

UNIVERSITY OF WATERLOO
Faculty of Engineering

Why is a learned approach better than a naive one for determining which insights will prompt a writer?

Wriber Inc.
151 Charles St W, Kitchener, Ontario.

Kevin Carruthers
2B Software Engineering
20463098
kcarruth
April 19th, 2014

Kevin Carruthers
271 Westcourt Pl, unit 111
Waterloo, Ontario. N2L 2R8

May 12, 2014

✓
Andrew Morton, Director
Software Engineering Program
University of Waterloo
200 University Ave. West
Waterloo, Ontario. N2L 3G1

Dear Andrew Morton:

This report, entitled “Why is a learned approach better than a naive one for determining which insights will prompt a writer?”, was prepared during my 2A work term at Wriber, where I worked in the Technical Research and Development department.

My position involved working on all layers of Wriber’s flagship product. Throughout my term, I touched on learning-based statistical analysis, grammatical creation of questions, semantic meaning extraction, back-end optimizations and integrity, and proper library utilities (such as verb conjugation and part-of-speech tagging).

This report will focus on the statistical analysis I have performed throughout my co-op, specifically commenting on why I eventually decided that a machine learning-based model would be preferred to the previous naïve model.

I wish to thank P. Kang for the idea of creating the L^AT_EX template with which I wrote this report.

✓ I hereby confirm that I have received no help, other than what is mentioned above, in writing this report. I also confirm this report has not been previously submitted for academic credit at this or any other academic institution.

Yours,



Kevin Carruthers, 20463098

Executive Summary

Wriber provides a tool through which any writer (e.g. bloggers, students, novelists...) may be given useful prompts and suggestions to ensure that they can complete their work without experiencing “Writer’s block” [1], becoming stuck, or being unsure what to write about. These insights are meant to trigger the writer’s thought-process in such a way as to ensure their writing experience is simple and stress-free.

Wriber is developing a system by which insights can be ranked in order of helpfulness to the writer in order to provide them (i.e. the user) with the insight most useful to them at any given time. This report evaluates ^{list these} several potential algorithm classes against a ^{list these} set of criteria to determine which should be implemented. Since this criteria is subjective, this report uses a variation of Multi-Criteria Decision Analysis (MCDA) [2] to provide a quantitative analysis of these alternatives.

There are two main classes of algorithms, within which there are several subclasses. Within each major class the algorithms are somewhat similar, though there are major differences across the classes.

✓ Ultimately, this report determines an unsupervised machine learning algorithm to be most appropriate. This report recommends that Wriber’s developers undergo whatever training is necessary to determine the most optimal unsupervised algorithm, design that solution, and then implement it.

~~In addition to describing the algorithms and the criteria for their selection, this report will describe the issues raised during the selected solution’s implementation.~~

Table of Contents

Executive Summary	iii
Table of Contents	v
List of Tables	vi
1 Introduction	1
1.1 Current Issues	1
1.2 Problem Statement	2
1.3 Report Structure	2
2 Current Implementation	3
3 Alternative Solutions	4
3.1 Manual Algorithms	4
3.1.1 Flat Algorithm	4
3.1.2 State-based Algorithm	4
3.1.3 Tiered Algorithm	4
3.2 Learned Approaches	5
3.2.1 Unsupervised	5
3.2.2 Supervised	5
4 Criteria	6
4.1 Cost	6
4.2 Ease of Maintenance	7
4.3 Development Time	7
4.4 Implementation Time	7
4.5 Quality Time	7
4.6 Flexibility	7
4.7 Quality of Results	8
5 Analysis	9
6 Conclusion	10
7 Recommendations	11
7.1 Further Recommendation	11
7.2 Issues	11

References	13
-----------------------------	-----------

List of Tables

Table 4.1: Criteria Weighting	6
Table 5.1: Multi-criteria decision-making results	9



1 Introduction

Wriber provides a tool through which any writer (e.g. bloggers, students, novelists...) may be given useful prompts and suggestions to ensure that they can complete their work without experiencing “Writer’s block” [1], becoming stuck, or being unsure what to write about [3]. These insights are meant to trigger the writer’s thought-process in such a way as to ensure their writing experience is simple and stress-free.

In the past, the ranking for which insight was to be shown to the user was based on a simple ranking system: potential insights were assigned a score based on the presence or lack of certain features (e.g. length of sentence, number of verbs, number of proper nouns...) and the insight(s) with the top ranking(s) according to this algorithm would be displayed.

The algorithm which determined the insight’s score was under constant development as developers noticed imbalances or thought of new features to take account of. This algorithm, then, became a confusing mix of terms with no explanation of how the structure arose and was difficult to maintain or improve.

1.1 Current Issues

There are two major issues with this approach: ^{awkward list, better to do in block format} 1. the algorithm must be carefully tweaked and (as it is extremely complex) will likely never be a ‘perfect’ portrayal of what insights writers would most benefit from, and 2. this algorithm assumes that the same insights will be of the same value at any point in the writer’s workflow (e.g. when they have only a title, when they are halfway through their work, when they are adding finishing touches...).

The former issue is one which could be overcome by the lengthy application of brute-force algorithm tweaking, but the work along these lines which had already been done was nowhere near sufficient for the algorithm to be considered accurate—in order to develop a working system in any reasonable amount of time, a better solution must be found.


The later issue is the more important of the two: without ensuring that the generated insights are useful at all times, it would be pointless to market Wriber as a overall solution—instead, Wriber would only be useful for a few minutes for each written work, after which it would provide no value.

1.2 Problem Statement


This report will deal with the possible solutions to these issues, the determination of the correct solution, and the problems faced whilst implementing this solution.

It will also provide follow-up recommendations for potential improvements to be made after the solution has been implemented. These recommendations were determined to be of value to Wriber's goal, but not directly a part of the selected solution.

1.3 Report Structure



This report will first analyze the current implementation, then it will enumerate the alternative solutions (of which the current implementation is one). The decision criteria will be described and the potential solutions will be subjected to these criteria in accordance with the MCDA variation. Finally, this report will conclude with a summary of the decision making process and an analysis of the implemented solution, including any further suggestions and issues faced in implementation.



This report is intended to be read by the developers at Wriber, such that they can make an informed decision as to the solution of the stated problem. Additionally, the scope of this report is Wriber's specific problem instance, though it may be applied to any and all similar problems.

2 Current Implementation

Wriber currently has a single ‘flat’ algorithm which takes in a list of insights and ranks them by extracting certain features and running each insight through that algorithm. This algorithm was developed manually, through the unscientific approach of guessing which features are most useful.

This algorithm has severe limitations, in that several large assumptions were made:

- every aspect of an insight which contributes to its quality has been accounted for (i.e. that all useful features are in the feature vector)
- insight aspects not contributing to its quality have been properly ignored (i.e. that the feature vector does not have extra, unused features)
- the exact effect of each feature upon insight quality has been determined (e.g. a feature’s quality is related to the inverse square of the third power of the aggregate length of its words)
- the various features have been properly weighted (e.g. the effect the length of an insight has upon its quality is exactly three-and-a-half times as important as the relative percentage of academic vs. slang words)
- the user’s ideal insight does not follow different criteria dependant on what they’ve already written about (e.g. how far into their work they are)
- each user’s criteria for determining a ‘useful insight’ is identical

These assumptions are far too extreme to consider the current implementation a valid one. In addition, the current algorithm has ‘grown organically’ over time; it is confusing to read, maintain, or improve and thus results in an overly great expenditure of developer time.

We will thus enumerate all other possible solutions (acquired by company-wide brainstorming) and determine the best possible alternative such that as many of these problems may be solved as possible, in order of those deemed most important.

3 Alternative Solutions

3.1 Manual Algorithms

It is possible to manually create an algorithm by extracting the features of an insight considered most useful and determining the relative weighting of these features. Within the broad category of manual algorithms, there are several more specific solutions which may be implemented.

All manual algorithms are similar to the current approach, though the variations between each specific type are worth comparing.

3.1.1 Flat Algorithm

This is the current approach. Selecting this alternative would mean re-implementing the current solution in such a way that feature additions and/or tweaks are simpler to implement and the code is determined ‘clean enough to be easily understood’.

This algorithm, also, could be further improved by determining which features are important and how much relative weighting should be assigned to each one by attempting many small changes sequentially and determining whether the results have improved or worsened with each step. This would be an essential part of the re-implementation process.

3.1.2 State-based Algorithm

The above algorithm may be modified by additionally taking in the current state (i.e. what the writer has already typed) and modifying the algorithm to account for this. For example, it may be the case that the more words the user has written, the longer they would prefer the insights to be. In this case, the algorithm may involve the addition of the following example term:

$$\text{abs}(\text{number_of_user_input_words} - \text{number_of_words_in_insight})$$

3.1.3 Tiered Algorithm

The standard manual algorithm may be replaced with several tiered ones. For example, if the two important things to be scored are determined to be an insight’s grammatical quality and its relevancy

to the piece of writing, two algorithms may be created. The former (i.e. the algorithm dealing with grammar quality) may not need to differ depending on the user state, though the relevancy-based algorithm might.

More detail needed

3.2 Learned Approaches

A different way of creating this ranking system would be through a machine learning approach. By allowing the system to learn which insights should be given a higher score at a given time, the developers would not need to spend any time manually determining the relative importance of various features and many assumptions can be removed from the end result.

Learned approaches are “reliably good” [4] at systems such as this (i.e. learning to rank [5]), thus this sort of approach should most definitely be considered.

3.2.1 Unsupervised

The machine learning system could be written in such a way that the insights which prompt a writer more often (which could be determined by, for example, keeping track of which topics a user discusses after an insight on that topic is displayed) could be given a higher score, thus influencing the system to show more of these results in the future.

This approach is the standard definition of a machine learning system, in that it ‘learns’ and improves as time goes on. In essence, every time a user writes with Wriber, Wriber gets a bit better for everyone.

More detail needed

3.2.2 Supervised

The system could be designed to learn from a large corpus of manually-ranked data. A large list of insights could be ranked by quality at various system states, then the learning system could be made to determine the algorithm which best implements their preferences.

More detail needed

4 Criteria

There are several criteria which will be used to determine which approach should be selected. Each of these criteria will be assigned a score from 1 to 5, as well as a relative weighting. By taking the sum of each of a method's criteria (multiplied by that criterium's specific weighting), the method with the greatest score will be declared to be the one which best meets the problem statement.

This methodology is similar to MCDA, a very well-known decision-making process, and should be adequate for this situation. The specific variations are to simplify the algorithm used in determining exact weighting and score; since the selected criteria are subjective, this will result in no loss of analysis quality.

Table 4.1: Criteria Weighting

Criterium	Relative Weighting	Percentage Weighting
Cost	1	6.25%
Ease of Maintainance	2	12.50%
Development Time	3	18.75%
Implementation Time	2	12.50%
Quality Time	1	6.25%
Flexibility	3	18.75%
Quality of Results	4	25.00%
Total	16	100.00%

The relative weightings of the criteria are represented in table 4.1 and then described in-depth below, including the determination of thee weightings.

4.1 Cost

Lower costs are always preferred in business. Wriber is dedicated to creating an excellent product, though, and thus places a low weight on the cost necessary to produce any one solution. This criterium is rated from zero to five, with zero being outlandishly expensive and five being free. Developer time (i.e. salaries) has been taken into account and algorithm licensing fees have been determined to be zero in all cases.

4.2 Ease of Maintenance

✓ It is preferable for a system to be as easy to maintain as possible. Code cleanliness guidelines are a significant part of this and should ensure reasonable ease amongst all algorithms, but it is also important to compare the relative algorithm complexity. For example: an algorithm which has been manually written and deals with hundreds of variables will be more difficult to maintain than one which deals with only three, regardless of cleanliness standards.

Though this criterium is important, the overall quality of the solution is of far greater importance. This criterium is rated from zero to five, with zero being virtually unmaintainable and five being 'impossible' to be confused about.

4.3 Development Time

✓ Wriber is working to develop a minimum viable product as soon as possible, thus any solution requiring less development time is preferred. This criterium is rated from zero to five, inversely proportional to the amount of time taken.

4.4 Implementation Time

✓ Separate from development time, this is the time it would take to implement this model with the existing code base. This criterium is rated in the same way as the former.

4.5 Quality Time

✓ The final of the time-based criteria, this deals with any time, over and beyond the previous two categories, which would be required to render this option useful, once it has been programmatically implemented. This criterium is rated in the same way as the former.

4.6 Flexibility

✓ This criterium takes into account the degree to which the solution will allow for user-specific and state-based results. This criterium is rated from zero to five, where zero represents 'not at all' and five represents 'completely'.

4.7 Quality of Results

This is a completely subjective measurement of the potential quality of the final results of using this algorithm. Though this is an important criterium, it is the most subjective of the criteria and should be treated carefully. This criterium is rated from zero to five, where zero represents a completely inaccurate and unuseable solution and five represents a perfect one.

5 Analysis

As shown in table 5.1, an unsupervised machine learning approach best meets the outlined criteria.

Table 5.1: Multi-criteria decision-making results

Criterion	Wt.	Flat		State-based		Tiered		Unsupervised		Supervised	
Cost	1	4	4	4	4	3	3	3	3	3	3
Ease of Maintenance	2	1	2	2	4	2	4	4	8	4	8
Development Time	3	5	15	4	12	3	9	2	6	2	6
Implementation Time	2	5	10	5	10	5	10	4	8	4	8
Quality Time	1	1	1	2	2	1	1	2	2	3	3
Flexibility	3	0	0	2	6	1	3	4	12	4	12
Quality of Results	4	1	4	2	8	2	8	4	16	3	12
Total		36		46		38		55		52	

- More detail needed, why did you assign the specific values the ways you did - this must be explained for the process to be sound

6 Conclusion

The results of this evaluation conclude that though the algorithms are similar in some regards, the unsupervised learning approach is clearly the best option. As such, it should be implemented immediately after an algorithm within that category is selected.

Both time and quality had a large impact in determining which algorithm was selected. Ultimately, the larger importance placed on quality over time was the determining factor.

It is also worth noting that even if creating the best possible system was the sole purpose of this analysis—without regards to which option could be implemented fastest or cheapest—this approach would remain the best solution.

7 Recommendations

This report recommends that Wriber implement an unsupervised learning system in order to produce high-quality results with a minimum of development time.

~~7.1 Further Recommendation~~

It was determined that a higher weighting should be assigned to the quality of results than to the time it takes an algorithm to be implemented. However, in the competitive startup world, one may be in the situation of wanting a minimum viable product right away and only focusing on 'perfection' thereafter.

A supervised learning algorithm would allow us to reach a working prototype in less time than using the unsupervised approach. Since it will require very little time to convert a supervised to an unsupervised approach, it may therefore be useful to complete a supervised approach first, to be replaced by an unsupervised approach when time permits (e.g. before the final product release).

Due to the nature of unsupervised learning problems, delaying the implementation as outlined above would not result in a decrease of insight selection quality. Since the bulk of the benefits received from an unsupervised approach occur when a system has a large number of users (i.e. after product release), the trade off between faster implementation (the supervised approach) and an initially slight insight quality improvement (the unsupervised approach) may be determined to be weighted in favor of the supervised approach.

~~7.2 Issues~~

One of the main issues which will be encountered whilst implementing this solution will be in extracting the features to be used by the learning tool. The general idea behind determining the proper features is to have the smallest list such that every feature contributes to the results.

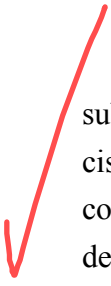
In this case, the best approach may be to:

1. Generate as many features as possible. This could be done through extended brainstorming sessions wherein Wriber developers attempt to think of as many possible text-based attributes as possible. Some potential features may include (for example) the subjectivity of a sentence,

the number of words in common with the user, or the number of times the word ‘aardvark’ appears in a sentence.

2. Iteratively reduce the number of features. This should be done manually: by examining the quality of the results both with and without a given feature, a developer can determine whether removing that feature would be beneficial or detrimental.
3. Iteratively remove ‘zeroed’ features. This should be done manually: a developer could look at the learned theta vector and remove from the programmed feature vector those features which have a theta representation of or near zero, since these features do more harm than good. By removing these features, the learning algorithm should be able to more accurately determine the results of the other features [6].

After following this process, an accurate feature vector and thus an accurate learning implementation should have been achieved.



Additionally, there may exist an issue in determining the best algorithm to be used, within the sub-category of ‘unsupervised learning algorithms’. Though the exact criteria for making this decision is beyond the scope of this report, it would be fairly simple to make the proper decision after completing the feature extraction outlined above. After the feature vectors have been finalized, the determination of the correct algorithm would be a simple matter of running the product with several different algorithms and deciding which gives the best results.

References

- [1] K. Rextin. (2013, Aug.) Start-up spotlight wriber. [Online]. Available: <http://blog.kwstartups.com/post/58069927060/start-up-spotlight-wriber>
- [2] L. Xu and J.-B. Yang. (2001, May) Introduction to multi-criteria decision making and the evidential reasoning approach. [Online]. Available: https://php.portals.mbs.ac.uk/Portals/49/docs/jyang/XuYang_MSM_WorkingPaperFinal.pdf
- [3] J. Zupancic. (2014, Apr.) Wriber homepage. [Online]. Available: <http://wriber.com>
- [4] L.-J. Lin. (2014, Mar.) How machine learning drives better ad performance. [Online]. Available: <http://www.businessinsider.com/how-machine-learning-drives-ad-performance-2014-2>
- [5] H. Li. (2011, Oct.) A short introduction to learning to rank. [Online]. Available: <http://research.microsoft.com/en-us/people/hangli/l2r.pdf>
- [6] K. Fukumizu, F. R. Bach, and M. L. Jordan. (2004, Jan.) Dimensionality reduction for supervised learning with reproducing kernel hilbert spaces. [Online]. Available: http://machinelearning.wustl.edu/mlpapers/paper_files/FukumizuBJ03.pdf

