A Probabilistic Iterative Local Search Algorithm Applied to Full Model Selection

Esteban Cortazar, Domingo Mery

Department of Computer Science,
School of Engineering,
Pontificia Universidad Católica de Chile
Vicuña Mackenna 4860, Santiago, Chile
ecortaza@puc.cl, dmery@ing.puc.cl

Abstract. Currently, there is no solution, which does not require a high runtime, to the problem of choosing preprocessing methods, feature selection algorithms and classifiers for a supervised learning problem. In this paper we present a method for efficiently finding a combination of algorithms and parameters that effectively describes a dataset. Furthermore, we present an optimization technique, based on ParamILS, which can be used in other contexts where each evaluation of the objective function is highly time consuming, but an estimate of this function is possible. In this paper, we present our algorithm and initial validation of it over real and synthetic data. In said validation, our proposal demonstrates a significant reduction in runtime, compared to ParamILS, while solving problems with these characteristics.

Keywords. Full Model Selection, FMS, Machine learning Challenge, Iterative Local Search, ILS.

1 Introduction

The Model Selection task can be described as choosing the model that best describes a data set [2]. In the machine learning context, this problem may be interpreted in several different ways, from feature selection to parameter tuning. In this paper we will use a broader interpretation, based on the definition of *Full Model Selection* (FMS) as described by [4]. The FMS problem is defined as: given a pool of preprocessing methods, feature selection and classification algorithms, select the combination of these that obtains the lowest classification error for a given data set. This task also includes the selection of the hyperparameters for the considered methods.

In today's practice, the supervised learning problem is usually solved by applying conventions (e.g. the number of neighbors considered in KNN should be relatively low), ad hoc choices (SVM has worked well in the past, why not apply it now), and experimental comparisons on a limited scale (testing three different classifiers with their default settings and comparing performances). The problem with this approach

is that, while it may return acceptable results, it does not truly consider the particularities of the problem at hand. The advantages of using a more specific solution over the generalized approach have been shown in several studies [6]. An explanation for this improvement is given in the *No Free Lunch Theorems for Optimization* [1]. In a nutshell, this theorem says that any improvement in the performance of a model over one class of problems is offset by a lower performance over another class. Therefore, in order to obtain the best possible performance over a certain data set, the generalist approach should be discarded and replaced by the search for a specific model for the problem at hand.

As mentioned before, the FMS problem explores different combinations of algorithms and their hyperparameters, resulting in a vast search space. Furthermore, in order to accurately evaluate each candidate model, training and testing using some validation technique (like Cross Validation) can take a long time, especially over large data sets, which are not uncommon in this field. This combination of a large search space with a long evaluation time, generates a problem well suited for stochastic optimization techniques.

The proposed approach is to use an *Iterative Local Search* (ILS) algorithm, which are well suited for combinatorial optimization problems like this one. Specifically, an implementation ParamILS [5] was adjusted to solve the FMS problem. ParamILS is a parameter tuning algorithm designed with runtime optimization of algorithms in mind, but can be easily modified to fit the needs of the FMS problem. In this paper we present a new algorithm, called PILS (*Probabilistic Iterative Local Search*), which is specifically designed for combinatorial optimization problems with long evaluation time.

This paper is organized as follows: Section 1 gives an overview of the problem. Section 2 describes the basic operators of ParamILS. Section 3 describes our proposed method, PILS. Section 4 reviews our technique for validating this algorithm and the initial results. Finally, Section 5 is a brief conclusion.

2 Iterative Local Search (ILS)

Iterated local search [5] (ILS) is a general meta-heuristic with two basic operators for generating new solutions. The first is the Local Search Operator, which attempts to find the local optimum in the neighborhood of a solution. The second is the Perturbation Operator, which is applied to the local optimum in order to generate a new starting point for a local search.

A general overview of the ILS algorithm is presented in Algorithm 1:

```
Algorithm 1: Iterative Local Search
loop
  x'  = Perturbation(x*);
  x'' = LocalSearch(x');
  if better(x'',x*)
    x* = x'';
```

2.1 ParamILS

ParamILS [5], is the ILS on which our algorithm is based. This is a simple, but powerful, algorithm designed for parameter tuning that relies on the following definitions:

Solution. As was mentioned before, ParamILS is a parameter tuning algorithm. Therefore, it defines each solution as an array of values, where each position in the array represents a specific parameter. At some point during the algorithm execution, depending on the values of the solution, some of the parameters may become inactive. An inactive parameter is a parameter that, if changed, has no effect on the cost function. The relation between parameters that defines when they become inactive must be defined beforehand and is used to describe conditional relationships among parameters.

Local Search. All ILS algorithms must define the way in which they look for a local optimum in the neighborhood surrounding a particular solution. ParamILS starts by defining a neighbor as a solution that differs from the initial one by only one parameter, as long as that parameter is active. Subsequently, it follows an *Iterative First Improvement* technique for finding a local optimum. This technique takes all the neighbors of an initial solution, in randomized order, and compares them to the initial solution. As soon as a solution is found to be better than the initial one, the process restarts using the new solution as a starting point. This will continue until a solution that is better than all of its neighbors is found (local optimum).

Perturbation. ILS algorithms jump from local optimum to local optimum. In order to do this, they must define an operator that allows them to escape from the optimum they are currently in and restart the local search. This operator is defined as the Perturbation Operator. In the case of ParamILS, a very straightforward technique is used to find a new starting point. The Perturbation Operator is defined as a number of jumps from neighbor to neighbor, starting from the current optimum. The number of jumps will define how different one solution is from its predecessor. A small number will increase the likelihood or falling back on the same local optimum, while a large number of jumps will end up with a completely random starting point.

Better. Any ILS algorithm requires a way of defining if one solution is better that another. ParamILS proposes two options for defining the Better Function. The first is BasicILS, which simply compares an each solution with a user defined cost function. The second is FocusedILS, which uses a variable number of training instances in each evaluation in order obtain results with a lower computational cost. The Better Function is precisely what is modified by our algorithm (PILS), so the implementation made by ParamILS is not explained in great depth here. For a more comprehensive understanding of the Better Functions and the ParamILS algorithm, please refer to [5].

3 Our approach (PILS)

ParamILS proves to be an effective way of moving along the search space finding local optimums. The problem that arises is that, with large datasets, the training and testing time necessary to accurately validate each candidate solution is too long. In turn, this means that, even though only a small portion of the search space is evaluated, a very long time is necessary to do it. In response to this problem we propose a new algorithm based on ParamILS, which redefines the Better Function in order to diminish its runtime. Because of its probabilistic approach, we called it *Probabilistic Iterative Local Search* (PILS)[1].

Definitions. Considering the optimization problem being solved by ParamILS, we define a function which is an estimate of the original objective function, but with a considerably smaller runtime. In exchange for the reduction in runtime, we allow this estimate to be noisy. We model this estimation as shown in (1), where it is described as the objective function $f(x)$ plus a random variable e_x representing noise. Finally, the estimate function is defined so that the mean of several evaluations converges to the objective function, as represented in (2) and (3).

$$\hat{f}(x) = f(x) + e_x \,. \tag{1}$$

$$\overline{\hat{f}(x)} \rightarrow f(x) \,. \tag{2}$$

$$\bar{e_x} \rightarrow 0 \,. \tag{3}$$

Assumptions. We assume that independent evaluations of the estimation function produce independent and identically distributed random noise variables. This powerful assumption allows us to use the *Central Limit Theorem*, with regards to the distribution of the noise mean. After analyzing empirical results from the problem at hand, this assumption has proven to be reasonable. Furthermore, we were able to observe that, even though, the noise variance for different candidate solutions (x) were not the same, they were very similar. This fact is integrated into the PILS algorithm and is, therefore, a necessary requirement for a correct use of this tool.

Algorithm. As before mentioned, PILS uses the same search strategy as ParamILS, but it redefines the function responsible for comparing two candidate solutions. The goal behind the formulation of this algorithm is to decrease the uncertainty only on candidate solutions that could be optimums, in order to waste as little runtime as possible on bad candidate solutions. The way in which it decreases the uncertainty of a candidate solution is by evaluating it several times and averaging the results, which should eventually converge to the objective function. Thanks to the *Central Limit*

[1] The MatLab code for this algorithm can be downloaded from http://dl.dropbox.com/u/3304215/PILS.zip (Note for reviewers: if the paper is published, this code will be linked from our webpage)

Theorem, we can model the average noise of several evaluations as a random variable with mean zero, and a variance dependant on the number of evaluations and the variance of these evaluations.

$$\bar{e_x} \sim N(0, \frac{\sigma^2}{N}) \tag{4}$$

Thus, we can easily define a function that, given several evaluations of two candidate solutions, calculates the probability that one is better than the other. Afterwards, the algorithm decides, based on this probability, which of three possible courses to follow. First, if the calculated probability is either very high or very low, then there is enough certainty to simply compare the two means directly. Second, if the probability is very close to 0.5 then the algorithm assumes that there is not a significant difference between the two candidate solutions and defines the one with the lowest variance as the best. This will, probably, save runtime in future comparisons. Third, if neither of the options mentioned is satisfied, the algorithm calculates a new evaluation of the estimation function for the candidate solution that has the lowest number of evaluations, and begins again.

In order to ensure that this function ends, a maximum number of evaluations parameter was added. In case the maximum number of parameters is reached by both candidate solutions, the two means are compared directly.

Our proposed Better Function can be seen bellow in Algorithm 2.

```
Algorithm 2: PILS Better Function
// x1, x2: Candidate Solutions that are being compared.
// mu1, mu2: Mean of x1 and x2 respectively.
// var1, var2: Variance of mu1 and mu2 respectively.
// N1, N2: Number of evaluations of x1 and x2.
better (x1, x2)
{
  loop
  {
    p = ProbabilityBetter (x1, x2);
    d = |0.5 - p|;
    if d>=Us
      return mu1>mu2;
    if d<=Ui
      return var1<var2;
    if min(N1, N2) >= Nmax
      return mu1>mu2;
    if N1<N2
      evaluate (x1);
    else
      evaluate (x2);
  }
}
```

Variance Estimation. The algorithm described in the previous section requires an estimation of the variance associated with each candidate solution.

$$var\left(\overline{\hat{f}(x)}\right) = \frac{var(\hat{f}(x))}{N} \tag{5}$$

The problem that arises is that we now require, at least, two evaluations of the estimation function, in order to calculate its unbiased variance. Empirical testing showed that the problem went further, because the variance calculated for only two samples was still a very poor estimation. Based on the assumption that the variances of different candidate solutions are similar, the calculation is formulated to consider a Global Variance variable.

$$varEstimate = \frac{GlobalVar*(Nmax-N)+var\left(\hat{f}(x)\right)*(N-1)}{Nmax-1} \tag{6}$$

$$var\left(\overline{\hat{f}(x)}\right) = \frac{varEstimate}{N} \tag{7}$$

The Global Variance strongly depends on the way the estimation function is formulated and the dataset at hand. Moreover, it would require extensive experimentation in order to estimate this variable a priori. For these reasons, a way of approximate this variable in real time is necessary. Our implementation considers a user-defined estimation (prior) and its weight, associated with the level of confidence in this estimation. Throughout the algorithms execution, the Global Variance is a weighted mean that considers the user-defined prior and all the sample variances calculated for different candidate solutions, as shown in (8). Empirical experimentation has shown that the Global Variance variable converges quickly, and is a good estimation of the variance mean.

$$GlobalVar = \frac{prior*weight+\sum_i var\left(\hat{f}(x_i)\right)*(N_i-1)}{weight+\sum_i(N_i-1)} \tag{8}$$

4 Experimentation

When validating a supervised learning model, 10-fold Cross Validation is usually considered an accurate estimation of its prediction abilities. Its downside is that it requires a long time for training and testing each subset. In our experimentation, we compare the use of ParamILS with 10-fold Cross Validation against PILS using a simple Hold Out Validation technique, which should take one tenth of the time but, on average, should converge to the same result.

Using different machine learning toolboxes, like Balu[2] and CLOP[3], and several small datasets for testing purposes, the assumptions listed for the PILS algorithm were

[2] Machine Learning toolbox available at http://dmery.ing.puc.cl/

found to be adequately satisfied. But, in order to accurately show the advantages of PILS over the ParamILS algorithm, a very large number of tests under different conditions were necessary. For this purpose, we developed an artificial objective function that mimicked the conditions observed in our testing of real datasets.

4.1 Real Data Set

In our initial approach, we ran PILS, using Hold Out Validation, against ParamILS, using Cross Validation 10-fold over the Fishbone Dataset presented in [7]. Even though the results looked very promising, where ParamILS took up to ten times longer than PILS to obtain the same classification performance, this was not a very effective way of testing our proposal for several reasons. First, the time required by different candidate models was very uneven, which meant that the result from one execution depended on the search trajectory, more than on the optimization algorithm itself. Second, running either PILS or ParamILS was still a rather long task, which meant that extensive testing, in order to obtain more general results or analyze the relevance of certain parameters, was extremely time consuming. Finally, even though this test allowed us to observe the characteristics of this problem, it did not give us much control over the scenarios we wanted to evaluate.

4.2 Artificial Objective Function

The artificial objective function used for these results was constructed using a mixture of five n-dimensional Gaussian functions. Also, random, but similar, variances were assigned to each point in this search space, in order to emulate the estimation function. Using this artificial data, two functions where created. The first is a simple evaluation of the Gaussian mix, representing the real objective function, equivalent to Cross Validation in the Model Selection problem. The second is an evaluation of the Gaussian mix plus a normally distributed noise, representing the estimation function, equivalent to Hold Out Validation in the Model Selection problem.

Table 1. Number of evaluations necesary for similar performance levels. The time column represents the persentaje of time that PILS would of needed, based on ParamILS.

ParamILS		PILS		Time
Performance	N. of Evaluations	Performance	N. of Evaluations	
82	628.962	83.5	1203.278	19.13
90.8	838.871	90	1685.198	20.09
99	1419.927	97	2867.64	20.20

[3] Machine Learning toolbox available at http://www.modelselect.inf.ethz.ch/

One thousand tests were performed using ParamILS with the objective function and PILS with the estimation function. Table 1 shows the average number of evaluations necessary for each algorithm to obtain similar performance levels. As shown in this table PILS requires approximately twice as many evaluations as ParamILS to obtain similar results. But, considering that this data was mimicking a situation where each evaluation by ParamILS should take ten times longer, it's easy to see the advantages offered by our proposal.

5 Conclusions

Even though further testing is necessary to fully validate our method, the initial results show that our proposal could be very useful in helping to solve the FMS problem. Still, a more in depth analysis of the algorithms parameters is necessary, in order to completely understand their impact on the output and find and adequate configuration for solving this particular problem. In addition to the FMS problem, the proposed algorithm could prove to be useful in solving other optimization problems where the definitions and assumptions listed in section 2.2 are valid.

Acknowledgments. We thank Alvaro Soto, Karim Pichara and Jorge Baier for many helpful discussions regarding this work.

6 References

1. Wolpert D.H., Macready W.G.: No Free Lunch Theorems for Optimization. IEEE Transactions on evolutionary computation, vol. 1, no. 1 (1997)
2. T. Hastie, R. Tibshirani, J. Friedman: The Elements of Statistical Learning. Data Mining, Inference, and Prediction: Springer Verlag (2001)
3. Zhang Q., Sun J.: Iterated Local Search with Guided Mutation. IEEE Congress on Evolutionary Computation (2006)
4. Escalante, H.J. Montes M., Sucar L.E.: Particle Swarm Model Selection. Journal of Machine Learning Research 10 (2009)
5. Hutter F., Hoos H.H., Leyton-Brown K.: ParamILS: An Automatic Algorithm Configuration Framework. Journal of Artificial Intelligence Research 36, 267--306 (2009)
6. Smit, S., Eiben, A.: Parameter Tuning of Evolutionary Algorithms: Generalist vs. Specialist. Applications of Evolutionary Computation, 542--551 (2010)
7. Mery, D.; Lillo, I.; Loebel, H.; Riffo, V.; Soto, A.; Cipriano, A.; Aguilera, J.: Automated Fish Bone Detection using X-ray Testing. Journal of Food Engineering, 105, 485--492 (2011)