Finding Regular Expressions Given a Set of Strings

Brendan Farrell

March 8, 2022

1 Main Idea

We view a regular expression as a description for a set of strings and propose an approach to determine a regular expression for a given set of strings. An analogy for this task to be determine a curve around a set of points where one must balance two criteria: the size of the set should not be too large, yet the boundary should not be too complicated.

Definition 1. An *alphabet* is a set of characters, and a *string* or *word* is a finite sequence of characters. A *regular expression* is a description of a set of strings according to a certain format. We use the following formats, which are from the python language but do not include all options available in python:

1. Our alphabet is the ASCII characters 32-126, i.e. digits, uppercase and lowercase letters and punctuations.

2. A *class* is a set of characters that is written in one of the following ways:

   (a) any character is denoted by a period (".").

   (b) a single character, e.g. 'b'

   (c) a set of ranges within the ASCII ordering, e.g. [a-m!-%] is characters 97-109 and 33-37.

   (d) the specific classes \d (all digits), \D (all non-digits), \w (all word characters: a-z,A-Z,0-9,_) , \W (all non-word characters, i.e. punctuation)

3. A class is followed by a *quantifier*. When no quantifier is written, the number of elements from the class is 1. The quantifier * means zero or more; ? means zero or one; + means one or more; {k,} means at least k; {,k} means up to k; and {k,l} means at least k and no more than l.

4. The statement $RE_1|\cdots|RE_k$ means RE_1...or...RE_k and will be called an *or statement* Parentheses are put around an or statement when it is concatenated with another statement.

5. We will not use any other regex symbols, e.g. the symbol ˆ (not), \b (beginning or end of a word), or \s (any white space), etc.

Definition 2. A regular expression is *simple* if it is a class followed by a quantifier or if it is an or statement. The symbol ϵ is the empty string and is used as a placeholder.

Definition 3. The *entropy* of a regular expression is the log_2 of the number of strings that satisfy the regular expression and is denoted *ent* . The *complexity* of the regular expression is the length of the regular expression, i.e. the total number of characters required to write the regular expression and is denoted K . The function ϕ on regular expressions is

$$\phi(RE) = ent(RE) + \alpha K(RE), \tag{1}$$

where $\alpha \in (0, \infty)$ (generally $(0, 1]$) is a weight.

Given a set of strings, we will determine a regular expression that all of the strings satisfy and that has minimal ϕ-value. The entropy is the size of the regular expression, and the complexity is how complicated it is. The complexity is based on Kolmogorov entropy, hence the letter K, but is somewhat different. Kolmogorov complexity of a string is the length of the shortest computer program that can write the string. Here, we only consider the shortest regular expression that can describe a set; that is we only allow specific ways of describing the sets.

2 Details

2.1 The Length 1 case

We start by looking at strings of length 1. Let S be the set of all characters appearing in a set of words of length 1. Given α, a function exists that returns a ϕ-minimizing regular expression for any set S. For example, if $S = \{0, 2, 4, 6, 8\}$, then two possible regular expressions are $r_1 =$ [02468] and $r_2 =$ \d . The ϕ values of these are $\phi(r_1) = log_2(5) + \alpha 7$ and $\phi(r_2) = log_2(10) + \alpha 2$, so that the ϕ-minimal regular expression depends on the value of α.

We only work with ASCII characters 32-126, so a set of characters can be represented as a vector of zeros and ones of length 94 and passed to a function with the parameter α to return a ϕ-minimizing regular expression.

2.2 Handling longer strings

Now consider n strings of length at most N. We arrange each word as a row in a table with one letter in each column, and words that have length less than N are padded with ϵ. The table for the set $\{house, mouse, cat\}$ is

$$\begin{array}{ccccc} \mathrm{h} & \mathrm{o} & \mathrm{u} & \mathrm{s} & \mathrm{e} \\ \mathrm{m} & \mathrm{o} & \mathrm{u} & \mathrm{s} & \mathrm{e} \\ \mathrm{c} & \mathrm{a} & \mathrm{t} & \epsilon & \epsilon \end{array} \tag{2}$$

We view each column as a set and then use the Length 1 Case to determine a regular expression composed only of classes for each column. That is if $(c_1|c_2)$ is the ϕ-optimal regular expression for a column, where c_1 and c_2 are classes, we record these as two separate classes. We now have N columns and for each column we have at least one class. We create a rooted tree with an empty root and a vertex from the root to each class in the first column. A vertex then connects each class in column i with each class in column $i+1$ such that a word in the set exists with a character from the class in column i followed by a character from the class in column $i+1$. Any word in the set can now be created by following a path along the tree.

2.3 Reconstructing a Regular Expression from the Tree

We apply the following rules to the tree to simplify it.

2.4 Reducing the ϕ-value of the Tree

The original tree is in general not ϕ-optimal.