Predicting the execution time of the interior point method for solving linear programming problems using artificial neural networks

Sophia Voulgaropoulou$^{1[000-0003-1393-3478]}$, Nikolaos Samaras$^{1[0000-0001-8201-7081]\star}$, and Nikolaos Ploskas$^{2[0000-0001-5876-9945]}$

^{1} Department of Applied Informatics, School of Information Sciences, University of Macedonia, Thessaloniki GR-54636, Greece
svoulgaropoulou@uom.edu.gr, samaras@uom.gr

^{2} Department of Informatics and Telecommunications Engineering, Faculty of Engineering, University of Western Macedonia, Kozani GR-50100, Greece
nploskas@uowm.gr

Abstract. Deciding upon which algorithm would be the most efficient for a given set of linear programming problems is a significant step in linear programming solvers. CPLEX Optimizer supports primal and dual variants of the simplex algorithm and the interior point method. In this paper, we examine a prediction model using artificial neural networks for the performance of CPLEX's interior point method on a set of benchmark linear programming problems (netlib, kennington, Mészáros, Mittelmann). Our study consists of the measurement of the execution time needed for the solution of 295 linear programming problems. Specific characteristics of the linear programming problems are examined, such as the number of constraints and variables, the nonzero elements of the constraint matrix and the right-hand side, and the rank of the constraint matrix of the linear programming problems. The purpose of our study is to identify a model, which could be used for prediction of the algorithm's efficiency on linear programming problems of similar structure. This model can be used prior to the execution of the interior point method in order to estimate its execution time. Experimental results show a good fit of our model both on the training and test set, with the coefficient of determination value at 78% and 72%, respectively.

Keywords: Linear Programming · Interior Point Method · CPLEX Optimizer · Artificial Neural Network

1 Introduction

Various algorithms exist for the solution of linear programming problems. When it comes to the selection of the most appropriate algorithm to use for a set of problems, the question still remains: how to select the most efficient method, in terms of the execution time? CPLEX Optimizer [3] includes several high-performance linear programming algorithms. In this study, we are exploring a first approach towards creating a prediction model for the efficiency of the interior point method. A thorough description of the interior point method would exceed the scope of this short paper, therefore, more information about its steps and functionality can be found in [2, 6, 8].

* Corresponding author

2 Empirical results

2.1 Data

For the purpose of our computational study, 295 benchmark linear programming problems were used from the netlib (25), kennington (13), Mészáros (217), and Mittelmann (40) libraries. The problems were solved with CPLEX's 12.6.1 [3] interior point method and the respective execution time, needed for their solution, was recorded for each problem. The linear programming problem characteristics which were examined in this study and set as input in our model are the following: (i) m: the number of constraints, (ii) n: the number of variables, (iii) $nnzA$: the number of nonzero elements of the constraint matrix, (iv) $nnzb$: the number of nonzero elements of the right-hand side vector, and (v) $rankA$: the rank of the constraint matrix. The execution time was set as the output of our model.

2.2 Neural network model

The neural network model presented in this study has been generated using the scikit-learn toolkit [5]. While testing several statistical environments, we found scikit-learn to be the most suitable for the purpose of our analysis, since it supports numerous methods of supervised and unsupervised learning, model selection and evaluation and transformation of data. The algorithm used for the generation of our model is the Multi-layer Perceptron (MLP). MLP is a supervised learning algorithm that can learn a function $f(\Delta) : R^x \rightarrow R^o$ by training on a dataset, where x is the number of dimensions for input and o is the number of dimensions for output. Given a set of input features and an output target, MLP can learn a nonlinear function approximator for either classification or regression. We also experimented with other supervised learning methods available in scikit-learn toolkit, such as linear regression, lasso regression, ridge regression and decision trees, but we obtained the best results using the MLP method.

The greatest challenge during our analysis was related to the number of hidden layers, which had to be set while testing our models, along with the activation function we had to choose. It was noted that the more hidden layers we have in our models, the worse results we eventually get. The activation function is very significant since it converts an input signal of the last hidden layer to an output signal for the next layer. Commonly used activation functions are the hyperbolic tan function (*tanh*), the logistic sigmoid function (*logistic*) and the rectified linear unit function (*relu*). Such activation functions were tested while repeated and extensive testing was also performed on the number of hidden layers. In addition, scaling and normalizing the original data is a significant issue that we experimented with. Evaluating the metrics of each model, we formed our model using the parameters shown in Table1. Apart from the activation function and the number of hidden layers, the solver selected for weight optimization is *LBFGS*, an optimizer in the family of quasi-Newton methods. *LBFGS* uses a weighted linear summation to transform the input values of previous layers to output values for the next layer. Tolerance value refers to the tolerance for the optimization. For example, if, upon a certain number of iterations, we fail to decrease the training loss or to increase the validation score by at least a value equal to tolerance, convergence is considered to be reached and training stops. The alpha value refers to the L2 penalty (regularization term) parameter, while the maximum number of iterations indicates that the solver will iterate until convergence (determined by tolerance value) or this number of iterations.

The ratio between the training and test set was chosen to be 70 to 30. To evaluate the performance of our model, certain metrics were taken into consideration. The R-squared (R^2, coefficient

Table 1: Model parameters for the MLP method

Algorithm	MLPRegressor
Hidden layer sizes	20
Activation function	relu
Solver	lbfgs
Alpha value	$1e-5$
Maximum iterations	1000
Tolerance	0.0001

of determination) provides an estimate of the strength of the relationship between a regression model and the dependent variable (output), while the Root Mean Square Error (RMSE) is the standard deviation of the residuals. We also include the mean absolute error that measures the average magnitude of the errors in a set of predictions, without considering their direction, and the median absolute error that is insensitive to outliers [1, 4, 7].

Table 2 presents the results of the neural network. For the training set, the model achieved an RMSE value of 123.32 and an R^2 value of 0.78, while for the test set the model achieved an RMSE value of 296.73 and an R^2 value of 0.72. Taking into account the variability in the features of the 295 linear programming models and the metrics' values, it is shown that our model can explain the data reasonably well. As an example, an R^2 value of 1 would indicate a perfect fit of the data, so the current R^2 value proves goodness of fit of our model. Further below, a graphical representation

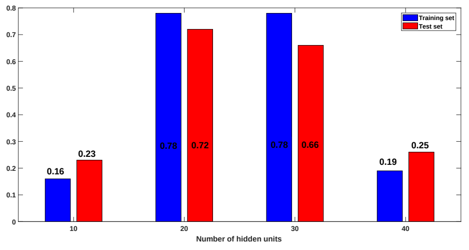
Table 2: MLP model for the interior point method execution time

	Training set	Test set
Root Mean squared error	123.32	296.73
Absolute Mean error	54.31	97.54
Absolute Median error	7.12	9.49
R^2	0.78	0.72

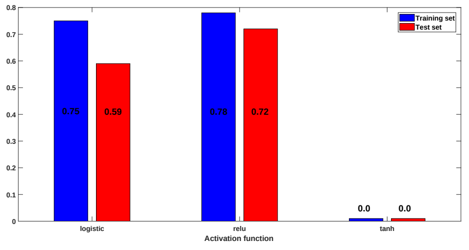
of the comparison between the metrics measured for some of the models we tested is presented in Figure 1 (different number of hidden layers, number of neurons, activation functions, scaling and normalization techniques).

3 Conclusions

An important step in solving linear programming problems is the selection of the most efficient algorithm. Most linear programming solvers have a heuristic procedure to select the most suitable algorithm based on the characteristic of the input linear programming problem. In this paper, we experiment with a neural network for predicting the execution time of CPLEX's interior point method. Experimental results show that the model can achieve an R^2 value of 78% for the training set and 72% for the test set. Taking into account the variability in the features of the benchmark linear programming models we examined and the metrics used for the comparison of the generated models, the current model proves to have a good fit on the data and thus, can be used for further prediction of the algorithm's efficiency.



(a) Tuning the number of hidden layers



(b) Tuning the activation function

Fig. 1: Tuning the parameters in the models

In future work, we plan to build prediction models for the primal and dual simplex algorithm and experiment with various supervised learning methods. Building accurate models for the prediction of the execution of the primal simplex algorithm, the dual simplex algorithm, and the interior point method will lead a linear programming solver to select the most efficient algorithm for a given linear programming problem. That would lead to great savings in solving linear programming problems.

References

1. Draper, N.R., Smith, H.: Applied regression analysis. John Wiley and Sons, 3rd edn. (1998)
2. Gondzio, J.: Interior point methods 25 years later. European Journal of Operational Research **218**(3), 587–601 (2012)
3. IBM ILOG CPLEX: Cplex 12.6.0 user manual. `http://www-01.ibm.com/support/knowledgecenter/SSSA5P\_12.6.1/ilog.odms.studio.help/Optimization\_Studio/topics/COS\_home.html?lang=en` (2017), [Online; accessed 15-February-2019]
4. Kutner, M.H., Neter, J., Nachtsheim, C.J., Wasserman, W.: Applied linear statistical models. McGraw-Hill, 5th edn. (2004)
5. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
6. Ploskas, N., Samaras, N.: Linear programming using MATLAB. Cham, Switzerland: Springer (2017)
7. Rao, C.R.: Coefficient of determination, linear statistical inference and its applications. New York: Wiley, 2nd edn. (1973)
8. Wright, S.: Primal-dual interior-point methods, vol. 54. SIAM (1997)