Regular Language Recognition System

Ștefan Vodiță

Faculty of Computer Science and Automatic Control, Polytechnic University of Bucharest

ABSTRACT

Aims. The aim of this paper is to investigate the possibility of solving formal language regularity by using the pumping lemma in a programmatic way.
Methods. The algorithm used gives random inputs for the lemma to be validated. This is repeated until the lemma is contradicted or a number of attempts deemed sufficient by the configuration has been reached.
Results. The data generated by the algorithm was clustered and non-regular languages could be split into several clusters depending on the number of tries needed on average to prove their non-regularity. Correctness could not be achieved for regular languages due to the limitation of having finite length words.

1. Introduction

Currently, there aren't any available language regularity calculators based on the pumping lemma which accept as their only input the language. This posed the question of whether such an algorithm can be implemented and, if it were, what information could be extracted from it.

The formulation of the pumping lemma we are going to refer to is as follows:

Let L be a regular language. Then, any word $w \in L$, with $length(w) \geq n$, where n is the pumping length, $n \in N$, can be written as w = xyz, with:

- $length(xy) \leq n$
- $length(y) \geq 1$
- $xy^kz \in L, (\forall)k \in N$

Since the lemma would need inputs other than the language, these were to be constructed randomly, and the algorithm would need to be run until either the lemma had been contradicted, thus proving the language non-regular, or a sufficient number of attempts had been made so as to answer with a certain degree of confidence that the language is regular.

2. Methods

2.1. Language Description

The language descriptions are separated in two: the word form and the conditions. For example, *aˆN|N>5.* describes the language containing words of type a^N with the condition of N being greater than 5.

2.2. Algorithm

The algorithm works as follows:

1. Check whether the language is finite. If all variables in the language description have a condition which imposes an upper limit, such as $N < 5$, then the language is finite, thus regular, and computation stops.
2. Choose a random word $w \in L$ and random $n < length(w)$
3. Select an xyz split of w.
4. Test $xy^kz \in L$. Return to step 3 if not all xyz splits have been tried.
5. If for every xyz split there was a k such that $xy^kz \notin L$, then the language is non-regular. To keep searching, return to step 2.

For k, an additional restriction, $k \geq 2$, had to be added. Having $k = 1$ is redundant, since $xy^kz = xyz \in L$ in that situation. $k = 0$ causes problems when working with finite words and pumping lengths. This is best illustrated in an example: Consider $L = aˆN|N>5.$, and $w = aaaaaa$ (a repeated 6 times, a^6). In this situation, any y in any xyz split will contain at least one a, as per the $length(y) \geq 1$ condition. $k = 0$ would erase that a, and the language would be considered non-regular.

Extra condition for the pumping length also lead to dramatic changes in the results. These will be discussed in section 3.

2.3. Data Gathering

Three features in the algorithm are of interest:

1. n_w_stop = The number of random words w tried before finding one that fails the lemma
2. k_stop = The value of k for which the lemma fails
3. is_regular = The final verdict, regular/non-regular, represented as 1/0.

Because the algorithm would not identify non-regular languages on every run, when averaging the results the 3^{rd} feature is no longer binary, but can take any value in the [0, 1] interval.

For the purpose of this study, 100 words w were picked for every language. 20 runs were averaged to obtain the final results over the 59 languages.

2.4. Data Clustering

Clustering is only relevant in this case for non-regular languages. For that reason, all languages that the algorithm consistently found to be regular were eliminated and only true non-regular languages are considered. In other words, this section only concerns non-regular languages that the algorithm identified as such in at least one run.

This implies the first two features, the number of words tried, and k, have meaningful values and the 3^{rd} feature, the regularity, is in the interval [0, 1), with a lower value meaning it was more often identified correctly.

The clustering algorithm used is k-means.

3. Variations

Due to the nature of the problem, perfect correctness is not possible.

Firstly, this is because the problem of finding a language's regularity using the pumping lemma is only semi-decidable. If a word fails the lemma's conditions, then the language is non-regular, but we can never conclude that the language is regular unless all words in the language have went through the computation. This is, of course, impossible to do on infinite languages. This means we *expect* the algorithm to give false positives (considers language regular when it isn't).

Unfortunately, false negatives also appear, in a way they would not in a pen-and-paper computation due to the limit of having finite words and pumping lengths.

The algorithm can be tweaked to improve its answers by imposing more limitations on the pumping length. 3 versions will be discussed:

1. The first approach relies strictly on the pumping lemma's formulation: for each random w, a random pumping length n is chosen, that obeys $length(w) \geq n$. All possible xyz splits are then evaluated. As discussed earlier, thought, this leads to false negatives.
2. The second approach improves on the first by reducing the rang of possible pumping values. In the first case, $length(xy) \in [1, n]$, where n is a random number at most equal to *length(w)*. By raising the lower bound, a number of the false negative cases can be avoided. For this purpose I have set the lower bound to be greater than the number of symbols in the alphabet. To exemplify how this helps, let's consider $L = ab^N|., w = ab, n = 1 \implies y = a \implies xy^kz = a^kb \notin L, k \neq 1$
3. Despite the improvements in version 2, the algorithm still produces false negatives. To further improve the results, the pumping length's upper bound can also be changed to be exactly *length(w)*. Now, $length(xy) \in [1, length(w)]$. However, this version favours regularity, and increases the number of false positives. In the first test set, of 41 languages, only 28.57% of non-regular languages were identified. For this reason, the results section will not deal with version 3, as the results skew towards regularity, and, as such, reduce the clustering data set.

4. Results

4.1. Calculator Algorithm Correctness

The following were achieved:

- **Version 1**
  False positives out of total positives: **27%**
  False negatives out of total negatives: **33%**
- **Version 2**
  False positives out of total positives: **12.5%**
  False negatives out of total negatives: **23%**
  We can see the improvement brought on by raising the lower bound of the pumping length.
- **Version 3**
  False positives out of total positives: **29.8%**
  False negatives out of total negatives: **9%**
  Removing the pumping length's random upper bound decreases the number of false positives substantially, but increases the number of false positives.

We can see that version 3 is preferable for minimizing false positives, but, as a whole, version 2 performs the best.

4.2. Clustering

Two questions need to be answered in this context for a useful clusterization:

1. How should we scale the features?
2. How many clusters should there be?

Regarding the first question, with the 3 features we evaluate, each happens to be one order of magnitude away from the next: $is\_regular \in [0, 1), k\_stop \in [2, 10], n\_w\_stop \in [0, 100]$.

This would make for an easy 10, 1, 0.1 scaling to bring all the features in the [0, 10] interval. However, if we take into consideration that the clusters will represent the difficulty with which the result was achieved (needing to try more words means the result is more difficult to obtain), then it is reasonable that n_w_stop would influence the clustering more than k_stop.

There is a case to be made that is_regular should be the most significant feature, as a value of 0.95 here means the algorithm almost always goes through all 100 words without finding anything. We can account for this by adding to the average the maximum k value and maximum number of words w when the algorithm goes through all of them.

Regarding the second question, we will be plotting the number of clusters against the sum of squares within clusters and look for the inflexion point in the graph.

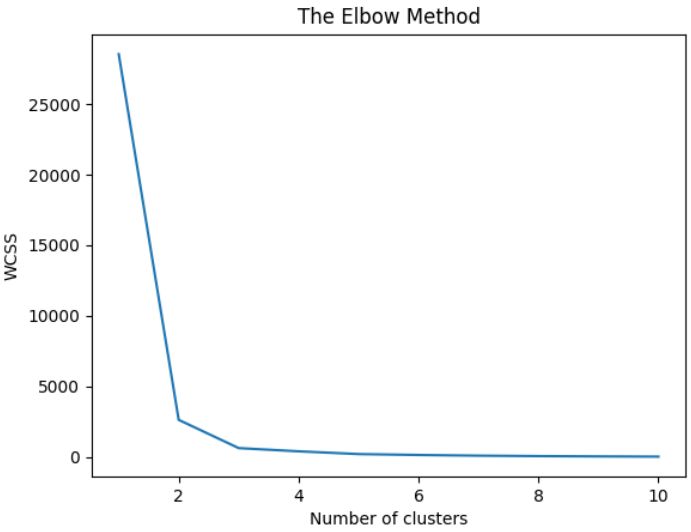

Fig. 1. The Elbow Method for version 2 without scaling

In **Fig. 1.** we see the sum of squares drop drastically from 1 to 2 clusters, and then drop again for 3. After that, the decrease is linear. As such, we will be classifying our data into 3 clusters.

Fig. 2. shows 3 clusters in blue, green and red. More intense colors mean more points (more languages) are overlapped at the coordinates.

The blue cluster represents easily provable non-regular languages. Interestingly, they stop suddenly at 20 word attempts, and from 20 to 60, there are no languages present. After 60 words, the green cluster appears. These are hard to prove non-regular, but they can consistently be proven so. Lastly, the red cluster represents languages that are often miss-classified and hard to prove non-regular.

If we scale the features so that they are as meaningful, we get a similar result in **Fig. 3.**

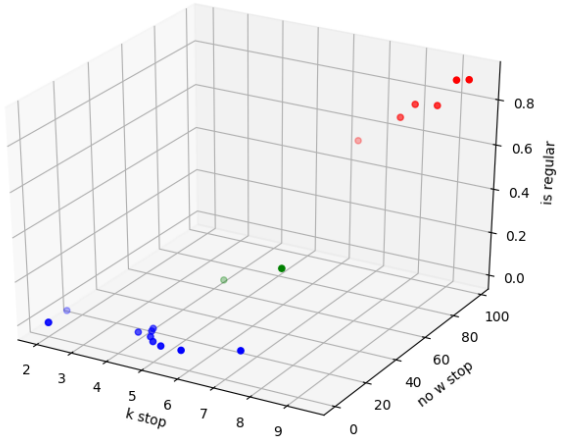

Fig. 2. Division into 3 clusters for version 2 without scaling

However, we should take into consideration that in this case, looking at **Fig. 4.**, we see a linear decrease from 2 to 4 clusters. 3 does not appear to be significant.

If we cluster into 4, as in **Fig. 5.**, we no longer get the red outlier, and instead have a clearer split into easy, hard, and miss-classifiable, with the addition of a green cluster representing very easy languages, with both low w counts and low k values.

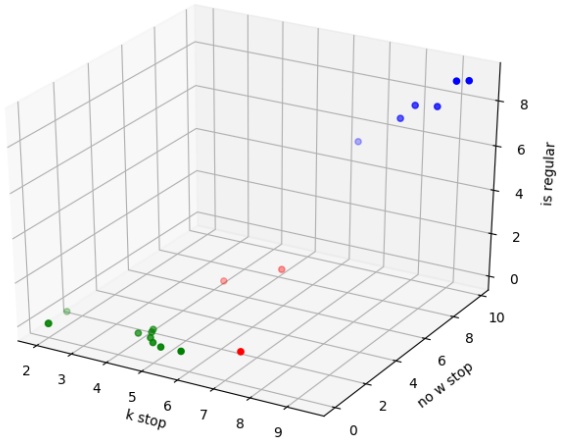

Fig. 3. Division into 3 clusters for version 2 with scaling

4.3. Conclusions

We have shown what difficulties a language regularity confronts and proposed ways to alleviate them. With the results from the algorithm we have further divided non-regular languages based on the difficulty of proving their non-regularity.

5. Using the Calculator

All resources, including the code for the algorithm, the clustering code, the data sets, and the graphs, are available on GitHub: pumping-lemma-calculator

For instruction on how to run the calculator, see the readme file.

Acknowledgements. Many thanks to Irina Mocanu for her support and the advice given on developing this subject.

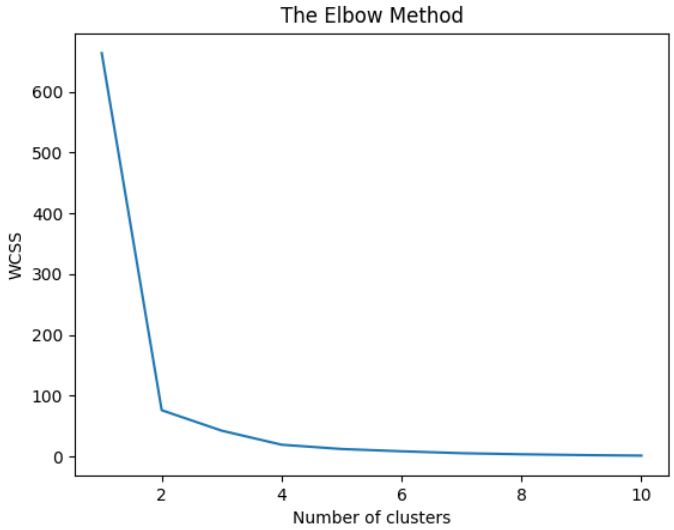

Fig. 4. The Elbow Method for version 2 with scaling

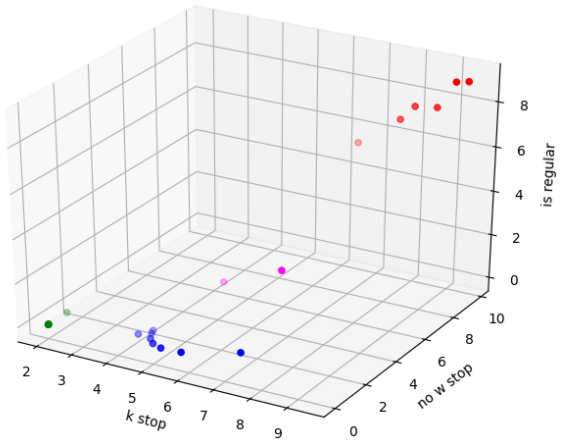

Fig. 5. Division into 4 clusters for version 2 with scaling

References

Pumping lemma calculator as a game: "The pumping game"
Fouad B. Chedid, 2010, "On proving languages non-regular"
Cogliati, 2004, "Visualizing the pumping lemma for regular languages"
Gerry, Ravikumar, 2011, "On Approximating Non-regular Languages by Regular Languages"