb	a2be	a2bf	a2c0
茶	肆	餐	弟	寒	尽	尝	缘	哉	带	登	亘	断
a2c1	a2c3	a2c4	a2c5	a2ce	a2d2	a2d3	a2d5	a2d6	a2d7	a2d8	a2da	a2dc
金	敢	骨	官	水	敝	曹	差	辰	虎	即	祭	既
a2dd	a2de	a2df	a2e0	a2e1	a2e2	a2e3	a2e4	a2e5	a2e6	a2e7	a2e8	a2e9
答	今	具	离	卢	录	尧	疑	以	雍	啬	老	希
a2ea	a2eb	a2ec	a2ed	a2ef	a2f0	a2f2	a2f3	a2f4	a2f5	a2f6	a3c1	a3c7
罩	杨	厄	切	兮	布	存	尔	叫	礼	决	至	钺
a3c8	a3ca	a3cc	a3cd	a3ce	a3cf	a3d0	a3d1	a3d3	a3d4	a3d6	a3d9	a3da
拖	责	径	练	服	判	岛	眈	泰	流	印	师	仰
a3e2	a3e4	a3e5	a3e6	a3e7	a3e8	a3e9	a3ea	a3ec	a3f1	a3f4	a3f5	a3f9
采	凹	八	半	卜	不	才	册	长	厂	车	虫	丑
a3fa	b0bc	b0cb	b0eb	b2b7	b2bb	b2c5	b2e1	b3a4	b3a7	b3b5	b3e6	b3f3
川	串	垂	寸	大	丹	单	刀	典	电	刁	丁	东
b4a8	b4ae	b4b9	b4e7	b4f3	b5a4	b5a5	b5b6	b5e4	b5e7	b5f3	b6a1	b6ab
儿	耳	二	发	非	丰	夫	弗	甫	干	甘	戈	革
b6f9	b6fa	b6fe	b7a2	b7c7	b7e1	b7f2	b8a5	b8a6	b8c9	b8ca	b8ea	b8ef
更	工	弓	瓜	龟	鬼	果	亥	黑	乎	惠	火	击
b8fc	b9a4	b9ad	b9cf	b9ea	b9ed	b9fb	baa5	bada	baf5	bbdd	bbf0	bbf7
及	几	己	夹	甲	兼	柬	巾	斤	井	久	九	臼
bcb0	bcb8	bcba	bcd0	bcd7	bce6	bced	bded	bdef	beae	bec3	bec5	beca
口	来	乐	里	吏	隶	力	两	了	龙	卵	马	毛
bfda	c0b4	c0d6	c0ef	c0f4	c1a5	c1a6	c1bd	c1cb	c1fa	c2d1	c2ed	c3ab
矛	卯	门	米	免	面	民	皿	末	母	木	目	乃
c3ac	c3ae	c3c5	c3d7	c3e2	c3e6	c3f1	c3f3	c4a9	c4b8	c4be	c4bf	c4cb
内	牛	农	女	皮	片	平	妻	七	其	气	千	且
c4da	c5a3	c5a9	c5ae	c6a4	c6ac	c6bd	c6de	c6df	c6e4	c6f8	c7a7	c7d2

丘	求	去	犬	壬	人	刃	日	入	三	山	善	上
c7f0	c7f3	c8a5	c8ae	c8c9	c8cb	c8d0	c8d5	c8eb	c8fd	c9bd	c9c6	c9cf
申	身	甚	生	失	尸	十	石	史	矢	士	世	氏
c9ea	c9ed	c9f5	c9fa	caa7	caac	caae	caaf	cab7	cab8	cabf	cac0	cacf
手	鼠	术	束	衰	水	丝	四	巳	肃	田	头	土
cad6	caf3	caf5	caf8	cba5	cbae	cbbf	cbc4	cbc8	cbe0	ccef	cdb7	cdc1
屯	瓦	丸	万	王	亡	韦	为	未	文	我	乌	无
cdcd	cddf	cde8	cdf2	cdf5	cdf6	cea4	ceaa	ceb4	cec4	ced2	ceda	cede
五	午	戊	西	夕	习	下	襄	乡	小	心	熏	丫
cee5	cee7	ceec	cef7	cfa6	cfb0	cfc2	cfe5	cfe7	d0a1	d0c4	d1ac	d1be
牙	央	羊	也	业	曳	一	夷	乙	尹	庸	永	用
d1c0	d1eb	d1f2	d2b2	d2b5	d2b7	d2bb	d2c4	d2d2	d2fc	d3b9	d3c0	d3c3
尤	由	又	于	雨	与	禹	玉	曰	月	丈	争	之
d3c8	d3c9	d3d6	d3da	d3ea	d3eb	d3ed	d3f1	d4bb	d4c2	d5c9	d5f9	d6ae
止	制	中	重	舟	帚	朱	爪	专	子	奏	丌	兀
d6b9	d6c6	d6d0	d6d8	d6db	d6e3	d6ec	d7a6	d7a8	d7d3	d7e0	d8a2	d8a3
丐	廿	丨	禺	丿	匕	乇	丶	乜	匚	刂	冂	亻
d8a4	d8a5	d8ad	d8ae	d8af	d8b0	d8b1	d8bc	d8bf	d8ce	d8d6	d8e7	d8e9
勹	亠	冫	冖	讠	卩	阝	叟	廴	凵	厶	艹	堇
d9e8	d9ef	d9fb	daa2	daa5	dae0	dae2	dbc5	dbc8	dbc9	dbcc	dcb3	ddc0
瞢	廾	尢	扌	弋	囗	彳	彡	犭	舛	夂	饣	忄
deab	dec3	decc	ded0	dfae	e0ed	e1dc	e1ea	e1eb	e2b6	e2ba	e2bb	e2e0
丬	爿	氵	宀	辶	彐	屮	孑	孓	纟	幺	巛	戋
e3dc	e3dd	e3df	e5b2	e5c1	e5e6	e5f8	e6dd	e6de	e6f9	e7db	e7dd	eaa7
攴	灬	礻	聿	肀	钅	衤	耒	虍	缶	臾	羲	艮
ebb6	ece1	ecea	edb1	edb2	eec4	f1c2	f1e7	f2ae	f3be	f4a7	f4cb	f4de
豕	豸											
f5b9	f5f4											

8. **Appendix B: The Chinese character data set**

There are 225,430,000 samples in the data set, with a total of 5496 different Chinese characters. Each unique Chinese character has a code, a tree presentation, and a count. Most of the codes are Chinese GB-code, with some additional codes for radicals. The tree presentations are manually constructed with the help of a Chinese character stroke sequence database. The counts are obtained from a large corpus. The following is a randomly selected portion of the data set.

count	code	tree
...	...	...
71128	c8d4	(LR d8e9 c4cb)
499885	c8d5	leaf
721	c8d6	(OI b8ea caae)
391	c8d7	(TD dcb3 b6fa)
1190	c8d8	(TD dcb3 (TD e5b2 (TD (TD b0cb c8cb) bfda)))
50493	c8d9	(TD (TD dcb3 daa2) c4be)
2524	c8db	(LR bbf0 (TD e5b2 (TD (TD b0cb c8cb) bfda)))
8320	c8dc	(LR e3df (TD e5b2 (TD (TD b0cb c8cb) bfda)))
123186	c8dd	(TD e5b2 (TD (TD b0cb c8cb) bfda))
5432	c8de	(LR e6f9 (OI b8ea caae))
1254	c8df	(TD daa2 bcb8)
548	c8e0	(LR ded0 (TD c3ac c4be))
4911	c8e1	(TD c3ac c4be)
20208	c8e2	(OI c4da c8cb)
744	c8e3	(TD dcb3 (LR c5ae bfda))
344228	c8e7	(LR c5ae bfda)
4630	c8e8	(TD (OI b3a7 (TD b6fe a2e5)) b4e7)
1808	c8ea	(LR e3df c5ae)
361960	c8eb	leaf
613	c8ec	(LR f1c2 (TD (OI b3a7 (TD b6fe a2e5)) b4e7))
1035	c8ee	(LR dae2 (TD b6fe b6f9))
...	...	...

References

[1] A. Berger, V. Della Pietra, S. Della Pietra. *A maximum entropy approach to natural language processing.* Computational Linguistics, 22, No. 1, 1996

[2] S. Della Pietra, V. Della Pietra, J. Lafferty. *Inducing Features of Random Fields.* IEEE PAMI, Vol. 19, No. 4, April 1997.

[3] J. Darroch, D. Ratcliff. *Generalized iterative scaling for log-linear models.* Ann. Math. Statist. 43, 1470-1480, 1972

[4] A. Gelman, J.Carlin, H. Stern, D. Rubin. *Bayesian Data Analysis.* Chapman & Hall, London, 1995

[5] R. Neal. *Probabilistic Inference Using Markov Chain Monte Carlo Methods.* Tech. Rpt. CRG-TR-93-1, Dept. of Computer Science, University of Toronto, 1993

[6] D. Beeferman, A. Berger, J. Lafferty. *Statistical Models for Text Segmentation.* Machine Learning, special issue on Natural Language Learning, C. Cardie and R. Mooney eds., 34(1-3), pp. 177-210, 1999