Text based Markov Models Using a Sequence Alignment Algorithm

Carl Davidson
Department of Computer Science
Simpson College
701 N. C St. Indianola, IA 50125
carl.davidson@simpson.edu

Abstract

Random text generation is a feature common among recreational computer applications. One common implementation of the feature involves the use of *n-gram Markov models*. Given a similar sample string of tokens, such implementations determine the probably a token will follow a string of tokens of size n, known as an n-gram.

n-gram models have met success in random text generation, but suffer flaws. Here, a potential improvement upon n-gram models is described, in which new n-grams are built upon a similar n-gram already within the model. Similarity is determined using a sequence alignment algorithm.

Results produced by the algorithm seem promising – the model generates text that is arguably more convincing than the output of standard Markov models, and the model is capable of generating novel output when given sample text that is typically too short for standard n-gram models. Complexity of the algorithm is analyzed, and future improvements are proposed.

Introduction

Random text generation is a problem addressed in numerous recreational computer applications. Given a sample string of tokens (most often either words or characters), the problem is to generate a semi-random string of tokens that somehow resembles the sample string without duplicating it. The problem is intimately tied with the decoding problem: given a sample string of tokens that represent the output of a finite state machine, reconstruct the finite state machine [1]. The reconstructed finite state machine may then be used to generate random text that is similar to the input, thus easily addressing the random text problem.

One common solution to the decoding problem involves the use of *Markov models*. Given a sample string of tokens, such solutions determine the probability one token will follow another in the sample string [2]. The resulting finite state machine can be represented as a directed cyclic graph. One improvement upon this solution determines the probability one token will follow a string of tokens of size *n,* known as an *n-gram.* Such implementations use what are known as *n-gram models* [2]. While *n-gram* models may also be represented as a directed cyclic graph, a simpler representation may be a tree in which paths represent n-grams and the tokens that follow them.

n-gram models have met success in random text generation, but suffer flaws [3,4]. Because each substring of size n is recorded from the sample text, the model is spatially complex. Spatial complexity may be reduced by representing n-grams in a tree as previously described, but the model remains complex relative to competitors. The model also requires sample data large enough to contain a variety of tokens following each n-gram, or else generated text will be virtually identical to the sample text [3,4]. This relates loosely to what is known as the *curse of dimensionality,* in which search space increases exponentially in relation to the number of dimensions [3,4]. Finally, the model fails to detect certain patterns within sample text [3,4]. For instance, when given “the quick brown fox jumps” and “the quick grey fox sneaks,” a trigram Markov model will not generate “the quick brown fox sneaks” or “the quick grey fox jumps” [3]. The use of large sample text may obscure the problem, but the model will require more memory and may still fail to detect certain patterns.

Various models have been proposed that improve upon or compete with n-gram models. Neural networks predict the next character of a string in a manner inspired by neurons in the brain [3]. Hierarchical Markov models consist of several Markov models, which are themselves states in an even greater Markov model [4]. Here, we describe one improvement in which tokens are associated with n-grams based upon a sequence alignment algorithm.

Sequence alignment algorithms are often used in Bioinformatics to analyze correspondences between DNA sequences sharing a common descent [1]. Given 2 sequences, such algorithms find a list of tuples representing alignments between nucleotides of each sequence [1]. In the context of text generation, such algorithms could compare an existing Markov model with a new segment of sample text, returning a list of tuples representing alignments between states of the model and tokens of the string. This list can then be used to incorporate the new segment into the model.

Implementation

Perhaps the greatest deviation from typical sequence alignment algorithms seen here is that a string must be compared to a graph, rather than another string. This graph represents an existing model that must be modified to accommodate for the string with which it is aligned. Because of this deviation, the algorithm used here could incorporate some aspects of network alignment algorithms. Given 2 graphs, such algorithms find a list of tuples representing alignments between nodes in each graph [5]. Such algorithms are also common to Bioinformatics, where they are used to analyze correspondences between biological networks in species sharing a common descent [5]. At the time of implementation, such algorithms were not considered, and the algorithm described here is simply a modified sequence alignment algorithm.

The basis for our sequence alignment algorithm traverses all possible pairings between 2 strings, storing results in a 2D array to prevent considering the same alignment more than once [1]. Because the algorithm seen here must compare a string to a graph, it traverses all possible pairings recursively, storing results in a 2D hash table. The recursive function accepts one node from the graph and one token from the string, and determines the best alignment that starts at the input. To do so, it determines the best alignment in the event either input is misaligned. If the inputs match, it also determines the best alignment in the event the inputs align. Each alignment is assigned a score based on the weighted number of matches, mismatches, and misalignment's within. The function returns the best overall alignment found, along with that alignment's score. Results of each function call are stored in a hash table indexed by tuples representing parameters, and are retrieved in the event the function is passed the same parameters. This is similar to the operation performed for each cell of the 2D array in the previously mentioned algorithm [1]. The algorithm can be represented in pseudocode as follows.

```
visited←new HashTable[]
Align(node, token):
    if (node,token) ∈ visited.keys:
        return visited[node,token]
    if node.links ≠ ∅ and token.links ≠ ∅:
        return 0, ∅

    (bestScore, bestPath)←(-∞, ∅)
    for link1 in node.links ∪ null:
        for link2 in token.links ∪ null :
            (newScore,newPath)←Align(link1, link2)
            if link1 = link2:
                newScore←newScore + 1
            newPath.add((link1, link2))

            if bestScore < newScore:
                (bestScore,bestPath)←(newScore,newPath)

    visited[node,token]←(bestScore,bestPath)
    return (bestScore, bestPath)
```

The `links` attribute is the set of all nodes to which a node links. For the sake of extensibility, sample text can be represented as a directed acyclic graph in which `token` is a node. As a result, `token` also has the `links` attribute.

Upon finding the best alignment, the string must be merged into the graph using the newfound alignment. This is done through the following function.

```
Merge(graph, alignment):
    marker←graph.start
    for each node, token in alignment:
        if node ≠ null and node = token:
            marker.link(node)
            marker←node
        else if token ≠ null:
            detour←new Node(token)
            marker←marker.link(detour)
            marker←detour
```

The function traverses the nodes of the graph in the order they appear in the alignment. When the function encounters a token that does not resemble an adjacent node of the graph, a new node is created to represent the token. In a subsequent iteration, the new node will link to the children of sibling nodes, as if it were a “detour” from a previously existing path. The `link()` function creates a link between nodes, and strengthens a link if one already exists. The strength of a link between nodes corresponds to the number of times those nodes appear next to one another in alignments with the sample text, and is identical to the probabilities assigned to nodes in traditional Markov models.

It is important to note the existing implementation only generates directed acyclic graphs. The modified sequence alignment algorithm assumes that graphs have a definite start and end. If tasked with extending a cyclic graph, the alignment algorithm would have no natural point of entry, and would experience an infinite recursive loop if one were specified.

Results

A Markov model was implemented as previously described. Tests were performed on a near exhaustive list of names for star constellations used in astronomy, provided by [6]. Names that consisted of multiple words (e.g. "ursus major"), and non-ASCII characters (e.g. "boötes") were omitted. Lower case was used throughout input. Results were compared to basic Markov models and bigram Markov models. Table 1 provides example input and typical results for each model. Similar tests were performed using only constellations beginning with the letter "A". Table 2 provides input and typical output for these tests. Figure 1 also depicts the model generated using this input.

Input	**Basic Markov**	**Bigram Markov**	**Alignment Markov**
andromeda	drelamepeleus	for	**ara**
antlia	cegiogedego	ca	depriornus
apus	a	ces	aquies
aquarius	gn	minix	caqus
camelopardalis	cigemecombrcagisancopiambrequmiembanscequpi	pium	aqus
equuleus	cencialecaleica	**hydra**	depriopeiax
eridanus	equlequlibrdaducicicom	hoens	celprolopeiax
fornax	geos	equila	arvus
gemini	demenapphumbr	ca	equies
grus	cema	**ara**	apricorus
lacerta	gibachucequpibama	heus	drius
leo	dum	dra	caries
lepus	foncophulupicegeda	cerpius	caus
octans	fopeleos	do	centara
ophiuchus	chequchecelesencigegiciduagadulibrdrici	pis	arcinus
orion	elienedaqus	**equuleus**	**ara**
pavo	ema	equulum	chamara
vela	mambrciombrdricibambrgemeo	taria	caquarus
virgo	idren	cama	dries
vulpecula	humbrchegiduiamer	cama	centaqus

Table 1. Typical input and output using 3 forms of Markov models. Bold text denotes the names of actual constellations.

Input	Basic Markov	Bigram Markov	Alignment Markov
andromeda	aquantliaquius	**apus**	aqus
antlia	antligapua	**antlia**	apuies
apus	andaqus	**apus**	**apus**
aquarius	antlies	arius	aqus
aquila	a	**aquila**	aquariga
ara	a	auries	aquius
aries	a	**ara**	aquiga
auriga	ariedr	aquara	aquarila

Table 2. Comprehensive input and typical output, using input of smaller size. Bold text denotes the names of actual constellations.

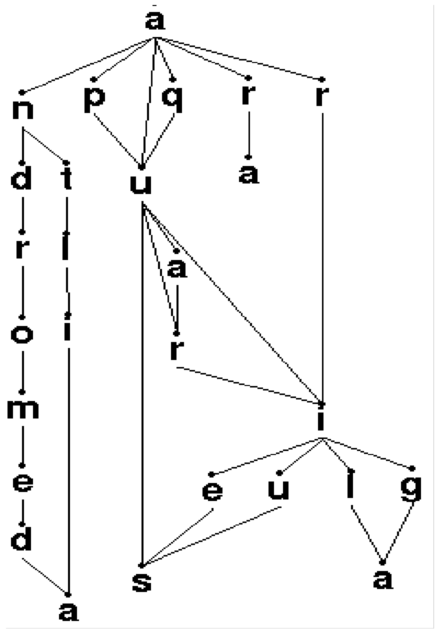

Figure 1. Graph representing the model generated using input from Table 2.

Discussion

Results demonstrate the proposed Markov model generates output that is more convincing than basic Models. Word length of output from basic Markov models was typically far longer or shorter than sample text, but output length from the proposed Markov models resembled input in terms of length. Furthermore, the proposed Markov models require less sample text than n-gram Markov models. This was most apparent on tests using only constellations beginning with the letter “A.” Due to the small size of sample text, bigram Markov models often failed to produce output that was different from sample text, but the proposed Markov model generated text that was consistently original.

In terms of complexity, the proposed Markov model has both strengths and weaknesses. Because the model does not store each substring of size n in its graph, the model is more spatially efficient than n-gram Markov models. However, due to the nature of its sequence alignment algorithm, the model is less temporally efficient. When comparing strings, the sequence alignment algorithm used by the model is of O(NM) complexity, where N and M are the lengths of each string [1]. Though the sequence alignment algorithm is modified to handle graphs, its complexity should remain unchanged, assuming N and M become the number of nodes in each graph. Because the model will ultimately incorporate all tokens of the sample text, the graph representing the model in the worst-case scenario will have the same number of nodes as tokens in the sample text. Based on this alone, one could conclude the model is of O(Nn) complexity where N is the size of the sample text and n is the length of an n-gram. However, sequence alignment is performed for several groups of tokens in sample input. For instance, while testing the model in the previous section, sequence alignment was performed for each constellation in the list of input. It is also possible to perform sequence alignment on each n-gram within the list of input, ignoring natural groups of tokens such as words or sentences. The latter is the worst case scenario, in which the model would be of $O(N^2n)$ complexity. In either case, though, the complexity of the model is greater than the O(N) complexity of n-gram Markov models.

Problems still exist in the implementation seen here. The greatest problem seen yet is a preference to imitate text that occurs early in input. This was most apparent while performing tests on constellation names. Because constellations were sorted alphabetically in the input, the proposed Markov model generated output starting with some of the first letters of the alphabet. At no point during the test did the proposed Markov model generate output that started with a letter beyond “e”. In an attempt to alleviate this problem, the effect a string had upon output was weighted based upon its position in sample text. This marginally alleviated the problem, extending the limit to “k”. It is believed that the earlier a string appears in the input, the more opportunity it has to be aligned with subsequent strings, and the greater the probability that it will be generated. However, it may be possible to take advantage of this problem. If certain patterns are known to exist in sample text, the sample text may be arranged such that these patterns appear earlier in the text. As a result, the model will favor such patterns, and will generate output that is more desirable.

The proposed Markov model also has room for improvement. The alignment function currently assumes the graph is acyclic, though it may be possible for the model to allow for cyclic graphs. Perhaps the easiest way to implement this feature would be to allow the alignment algorithm to jump to distant nodes of the graph, provided the alignment’s score incurs a penalty. It may also be possible to replace the sequence alignment algorithm with an existing network alignment algorithm that accommodates for cyclic graphs. Indeed, there are many algorithms common to Bioinformatics that could be applied to random text generation. A common problem in Bioinformatics is to extract patterns in biological data, such as genetic code, protein composition, or regulatory networks. Algorithms developed to address this problem may extend to natural languages, and the patterns found by such algorithms could be used to generate imitations of these natural languages.

Conclusion

Bioinformatics shares many similar problems with computational linguistics. Both fields seek to extract patterns from extensive and often poorly understood data. Very often, data are sequential – for instance, genetic code or strings of text. As a result, both fields share potential solutions to each other's problems. Techniques such as Markov models are already used for certain problems in Bioinformatics, and it is possible that many other algorithms have yet to be shared [1].

Here, an algorithm common to Bioinformatics was used to construct a natural language model that could generate random text. The model generates text that is arguably more convincing than the output of standard Markov models, and the model is capable of generating novel output when given sample text that is typically too short for standard n-gram models. However, these benefits come at the expense of complexity, and the model prefers to imitate the text to which it is first introduced. Nevertheless, improvements upon the model are possible, and at the very least, the model demonstrates the potential for collaboration between these two fields.

Acknowledgements

This work was done as part of the coursework for CMSC 310 Artificial Intelligence offered by the Department of Computer Science at Simpson College and taught by Dr. Lydia Sinapova.

References

[1] N. C. Jones, and P. A. Pevzner, *An Introduction to Bioinformatics Algorithms*, Cambridge, MA: Massachusetts Institute of Technology, 2004.

[2] J. Allen, *Natural Language Understanding*, Redwood City, CA: Benjamen/Cummings Publishing Company, 1995.

[3] Y. Bengio, et al., “A Neural Probabilistic Language Model,” *Journal of Machine Learning Research*, vol. 3, pp. 1137-1155, 2003.

[4] J. Blitzer, K. Q. Weinberger, L. K. Saul, and F. C. N. Pereira, “Hierarchical Distributed Representations for Statistical Language Modeling,” in *Advances in Neural Information Processing Systems*, 2005, pp. 185-192.

[5] B. P. Kelly, et al., “Conserved Pathways within Bacteria and Yeast as Revealed by Global Protein Network Alignment,” *Proceedings of the National Academy of Sciences*, vol. 100, no. 20, pp. 11394-11399, Sept. 2003.

[6] International Astronomical Union, “The Consellations,” *International Astronomical Union* [Online]. Available: http://www.iau.org/public/constellations/. [Accessed: Dec. 16, 2009].