

Evaluating Wrapped Progressive Sampling for Automatic Algorithmic Parameter Optimisation

Hendrik J. Groenewald, Gerhard B. van Huyssteen and Martin J. Puttkammer
Centre for Text Technology (CTeX)
North-West University
Potchefstroom 2531, South Africa
{handre.groenewald, gerhard.vanhuissteen, martin.puttkammer}@nwu.ac.za

Abstract

Determining the algorithmic parameter combinations that deliver the best performance in applications using machine learning algorithms is a very important part in the development process. Exhaustive searches are slow and computationally expensive, which motivates the investigation of more efficient methods of automatic algorithmic parameter optimisation. Wrapped progressive sampling is one such a method and is utilised in a tool named *Paramsearch*. An alternative method for determining the sizes of the progressive datasets used in the wrapped progressive sampling procedure is proposed and implemented as *PSearch*. *PSearch* and *Paramsearch* are evaluated and compared to an exhaustive search on the tasks of lemmatisation and hyphenation in Afrikaans. Results indicate that both *PSearch* and *Paramsearch* are generally more efficient in terms of execution time and computational resources than an exhaustive search. It is also shown that *PSearch* delivers more accurate results than *Paramsearch* on the tasks of lemmatisation and hyphenation in Afrikaans.

Keywords

optimisation, machine learning, parameters, lemmatisation, hyphenation.

1. Introduction

It is a well-known fact that changing the parameter settings of machine learning algorithms causes large fluctuations in generalisation accuracy [1]. The default parameter settings for machine learning algorithms are not guaranteed to deliver the best performance, while estimating for forecasting the best performing parameters is also generally very hard, due to the complexity of parameter interactions.

One way of finding the best algorithmic parameter settings is to perform an exhaustive search throughout all of the possible combinations of parameter settings. However, this approach is time-consuming and computationally intensive, because the system has to be retrained for every possible parameter setting of the involved algorithm. For instance, the Tilburg Memory-Based Learner [2], a machine learning system, has five algorithms, six distance metrics, five feature weighting possibilities and three class voting weights. The number of nearest neighbours to consider can also be defined for some algorithms. This will amount to circa 4,500 (5x6x5x3x10) different combinations of parameter settings, which implies

that the system has to be retrained 4,500¹ times to evaluate the performance of every combination. Such an exhaustive search can be expected to be a lengthy operation; for example, an exhaustive search throughout all of the valid combinations of algorithmic parameters for one machine learning algorithm on the training data of the Afrikaans lemmatiser took 176 hours and 28 minutes to complete.

The purpose of this study is to investigate alternative, more efficient ways of parameter optimisation than exhaustive searches, in order to obtain the best performing algorithmic parameter combinations for two Afrikaans core technologies (viz. a lemmatiser and a hyphenator), when trained on two machine learning algorithms.

The next section focuses on wrapped progressive sampling as a method for parameter optimisation. *Paramsearch* and *PSearch*, two tools for automatic parameter optimisation are also introduced. In Section 3, various aspects of the performance of *Paramsearch* and *PSearch* are evaluated and compared in terms of classifier ranking, accuracy and execution time. The article ends with some general concluding remarks and suggestions for future work in Section 4.

2. Wrapped Progressive Sampling

2.1 Paramsearch

As an alternative approach to an exhaustive search, Van den Bosch [3] developed a tool (*Paramsearch*) that produces combinations of algorithmic parameters that are estimated to deliver best results. *Paramsearch* implements wrapped progressive sampling (WPS), a combination of classifier wrapping [4] and progressive sampling [5], for data sets containing more than a 1,000 instances. WPS is currently not widely used in the field of natural language processing, but it could be of great advantage if implemented for applications involving the machine learning of natural language.

The general idea behind *Paramsearch* is to determine the best performing algorithmic parameter settings through

¹ This number of experiments is only for purposes of illustration as some of the parameter settings can only be used in conjunction with certain algorithms.

competitions among all the possible combinations of parameter settings, based on their performance evaluated on smaller subsets of the original dataset. The process starts by randomising and dividing the original dataset into training sets (80% the size of the original dataset) and evaluation sets (20% the size of the original data set). The system is then trained with all of the valid parameter combinations on the first subset (containing 500 instances) of the original dataset. The worst performing parameter combinations are discarded after this step, and the size of the training data set is increased. This process of discarding the worst performing parameter combinations and increasing the size of the training data set is continued until only one setting is left, or until the largest available data set has been used for training.

The sizes of the data sets are generated according to the following three-step procedure [6]:

Step 1: Let n be the number of instances in the data set used for training (80% of the original data set). A quadratic sequence of 20 data sets is created by using a factor f as in Equation 1:

$$f = \sqrt[20]{n} \quad (1)$$

Step 2: A sequence of $i = \{1 \dots d\}$ data sets is generated containing $size_i$ number of instances each. For $i=1$, $size_1 = 1$ and then for every $i > 1$, the number of instances contained is defined as:

$$size_i = size_{i-1} * f \quad (2)$$

Step 3: The data sets are then limited to those containing more than 500 instances. A data set of 500 instances is used as the first set. An evaluation set, 20% the size of every generated data set, is also generated for every data set. The evaluation sets are extracted from the 20% of the original data set used for evaluation.

Daelemans and Van den Bosch [6] have evaluated *Paramsearch* on a number of machine learning algorithms and NLP tasks, such as named entity recognition and Dutch morphological analysis. Their main finding was that *Paramsearch* does not produce much effect with algorithms that have little variation in their parameters.

Experimenting with *Paramsearch* on the training data of the Afrikaans lemmatiser indicated that 99% of the parameter settings were discarded based on only 2% of the available training data (see Table 1 below). This raised suspicion about the ability of *Paramsearch* to deliver the best performing algorithmic parameter combinations.

Table 1. Relation between data set size and number of parameter settings evaluated by *Paramsearch*

Data set #	Size (# Instances)	# Evaluated Parameter Settings
1	500	925
2	720	232
3	1,245	13
4	2,154	3
5	3,727	3
6	6,448	1

The problem here is that the small differences between the sizes of the progressive data sets generated at the start of the procedure cause a large number of parameter settings to be filtered out at the start of the process, when the sizes of the subsets are still relatively small in comparison to the original data set. Some of the parameter settings that are filtered out in the beginning could possibly include settings that could have performed very well later on in the process when more data is used for training purposes.

2.2 PSearch

In order to overcome the problem relating to the small differences in the progressive data sets and the fact that *Paramsearch* is only available for two of the five TiMBL classification algorithms, we created our own implementation of *Paramsearch*, which we call *PSearch*. *PSearch* operates on the same principles as the original *Paramsearch*, with the major differences being the way that the sizes of the training and evaluation sets are generated and the number of algorithmic parameter combinations that are discarded after each step in the WPS process. In addition, *PSearch* supports all of the TiMBL classification algorithms, as well as C4.5 [7].

The approach followed by *PSearch* is to discard equations 1 and 2 above for the generation of the progressive data set sizes and instead define the sizes of the progressive data sets as percentages of the size of the available data set. In this way, the 80% of the original data set (i.e. the training set) is divided into 8 subsets, which are calculated² as follows:

- $Size_{(Subset\ 1)} = 0.01 \times Size_{(Original\ data\ set)}$
- $Size_{(Subset\ 2)} = 0.02 \times Size_{(Original\ data\ set)}$
- $Size_{(Subset\ 3)} = 0.04 \times Size_{(Original\ data\ set)}$
- $Size_{(Subset\ 4)} = 0.05 \times Size_{(Original\ data\ set)}$

² The percentages used in calculating the new sizes of the datasets were iteratively determined on the training data of the Afrikaans lemmatiser.

- $\text{Size}_{(\text{Subset } 5)} = 0.3 \times \text{Size}_{(\text{Original data set})}$
- $\text{Size}_{(\text{Subset } 6)} = 0.65 \times \text{Size}_{(\text{Original data set})}$
- $\text{Size}_{(\text{Subset } 7)} = 0.85 \times \text{Size}_{(\text{Original data set})}$
- $\text{Size}_{(\text{Subset } 8)} = 1 \times \text{Size}_{(\text{Original data set})}$

Another difference between *PSearch* and *Paramsearch* is that *PSearch* limits the number of algorithmic parameter setting combinations that are discarded after each iteration when the data set size is enlarged. The number of parameter settings is limited to prevent any excessive decreases occurring in a single iteration. If the number of "surviving" parameter combinations is less than 20% of the number of parameter setting combinations in the previous iteration, the parameter settings in the preceding bin are also included. This process of including more bins of algorithmic parameter settings is continued until the number of "surviving" parameter settings is larger than, or equal to 20% of the number of parameter setting combinations evaluated in the previous round. The advantage of this process is that it prevents *PSearch* from discarding large numbers of parameter settings based on their performance during the early stages of the WPS procedure when the data set sizes are still relatively small.

Table 2 displays the new sizes of the progressive data sets, as well the number of parameter settings evaluated by *PSearch*.

Table 2. Relation between data set size and number of parameter settings evaluated by *PSearch*

Data set #	Size (# Instances)	# Evaluated Parameter Settings
1	800	925
2	1,500	630
3	2,000	238
4	4,000	135
5	15,000	114
6	30,000	72
7	40,000	33
8	57,781	6

Compared to Table 1, Table 2 shows a more gradual decrease in the number of evaluated parameter settings and therefore represents a much more desirable situation, as the chances of good performing parameter settings being eliminated at the beginning of the process are decreased.

In the next section, performance of *Paramsearch* and *PSearch* are compared in terms of ranking of best classifiers, accuracy of best-ranked parameter settings compared to default settings, and execution time.

3. Comparing *Paramsearch* and *PSearch*

The performance of *Paramsearch* and *PSearch* is compared in this section on a lemmatisation (section 3.1) and a hyphenation task in Afrikaans (section 3.2). Both *Paramsearch* and *PSearch* were used to generate combinations of algorithmic parameter settings that are expected to do well on these tasks. These combinations of algorithmic parameters are compared to the results obtained from exhaustive searches. The exhaustive searches were performed throughout all the valid parameter combinations, as tested by *Paramsearch* and *PSearch* for IB1 (the default *k*-Nearest Neighbour algorithm in TiMBL [2]) and C4.5 [7]. 925 different combinations of parameter settings were tested in the IB1 exhaustive search, with 180 combinations for C4.5.

3.1 Lemmatisation

The training data for the lemmatisation task consist of 72,226 instances, with 20 features each. Every instance represents a lemmatised word and the features are made up of letter sequences. The data contains 278 classes, containing information for transforming the inflected word-form to its linguistically correct lemma.

Ranking

Table 3 shows a comparison of the rankings produced by *PSearch* and *Paramsearch* for the IB1 and C4.5 algorithms, evaluated on the training data of the Afrikaans lemmatiser. The ranks as calculated in the exhaustive searches are displayed in Table 3 for the five best combinational settings as predicted by *Paramsearch* and *PSearch*. The exhaustive search ranking signifies a position out of 925 for IB1 and a position out of 180 for C4.5. Thus, a ranking of 1 signifies the best combinational setting out of 925 (or 180 in the case of C4.5), while a ranking of 925 signifies the combinational setting with the lowest accuracy score. Consider for example the combinational setting ranked by *Paramsearch* as the best performing setting for IB1, which in fact achieved a ranking of 271 out of 925 (see Table 3).

Table 3. Comparison of the rankings produced by *PSearch* and *Paramsearch*

Predicted Ranking	Exhaustive Search Rankings (<i>Paramsearch</i>)		Exhaustive Search Rankings (<i>PSearch</i>)	
	IB1	C4.5	IB1	C4.5
1	271	88	1	55
2	149	126	2	56
3	154	135	7	57
4	88	153	11	58
5	89	169	12	59

The results in Table 3 indicate that *PSearch* delivers combinational settings with higher rankings than *Paramsearch*. *PSearch* even delivered the combinational settings ranked 1 and 2 in the case of IB1, which shows that *PSearch* can be assumed to be more suitable than *Paramsearch* for producing good performing algorithmic parameter settings for the lemmatisation task in Afrikaans, especially when used for IB1. A reason for the good performance of *PSearch* is that the larger datasets used at the start of the WPS procedure reduce the chance of discarding settings that performs badly on small amounts of training data, but performs better on larger datasets.

Best setting compared to default setting

Table 4 shows a comparison based on accuracy between the best settings predicted by *Paramsearch* and *PSearch* and the default setting of the involved algorithm. The percentage error reduction is measured as the percentage of error that was saved by *PSearch* or *Paramsearch*. The percentage error reduction may be negative if the predicted settings perform worse than the default setting.

Table 4. Predicted best settings compared to default setting (Lemmatization)

	Best Setting Accuracy	Default Setting Accuracy	% Error Reduction
<i>Paramsearch</i> (IB1)	92.40%	91.36%	12.03
<i>PSearch</i> (IB1)	92.80%	91.36%	16.66
<i>Paramsearch</i> (C4.5)	88.41%	90.81%	-26.09
<i>PSearch</i> (C4.5)	91.21%	90.81%	4.35

The results in Table 4 indicate that the best settings predicted by *Paramsearch* and *PSearch* deliver better results than the default settings, except in the case of C4.5 where the error percentage was in fact increased by the setting produced by *Paramsearch*.

Execution Time

Table 5. Comparison of *PSearch* and *Paramsearch* execution times to an exhaustive search (Lemmatization)

	Execution Time (min)	
	IB1	C4.5
<i>Paramsearch</i>	8.75	0.05
<i>PSearch</i>	29.5	0.78
Exhaustive Search	10 588.00	41.1

Table 5 illustrates the significant advantage in terms of execution times that *PSearch* and *Paramsearch* have over an exhaustive search. *Paramsearch* is also much faster than *PSearch*, but this difference seems to be of less importance

when considering the fact that *PSearch* is more likely to produce better results than *Paramsearch* as far as the lemmatisation task is concerned. The difference in execution times of *PSearch* and *Paramsearch* can be accredited to the larger subsets utilised by *PSearch* and the fact the *PSearch* limits the number of settings that can be discarded after each iteration in the WPS procedure.

3.2 Hyphenation

The comparisons of the previous section are repeated for an Afrikaans hyphenation task. The hyphenation training data consist of 100,000 instances. A context window of 6 characters was used, resulting in 12 features. There are only two classes in the training data, indicating whether the word should be hyphenated at a certain position or not.

Ranking

Table 6. Comparison of the rankings produced by *PSearch* and *Paramsearch* (Hyphenation)

Position as Ranked by <i>PSearch</i> and <i>Paramsearch</i>	Exhaustive Search rankings for settings produced by <i>Paramsearch</i>		Exhaustive Search rankings for settings produced by <i>PSearch</i>	
	IB1	C4.5	IB1	C4.5
1	8	50	1	19
2	9	80	3	22
3	5	74	2	21
4	21	109	4	15
5	25	137	13	28

Table 6 shows that *PSearch* performed better than *Paramsearch* at the hyphenation task for both algorithms. *PSearch* was less successful at predicting the best performing algorithmic combinations for C4.5 when compared to the good results obtained for IB1, but nevertheless still outperformed *Paramsearch*.

Best setting compared to default setting

Table 7 indicates that the default parameter settings of IB1 and C4.5 perform better than the settings predicted as the best by *Paramsearch*. The reason for this is that the default settings performed badly on the first data sets generated at the start of the WPS procedure and was accordingly discarded at an early stage of the procedure by *Paramsearch*. Table 7 further shows that the *PSearch* approach of enlarging the sizes of the datasets used early on in the WPS process and limiting the number of parameter setting combinations that are discarded after each iteration have a positive effect on the results, since *PSearch* was able to predict parameter settings that perform better than the default settings.

Table 7. Predicted best settings compared to default setting (Hyphenation)

	Best Setting Accuracy	Default Setting Accuracy	% Error Reduction
<i>Paramsearch</i> (IB1)	98.2%	98.32%	-7.14
<i>PSearch</i> (IB1)	98.39%	98.32%	4.56
<i>Paramsearch</i> (C4.5)	96.59%	97.21%	-25
<i>PSearch</i> (C4.5)	97.91%	97.21%	25

Execution Time

Table 8. Comparison of the rankings produced by *PSearch* and *Paramsearch* execution times to an exhaustive search (Hyphenation)

	Execution Time (min)	
	IB1	C4.5
<i>Paramsearch</i>	2.51	0.05
<i>PSearch</i>	38.23	1.57
Exhaustive Search	952.1	41.60

The results in Table 8 show the same trend as the results in Table 5. The execution times of *Paramsearch* are generally much faster than that of *PSearch*, but these differences in execution time seem to be very small in comparison to the execution times of an exhaustive search.

4. Conclusion

Results indicate that both *PSearch* and *Paramsearch* can be used to predict good performing algorithmic parameter combinations and that both these two methods are more efficient in terms of execution time than an exhaustive search. *PSearch* did however perform better than *Paramsearch* on both the lemmatisation and hyphenation tasks. An important result is that although the sizes of the progressive datasets as utilised by *PSearch* were customised with the lemmatisation task in mind, good results were obtained when applying *PSearch* to the hyphenation task.

The most important advantage of WPS remains the significant reduction in execution time when compared to an exhaustive search. In this sense *PSearch* has the disadvantage that it has a longer execution time than *Paramsearch*. The difference in execution time can be attributed to the larger sizes of the progressive datasets employed by *PSearch*, as well as the fact that *PSearch* limits the number of combinations of parameter settings than are discarded after every step in the WPS procedure.

The execution time of *PSearch* can however be considered relatively small when compared to the execution time of an exhaustive search.

The performance of *PSearch* and *Paramsearch* seems to be dependent on the machine learning algorithm of choice, the sizes of the progressive datasets, the interactions between these variables and also on the structure of the training data (i.e. features, number of classes etc.). Controlling and predicting these interactions are a difficult task and provides further motivation for experimenting with the sizes of the progressive datasets in the WPS procedure when different machine learning algorithms and tasks are involved.

Determining the best performing parameter combinations in an effective manner remains an important part of the development process of applications using machine learning algorithms. Future work is necessary to determine the relations between the sizes of the progressive training sets employed by WPS and the other variables that may affect the performance of the algorithms. This can be done by extending *PSearch* to more machine learning algorithms and evaluating the performance of *PSearch* on alternative classification tasks than lemmatisation and hyphenation.

5. Acknowledgements

The authors would like to acknowledge the inputs and assistance received from Antal van den Bosch.

6. References

- [1] A. van den Bosch. Wrapped Progressive Sampling Search for Optimizing Learning Algorithm Parameters. Proceedings of the 16th Belgian-Dutch Conference on Artificial Intelligence, R. Verbrugge, N. Taatgen and L. Schomaker, Eds. 2004.
- [2] W. Daelemans, A. van den Bosch, J. Zavrel and K. van der Sloot. TiMBL: Tilburg Memory Based Learner, Version 5.1, Reference Guide, ILK Technical Report 04-02. Tilburg, 2004.
- [3] A. van den Bosch. Paramsearch 1.0 Beta Patch 24. Available: <http://ilk.uvt.nl/software.html#paramsearch>
- [4] R. Kohavi and G.H. John. Wrappers for Feature Subset Selection. Artificial Intelligence Journal, 1997.
- [5] F. Provost, D. Jensen and T. Oates. Efficient Progressive Sampling. Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining, 1999.
- [6] W. Daelemans and A. van den Bosch. Memory-based Language Processing. Cambridge University Press, Cambridge, 2005.
- [7] J.R. Quinlan. Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA, 1993.