A Bayesian Framework for Learning Regular Expressions

P. Thomas Barthelemy (bartho@stanford.edu)
Department of Computer Science, 353 Serra Mall
Stanford, CA 94305 USA

Nicholas Borg (nickborg@stanford.edu)
Symbolic Systems Program, Margaret Jacks Hall
Stanford, CA 94305 USA

Abstract

Regular expressions are used to define a set of strings that match a certain pattern. In the following experiment, we posit that the means by which humans judge similarity of words can be represented by a method of inference using regular expressions. In short, there is a tradeoff of generality and specificity that allows for a selection of an optimal regular expression to categorize a list of input strings.

Keywords: Model inference, regular language, regular expression, deterministic finite automaton.

Overview and Motivation

Understanding the human ability to find structure in data is integral to both cognitive psychology and artificial intelligence. In recent years, a number of studies have shown that human inference across a number of tasks may be Bayesian at some level of analysis. Inspired by the treatment of the number game in Tenenbaum (1999), we propose a regular expression game as follows: Given a set S of input strings drawn from a language $\mathcal{L}$, what is the likelihood that a new string s is in $\mathcal{L}$. Playing such a game requires inferring the rules that govern $\mathcal{L}$ from S. Noting that humans perform this task with success on a daily basis in writing regular expressions to process string data, we considered the hypothesis that human inference of regular expressions is at some level Bayesian.

In doing so, we first generated a Bayesian model that, given a set of input strings S, generated a regular expression that describes S with the highest posterior probability. This model is discussed in the first part of this paper. We then ran a behavioral experiment in which subjects were asked whether or not a given string s fits with a set of strings S. We compare the result of that experiment with results generated given an extension of our model.

Description of the Model

Our model has two parts: We generate a set of regular expressions which describe a set of input stings S, and we find the regular expression in that set with the highest posterior probability. The posterior probability is proportional to the product of the hypothesis' prior probability and its likelihood.

Below, we define the prior probability such that our model biases smaller hypotheses, and the likelihood of generating a given hypothesis is inversely proportional to its cost. Our model aims to balance generality with specificity. For example, given `abc, abd`, we wish to favor the regex `ab(c|d)` over `A*` where `A` represents any letter,[1] which is less likely given the size principle. At the same time, given `aba`, `abc`, `abd`, `abe`, `abf`, we may wish to favor `abA` over `ab(a|c|d|e|f)`.

Representation of Hypotheses

Regular expressions are equivalent to deterministic finite automata (DFA). Because DFA are often easier to reason with, we use them to represent regular expressions in our model (See Figure 1 for an example DFA.)

Further, we define the probability with which a DFA would stop generating a string at a start state . For instance, given the regular expression `a*`, we would need to define the probability of generating strings `a`, `aa`, `aaa`, etc.

Our generative version of the DFA is not to be confused with a probabilistic atumaton, which has transition probabilities much as described above and a final acceptance threshold. Our generative automata does not have this threshold, and instead has an added parameter which controls the likelihood of stopping to generate a string.

As our alphabet, we use the set of all lowercase letters. This set, we feel, has an intuitive boundary, as subjects seeing strings of all lowercase letters are unlikely to suppose an arbitrary uppercase letter or special character, and thus we assume the expressiveness of wildcards to be fixed.

Generating Hypotheses

The hypothesis space is initiated with the base hypothesis, the term applied to a DFA that accepts strictly the input strings. We begin generating a series of consecutive states, where the transition between the first state and the second state is governed by the first letter of the first string, the transition between the second and the third is governend by the the second letter, and so on. For each additional string, if the first i characters overlap with those of any of the first i characters of one previous string but differ in the $i+1$th, we add no new states to the DFA until the transition between the i-1th state and the ith state. We then generate new states for the remaining characters of the string. See Figure 1 for an example of the first hypothesis generated from a set $S = \{$`ababb`, `abbab`$\}$.

Next, hypotheses are added to the hypothesis space through two processes of generalization: merging and applying wildcards.

[1] See the Appendix for a brief summary of our regex notation.

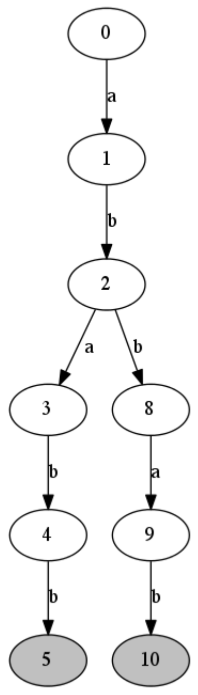
Figure 1: An example base hypothesis DFA generated from the strings ‘ababb’ and ‘abbab’. Note that the two strings share the first two states, but that the fifth and tenth are distinct.

Merging Merging is the process of combining states of a DFA. The newly create state has all incoming and outgoing transitions of its predecessors. Thus, the DFA will continue to accept the same set of strings as the original DFA. However, the DFA will often accept more strings after merging, thus merging states results in a more generalized DFA.

If merging is continued, the DFA will eventually have only one state with all transitions leading to itself. Such a state accepts all strings containing only zero or more of the original characters. A series of merges are illustrated in Figure 2.

Wildcards Wildcards generalize the DFA by extending state transitions to apply to a class of characters. Figure 3 illustrates an example of this.

Enforcing Determinism Merging states and replacing transitions with wildcards can allow one state to have non-unique outbound transitions. However, this turns our representation into a nondeterministic finite automaton (NFA) instead of a DFA. Both are equivalent, but the latter afford easier manipulation for our experiment. Thus, to convert the NFA back to a DFA, we use the subset construction, the effect of which is illustrated in both Figure 2 and Figure 3.

Searching the Hypothesis Space

The number of hypotheses grows quickly with the number of states in the base hypothesis. As a lower bound, the number

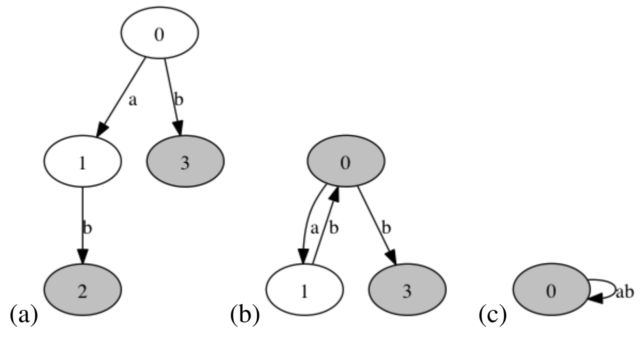
Figure 2: An example of state merging. The DFA starts at state 0 and accepts only at the shaded states. The base hypothesis accepts “b” and “ab” (a). After merging states 0 and 2, the next DFA accepts an infinite set of strings (b). Finally, the merge of states 0 and 1 forces the merge of state 0 and 3 to maintain one unique outbound transition for any state and character (c).

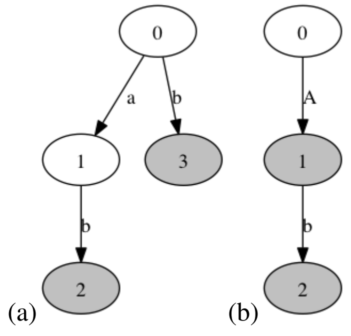
Figure 3: An example of inserting a wildcard, “A”, which represents any character. Replacing a wildcard for one of the outbound transitions of state 0 has forced a merge of states 1 and 3.

of hypotheses is at least as large as the number of possible starting nodes. Because the starting node could be merged with any subset of the remaining $n-1$ nodes in the DFA, there are $2^{n-1} = \Omega(2^n)$ possible start states. Thus, the hypothesis space is at least exponential in the number of starting nodes in the regular expression.

Similarly, the number of possible regex variants created by simply replacing transitions with wildcards is given by 2^m where m is the number of transitions. Thus, simply modifying expressions to include wildcards is also exponential.

An upper bound for the complexity of model merging is much more extreme. Performing subset construction for an NFA with n states can produce, in the worst case, a DFA with 2^n states. Assuming that any subset of these states also produces a valid DFA, we can have at most 2^{2^n-1} DFA.

We resolve this issue by performing beam search. In beam first, we perform breadth first search, but for any given depth of the search tree, we only consider children of the m most promising hypotheses. In our case, the most promising hypotheses were defined as those with the highest posterior probability. (For a more detailed discussion of Beam Search, see (I. Hwang & Goodman, 2011)

Inference with DFA

The posterior probability for a DFA as a hypothesis is given by the following.

$$p(h|S) \propto p(S|h)p(h)$$

Thus, inferring the best DFA given a set of input strings was essentially a problem in balancing simplicity with conservation of belief. That is, DFA were biased a priori for being simpler, but simpler DFA tended to accept more strings.

Prior Probability

We assigned the prior probability to the hypotheses (i.e. regexes) based on the size of the DFA and the number of transitions. In the following equation, $|S|$ is the number of states in the DFA and $|\delta|$ is the number of tranisitons in the DFA, where a wildcard counts as only one transition.

$$P(h) \propto e^{-\alpha|S|-\beta|\delta|}$$

Thus, we have biased smaller DFA that use wildcards instead of literal characters. [cite Program Merging model]

Likelihood

To define the likelihood of producing certain strings, we turn a DFA into a generative automaton with a probability distribution over generated strings. Letting the input strings be independent, the probability of generating the set of strings S is equal to the product of the probabilities of generating each string.

$$p(S) = \prod_{s \in S} p(s)$$

Let the string s be the concatenation of characters $\{a_0, a_1, ..., a_{n-1}\}$. We define the probability of generating a string as the product of the probability of generating the letters in sequence $p(a_0a_1...a_{n-1})$ and the product of stopping the string generation on an accept state $p(\text{stop})$.

$$p(s) = p(a_1a_2...a_n)p(\text{stop})$$

We define the probability of generating the characters as the product of the probabilities of making each state transition.

$$p(a_1a_2...a_n) = \prod_{i=0}^{n-1} p(\delta(r_i, a_i))$$

where r_i is the state reached after the first i characters, and r_0 is the start state.

To simplify inference, we assume uniform probability over all possible outbound transitions for a given state. Formally, the probability of making a state transition is given by:

$$p(\delta(r_i, a_{i+1})) = \frac{1}{\sum_{a \in A} 1\{\delta(r_i, a) \in Q\}}$$

where $1\{\cdot\}$ is the indicator function, which evaluates to 1 if the enclosed proposition is true and 0 otherwise. $\delta(r_i, a) \in Q$ indicates that the state transition leads to another state of the DFA, so the denominator counts the number of characters for which a transition is defined. In the case of the wildcard `A`, the transition is defined for all 26 letters, so the denominator is 26.

For the cases in which a string may alternatively be accepted or continue, one must assign a probability for accepting or continuing the string. For instance, consider the DFA in part (c) of Figure 2. The DFA may generate the empty string, `E`, or it may generate any arbitrary string with `a` and `b`, like `ababbba`. We define the probability of stopping at a given string to be a fixed value τ. This makes shorter strings more likely to be generated.

Instead of leaving τ as a parameter of the model, we assume a uniform prior over the parameter and marginalize it when calculating the likelihood of a string. So, let b be the number of times the DFA skipped an accept state before accepting. That is, b is given by the equation below, where F is the set of accept states.

$$b = \sum_{i=1}^{n-1} 1\{r_i \in F\}$$

The probability of stopping at an accept state is given by the equation below. The condition $r_n \in F \wedge \exists a : \delta(r_n, a) \in Q$ states that the last state reached by the string is an accept state, and that that state has at least one outbound transition.

$$p(\text{stop}) = \begin{cases} \tau(1-\tau)^b & r_n \in F \wedge \exists a : \delta(r_n, a) \in Q \\ (1-\tau)^b & r_n \in F \wedge \nexists a : \delta(r_n, a) \in Q \\ 0 & \text{otherwise} \end{cases}$$

Marginalizing over all possible values of τ, we have:

$$\int_0^1 \tau(1-\tau)^b = \frac{1}{b^2+3b+1}$$
$$\int_0^1 (1-\tau)^b = \frac{1}{b+1}$$

Our equation for $p(\text{stop})$ is thus:

$$p(\text{stop}) = \begin{cases} \frac{1}{b^2+3b+1} & r_n \in F \wedge \exists a : \delta(r_n, a) \\ \frac{1}{b+1} & r_n \in F \wedge \nexists a : \delta(r_n, a) \\ 0 & \text{otherwise} \end{cases}$$

Experiment Design and Model Fitting

Having generated the above Bayesian model for selection of the regular expression with the highest posterior probability, we then conducted a behavioral experiment. 41 subjects (Median age 34, std-dev 13.3) were surveyed on Amazon Mechanical Turk. Two groups (one of 20, one of 21) were presented with 10 yes or no questions in which they were asked to determine whether a given string 'fit' with a list of strings. A sample question was as follows:

Given the list:
yyyt
yt
yyyt
yyyt
yt
Does **yyt** fit?

The experimental data served two purposes. Using the responses to four out of 23 questions, we set the parameters for alpha and beta. We then used the remaining questions to test our model's performance in a regex game as shown below.

Parameter Fitting

Because the role of β had the effect of biasing models with fewer transitions, it also had a tendency to bias regular expressions with fewer states. For instance, the β term would prefer the regular expression `a*` over `aaaa`. Thus, this term was fitted before fitting the α term.

To fit the β term, we observed experiments that would indicate the transition point between a preference for generality over specificity for wildcard-based experiments (e.g. the first two experiments outlined in Table 1). Utilizing the fact that increasing the ratio of repeated strings to new strings decreases the likelihood of a wildcard, and decreasing the length of the sample strings would increases the likelihood of a wildcard, the first two experiments suggest a decision point. That is, if the DFA model is a valid representation of the subjects' responses, then the β term should be between 3 and 5. For further experimentation, we fixed it at 4.

Fitting the α term was less clear. In particular, the desired behavior was easily observed with $\alpha = 0, \beta = 4$. However, a term of $\alpha = 1$ was used to bias hypotheses with fewer states given the same number of transitions.

Table 1: Parameter fitting experiments

Input	Test	Subject Accept	Model Accept
efg, hfg, ifg	ffg	45%	$\beta > 3$
glts, elts, glts, flts	ilts	70%	$\beta > 5$
ab, abb	abbb	90%	$\alpha = 0, \beta = 4$
ao, aoo, aooo	aoooo	80%	$\alpha = 0, \beta = 4$

The Regex Game

Given the fitted model, we then considered the following task:

Given a language $\mathcal{L}$, a set of regular expressions $\mathcal{H}$ (where these are candidates for $\mathcal{L}$), a set of strings drawn S from $\mathcal{L}$, and a string s, we define

$$p(s \in \mathcal{L} | S \subseteq \mathcal{L}) = \sum_{h \in \mathcal{H}} 1\{s \in h\} p(h|S)$$

For human participants, because the hypothesis is unobserved, we take $p(s \in \mathcal{L} | S \subseteq \mathcal{L})$ to be the average acceptance of a test string given a set S.

We define $\mathcal{H}$ to be the set of all hypotheses generated in our model. Note, however, that this is the set of regular languages which were identified in beam search, so it does not represent the set of all possible hypotheses.

Results and Discussion

Figure 4 shows the comparison between our model and the human subject data.

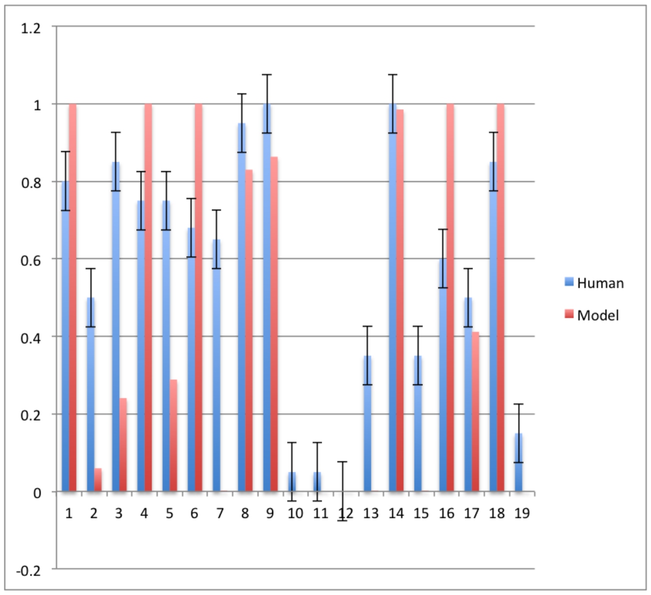

Figure 4: Chart showing $p(s \in \mathcal{L} | S \subseteq \mathcal{L})$. Human subject data is in red, model data is in blue. $R = .75$

We note that subjects completed the 10 or 11 question survey on average in around three minutes. It remains unclear if more standardized results would have emerged in certain cases if subjects had taken more time with the task. In addition, more fine grained results may have been obtained if we had instead allowed users to bet certain amounts on whether or not a test string was in S.

We note that in cases 1 (S = {`yyyt`, `yt`, `yyyt`, `yyyt`, `yt`}, s= `yyyyyt`), 4 (S = {`jojojojo`, `jojojojo`, `jojojojo`}, s = `jojo`), and 6 (S = {`twtwtw`, `twtwtw`, `twtwtw`}, s = `tw`), it is clear that the model weighs a smaller number of transitions greater than human subjects do. However, because this same parameter in part controls for wildcards, lowering it reduces performance on many other cases. This suggests that our model's parameter β may in fact be better represented by more than one variable in human decision making.

We also observe that in such cases, there was a relatively small set of probable hypotheses, and that any favoring of smaller hypotheses was magnified by this fact.

Conclusion and Future Work

future work idea:

-different wildcards (numbers, etc). this was implemented, but we didn't implement any experiments

Despite the number of ways in which our model might be improved mentioned above, it remains highly implausible that the brain generates its set of candidate hypotheses by ever considering even 2^n hypotheses in the first level of beam search we perform, much less more than that. Doing so would be highly inefficient: the number of implausible hypotheses generated would in most cases be far greater than the number of 'good' hypotheses. Far more plausible a priori is the hypothesis that the brain uses of number of heuristics to generate a certain set of candidate hypothesis, and then uses some process analogous to Bayesian inference to select among those hypotheses. To shed light on how this hypothesis selection process may occur, we designed an experiment in which candidates chose among a number of possible rules to explain a set of strings S, but have yet to gather enough data to reach any significant conclusions.

Further experiments could be conducted to determine what heuristics may be at work in the hypothesis generation process. For any set of candidate heuristics, it will remain to be determined whether or not such heuristics are at some point learned. Clearly some semantic considerations are at work in human pattern recognition on strings: cat and mat but not qat are meaningful english words; 'ao' and 'eo' and 'io' begin with vowels unlike 'xo'. (This experiment was designed to avoid such candidate hypotheses.)

We note that we do not directly solve the problem of finding a most plausible way to align a set of strings S. Humans are excellent at quickly noticing the pattern in 'aall', 'ball', and 'call'. Although our model will infer that the last three letters can be represented with only three states in each case, in doing so it generates many candidate regular expressions which a human being would not consciously consider.

Future work may be directed towards investigating the hypothesis that humans first align candidate strings and then generate a number of hypotheses. For example, humans would be very quick to notice that "shrew", "reward", and "threw" all contain the 'rew' subsequence, and would most likely generate candidate regular expressions which preserved that substring. However, following Fernau (2009) we note that the related task of the multiple sequence alignment problem is NP hard, and that a similar model which considers the most likely alignment of a set of strings given many possible alignments will consider a very large number of hypotheses when strings are long enough.

References

Fernau, H. (2009, April). Algorithms for learning regular expressions from positive data. *Inf. Comput.*, *207*(4), 521–541. Available from `http://dx.doi.org/10.1016/j.ic.2008.12.008`

I. Hwang, A. S., & Goodman, N. D. (2011). *Inducing probabilistic programs by bayesian program merging* (Tech. Rep. No. arXiv:1110.5667v1). Pittsburgh, PA.

Tenenbaum, J. (1999). *A bayesian framework for concept learning*. Unpublished doctoral dissertation, Massachusetts Institute of Technology.

Appendix: Technical Notes

Regex and DFA Notation

We used a limited alphabet wherein normal characters are all lower case. We also used the following rules to define regular expressions.

1. Acceptance of specific strings is represented as concatenated characters. E.g. `abc` accepts only `abc`.

2. Acceptance of disjunctive strings is represnted by `[a|b]`. E.g. `[abc|c]`accepts only `abc` and `c`.

3. Acceptance of any character is represented by the wildcard `A`. E.g. `aA` accepts `aa`, `ab`, `ac`, etc.

4. Acceptance of the empty string is represented by `E`. `Ea` is the same as `a`.

5. Acceptance of any sequence repeated zero or more times is represented by the Kleene star, `a*`. E.g. `a*` accepts `E`, `a`, `aa`, etc,

Sizing the Hypothesis Space

Ignoring wildcards and ignoring the mergings that require subset construction, each hypothesis can be defined as a set of mutually exclusive subsets of the states contained in the base hypothesis. For instance, given states s_1, s_2, s_3, the possible hypotheses are:

$$\begin{aligned}&s_1, s_2, s_3\\&\text{merge}(s_1, s_2), s_3\\&\text{merge}(s_1, s_3), s_2\\&\text{merge}(s_2, s_3), s_1\\&\text{merge}(s_1, s_2, s_3)\end{aligned}$$

To generalize this to a case of n_b states in the base hypothesis, one can consider individually the number of regexes with n_f states in the finished regex. For the cases where $n_b = n_f$ or $n_f = 1$, it is easy to verify that there exist only one regex. When creating hypotheses where $1 < n_f < n_b$, however, one can simply look to the case of $n_b - 1$. In particular, for every n_f, one can duplicate all of the hypotheses from the $n_b - 1$

case with an additional unmerged state s_{n_b}. Further, one can add a new hypothesis for each subset of the hypotheses in the $n_s - 1, n_f$ case. So, the expression for the total number of regexes given the number of states in the base hypothesis is given by $g(n_b)$.

$$f(a,b) = \begin{cases} 1 & a = b \vee b = 1 \\ f(a-1,b-1) + bf(a-1,b) & \text{otherwise} \end{cases}$$

$$g(n_b) = \sum_{n_f=1}^{n_b} f(n_b, n_f)$$

One of our experiment examples contained 25 nodes in the base hypothesis, which means that the hypothesis space is $g(25) \approx 5.0 \times 10^{17}$ regular expressions. Needless to say, it is intractable to exhaustively search the hypothesis space.