mathematics

MDPI

Article

Predicting the Execution Time of the Primal and Dual Simplex Algorithms Using Artificial Neural Networks

Sophia Voulgaropoulou [1,†], Nikolaos Samaras [1,*,†] and Nikolaos Ploskas [2,†]

[1] Department of Applied Informatics, School of Information Sciences, University of Macedonia, GR-54636 Thessaloniki, Greece; svoulgaropoulou@uom.edu.gr
[2] Department of Electrical & Computer Engineering, Faculty of Engineering, University of Western Macedonia, GR-50100 Kozani, Greece; nploskas@uowm.gr
* Correspondence: samaras@uom.edu.gr; Tel.: +30-2310-891866
† These authors contributed equally to this work.

Abstract: Selection of the most efficient algorithm for a given set of linear programming problems has been a significant and, at the same time, challenging process for linear programming solvers. The most widely used linear programming algorithms are the primal simplex algorithm, the dual simplex algorithm, and the interior point method. Interested in algorithm selection processes in modern mathematical solvers, we had previously worked on using artificial neural networks to formulate and propose a regression model for the prediction of the execution time of the interior point method on a set of benchmark linear programming problems. Extending our previous work, we are now examining a prediction model using artificial neural networks for the performance of CPLEX's primal and dual simplex algorithms. Our study shows that, for the examined set of benchmark linear programming problems, a regression model that can accurately predict the execution time of these algorithms could not be formed. Therefore, we are proceeding further with our analysis, treating the problem as a classification one. Instead of attempting to predict exact values for the execution time of primal and dual simplex algorithms, our models estimate classes, expressed as time ranges, under which the execution time of each algorithm is expected to fall. Experimental results show a good performance of the classification models for both primal and dual methods, with the relevant accuracy score reaching 0.83 and 0.84, respectively.

Keywords: linear programming; primal simplex; dual simplex; CPLEX optimizer; artificial neural network

MSC: 65Y20; 90C05

Citation: Voulgaropoulou, S.; Samaras, N.; Ploskas, N. Predicting the Execution Time of the Primal and Dual Simplex Algorithms Using Artificial Neural Networks. *Mathematics* **2022**, *10*, 1038. https://doi.org/10.3390/math10071038

Academic Editor: Qing-Wen Wang

Received: 23 February 2022
Accepted: 19 March 2022
Published: 24 March 2022

1. Introduction

Linear programming is perhaps the most important and well–studied optimization problem. Linear programming is the process of minimizing or maximizing a linear objective function $z = \sum_{j=1}^{n} c_j x_j$ taking into account a number of linear equality and inequality constraints. Consider the following linear program in the standard form:

$$\begin{aligned} \min\ & c^T x \\ \text{s.t. } & Ax = b \\ & x \geq 0 \end{aligned}$$

where $A \in \mathbb{R}^{mxn}$, $(c, x) \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and T denotes transposition. We assume that A has full rank, rank$(A) = m$, $m < n$. Consequently, the linear system $Ax = b$ is consistent. Detailed information on the theoretical background and different forms of linear programming problems, along with description of the geometry of the feasible region and the duality principle, is covered by [1,2].

The most well-known method for solving linear programming problems is the simplex algorithm developed by George B. Dantzig [3,4]. It is one of the top ten algorithms with the greatest influence in the 20th century [5]. At the same time as Dantzig presented the simplex algorithm, Lemke [6] proposed the dual simplex algorithm, while other researchers proposed the interior point algorithms [7–9]. A thorough description of the linear programming algorithms would exceed the scope of this paper, therefore, more information can be found in [1,2].

Linear programming problems can nowadays be solved with a plethora of algorithms. However, regardless of the number of algorithms which are available and can solve a specific linear programming problem, it is still difficult to decide upon which algorithm would be the most appropriate to use. Algorithm selection is facilitated through algorithm tuning processes with several interesting studies conducted in this field [10–12]. Researchers and software designers often perform extensive computational experiments to find the default values of parameters that will perform well on most instances of their data. In this concept, modern mathematical solvers have become complex software systems having various parameters that can be efficiently tuned. The configuration process of the solver's parameters is referred to as solver tuning [13] and has been applied successfully in various mathematical solvers [14–19].

A fundamental question that is often raised by researchers is the selection process of the most efficient algorithm in terms of its execution time. In this work, we are looking into this problem exactly. We are concentrating on the execution time, rather than the number of iterations for the solution of an LP problem (as examined for the Exterior Point Simplex Algorithm in [20]), which will be subject to further examination in the future. We are interested in predicting the execution time of a solver for a specific problem instance. By gaining information about the execution time needed to solve an instance, a mathematical solver may devise different options to solve this instance. Similar studies have been conducted in the past, such as [21] which aimed to predict the solution time of Branch-and-Bound algorithms for mixed-integer programs (MIPs) and proposes a double exponential smoothing technique, evaluating it with three MIP solvers.

Meta-learning approaches have been utilized for tuning the performance of algorithms, mainly machine learning ones. The more similar those previous problems are, the better performance we can achieve [22]. Of course, there is no free lunch [23]. When a new problem comes in, leveraging prior experience may not be effective. Apart from tuning the performance of algorithms, meta-learning approaches have been also utilized for predicting their execution time [24,25].

In the field of performance modeling of software systems, analytical modeling and machine learning methods are mostly used [26–29]. Analytical modeling exploits existing knowledge of the internal dynamics of the software system and can express the relationship of the inputs and outputs using a set of analytical equations. However, software systems have become more complex over the years and applying analytical modeling techniques to predict their execution time does not yield good results. Therefore, various researchers explored machine learning techniques to predict the execution time of software systems [30–33].

To the best of our knowledge, a performance modeling tool for any mathematical optimization solver does not exist yet. With this work, we aim to predict the execution time of CPLEX's [34] primal and dual simplex algorithms for solving linear programming problems and we investigate if this tool can be built with regression or classification algorithms. Computational results show that a regression model cannot achieve adequate goodness of fit, while on the other hand, a classification algorithm achieves considerably good accuracy.

CPLEX includes several high-performance linear programming algorithms, supporting also primal and dual variants of the simplex algorithm, and the interior point method. In [35], we proposed a regression model using artificial neural networks (ANNs) for the prediction of the execution time of CPLEX's interior point method on a set of benchmark lin-

ear programming problems (Kennington [36], Mészáros [37], Mittelmann [38], Netlib [39]). Now, we are investigating whether or not a regression model can also be built for CPLEX's primal and dual simplex algorithms. Our computational results show that, with regression, we cannot generate models that achieve goodness of fit for our data and, thus, they cannot stand as accurate prediction models for the execution time of the examined algorithms. An important parameter to consider regarding the difficulty to successfully create prediction models through regressions, for both primal and dual algorithms, is the different time complexities they demonstrate (i.e., worst case, average). Analyzing and describing the algorithms' complexity would exceed the scope of this study, however, this aspect should be further examined in future work. Moreover, regression analysis could prove more effective with the use of several other techniques as well, such as principal component analysis (PCA), so this is also taken into consideration for the next steps of our research. Moving forward with our analysis, we examined our problem further as a classification problem. We attempt to predict the time range under which the execution time will fall, rather than estimate its exact value, and our goal is achieved with the use of classification techniques.

The subsequent section consists of a presentation of our dataset, while a brief description of regression and classification concepts follows next. Moving forward, we first provide an overview of the results we achieved with regression for the prediction of the execution time of the interior point method. Afterwards, we elaborate on the analysis conducted to form regression models, using artificial neural networks, for the prediction of primal and dual simplex algorithms' execution time. A thorough description of the models, which were generated using classification techniques, is provided along with the respective results. Before concluding this study, we include a graphical representation of the comparative analysis among the generated models, which was utilized in order to select the most appropriate model for each algorithm.

2. Materials and Methods

2.1. Dataset

For the purpose of our computational study, 295 benchmark linear programming problems (LPs) were used from the Netlib (25), Kennington (13), Mészáros (217), and Mittelmann (40) libraries. The problems were solved with CPLEX's 12.6.1 [34] primal and dual simplex algorithms. The execution time for their solution, was recorded for each problem. In this study, we examined the following LP characteristics which were set as input to our model. The execution time was set as the output of the model.

- m, n: number of constraints and variables, respectively;
- *nnzA*, *nnzb*: number of nonzero elements of the constraint matrix and right-hand side vector, respectively;
- *rankA*: constraint matrix rank.

Apart from the above problem characteristics, we also examined other characteristics as well, such as the number of variables (after adding slack variables), the problem density, the data length (bit length) that represents integer data, and the constraint matrix norm. However, these characteristics showed no statistically significant contribution to the creation of our models, based on the evaluation metrics we used, that are explained in Section 3.1 below.

Table 1 includes the lower and upper values of the analyzed characteristics for the current set of problems. These values may occur in different problems and are provided only as reference in this paper (i.e., they are not all related to the same problem).

The reason behind the selection of our dataset was our interest in predicting the performance of CPLEX's primal and dual simplex algorithms on well-known LPs, such as the ones described above. Including problems of different natures and structures (such as quadratic or mixed-integer problems) could increase the size of our dataset for training the examined ANNs, however, the diversity of the examined problems would not contribute to meaningful results regarding the problems' solution. In order to ensure that our dataset size is sufficient for training an artificial neural network, we took into consideration commonly

applied rules-of-thumb, such as that the dataset size should be at least a factor of (a) 50 to 1000 times the number of predicted classes [40], and (b) 10 to 100 times the number of the examined features/characteristics [41–45]. The implementations of primal and dual simplex algorithms which have been utilized in the current study are the ones supported in CPLEX Optimizer 12.6.1, as described in previous sections. We executed both algorithms with the default options of CPLEX Optimizer, in order to minimize subjectivity in our observations, which may have resulted from different settings in CPLEX Optimizer. This should be considered as an interesting area of research for the future, especially in combination with solver tuning techniques. Since the hardware and software characteristics are crucial parameters of a specific computing environment, which prove to have a significant impact on the performance of an algorithm [46], all experiments were conducted in the same environment, thus, this factor can be considered the same for all problems, with no fluctuation from one problem to another. More specifically, the computational comparison was performed on a quad-processor Intel Core i7 3.4 GHz with 32 GB of main memory and 8 cores, a clock of 3700 MHz, an L1 code cache of 32 KB per core, an L1 data cache of 32 KB per core, an L2 cache of 256 KB per core, an L3 cache of 8 MB, and a memory bandwidth of 21 GB/s, running on Microsoft Windows 7 64 bit. In case the computational experiments need to be performed under different hardware conditions, a re-training process on the respective ANNs is required.

Table 1. Lower and upper values of examined LP characteristics.

	Minimum	**Maximum**
Constraints	25	986,069
Variables	882	15,000,000
Nonzero elements of constraint matrix	3108	30,000,000
Nonzero elements of right-hand side vector	0	512,209
Rank of the constraint matrix	25	526,121
Primal simplex execution time (s)	0.01	4541.88
Dual simplex execution time (s)	0.01	5184.27

The neural network models presented in this study have been generated and examined with the use of the scikit-learn toolkit [47]. The scikit-learn toolkit integrates a plethora of widely used machine learning techniques, which can be further utilized for inferential statistical data analysis. In contrast to descriptive analysis, which focuses on the attributes of the sampled dataset only, inferential analysis is closer to the concept of a data population, which the observed data derive from. Scikit-learn supports a plethora of supervised and unsupervised learning methods, along with methods for model selection, and evaluation and transformation of data. Extending our previous work [35], as this was performed for the interior point method, we first used the Multi-layer Perceptron (MLP) Regression algorithm (MLPRegressor) for the generation of our models for primal and dual methods. Apart from MLPRegressor, we also experimented with other supervised learning methods and, more specifically, with regression (such as ElasticNet, Ridge, Random Forest, etc.) and classification methods (such as MLPClassifier, KNeighborsClassifier) available in scikit-learn. The results of the complete process of our analysis are presented in the next sections of this paper.

2.2. Regression vs. Classification

Regression and classification belong to the broader family of supervised machine learning techniques. Both regression and classification apply the same concept of using known datasets (i.e., training datasets) to make predictions about new incoming data. Considering that an input variable x and an output variable y are available, a supervised learning algorithm aims to "teach" a mapping function (that is, $y = f(x)$) from the input variable x to the output variable y. This way, whenever there are new input data x, the

respective output variable y will be able to be predicted, with the help of regression or classification predictive models. Although these techniques share the same objective, regression and classification have a main difference, which is that the output variable for classification is categorical (or discrete), while in regression it is numerical (or continuous). Classification predicts a discrete class label, while regression, a continuous quantity. There are some algorithms, though, which can be used both for classification and regression, with only slight modifications, such as artificial neural networks (ANNs) and decision trees. Classification predictive models can be evaluated using the accuracy value, whereas regression predictive models are evaluated through the respective coefficient of determination and the root mean squared error (quantities that cannot be measured for classification predictions). In this study, we used several regressors, which are mentioned specifically in the following sections, however, as analyzed further below, the respective models could not be used as accurate prediction models of the execution time. More information on regression, classification, and artificial neural networks can be found in [44,48–51].

MLPRegressor belongs to the family of supervised learning algorithms and it can learn a function $f(x) : R^x \rightarrow R^o$ by training on a dataset, where the number of dimensions for input is denoted by x, while the number of dimensions for output is denoted by o. The parameters of MLPRegressor are further described in this section in order to better understand its functionality. One of the most important parameters is the number of hidden layers that needs to be examined and defined, along with their size and the activation function which we have to choose. The activation is responsible for converting the 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*). Numerous combinations of all supported activation functions with different numbers of hidden layers were tested in this study. Since we are working with ANNs, it is significant to test the solver of our model. Neural networks consist of a number of simple but highly interconnected nodes, the so-called "neurons", which are organized in layers. Neural networks are extremely helpful in finding patterns that are too complex to be manually extracted and taught for any kind of machine. In an ANN, the input layer (which has one neuron for each element of the input data) communicates to one or more hidden layers that are present in the network. The hidden layers are actually the place where all the processing of the information takes place, thus their name may not be so representative of their real significance; they are characterized as "hidden" only because they do not constitute the input or the output layer. The information is processed through weights and biases (commonly referred to as W and b). In more detail, once the input is received, the neuron calculates a weighted sum (by also adding the bias) and according to the result and the preset activation function, it is activated or not. The neuron transfers this information to its connected neurons, ending up at the last hidden layer that is linked to the output layer, which has only one neuron for the respective output. The solver of the model is related to the weight optimization process that takes place while transmitting information through hidden layers. There are several solvers that can be used, such *lbfgs*, an optimizer in the family of quasi-Newton methods, and *sgd*, concerning stochastic gradient descent. For small datasets, *lbfgs* has proven to converge faster and perform better in general, as explained in [47]. Solver *lbfgs* works by using a weighted linear summation which transforms the input values of previous layers to output values for the next layer. One other parameter is the tolerance value, which 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 parameter (Ridge regression; regularization technique used to address over-fitting and feature selection). The solver will iterate until convergence (defined by the tolerance value) or the maximum number of iterations. Last but not least, one more aspect that needs to be taken into consideration is the scale of our data, since the parameters which are given as input to a model may be of different scale. Scaling and normalizing the original data

were significant steps we took upon generating our models to minimize any difficulty for the examined problems to be modeled.

3. Results and Discussion

3.1. Regression Evaluation Metrics

A simple regression model is the linear regression model, which implies that there is a linear relationship between the dependent and independent variable. This linear relationship is represented by a line, i.e., the regression line, which is found to be closer to the data points than other lines, according to a specific mathematical criterion, and can be calculated with the least squares method [51,52]. The distinctive feature of the least squares regression line is the vertical distance between the data points and the regression line, which is the smallest possible. The least squares method, and thus the regression line, are named as such because the best line of fit is the one that minimizes the sum of squares of the errors (i.e., variance). This may be difficult to visualize, however, and the main purpose is to find the equation that fits the data points as closely as possible. A simple linear regression model is represented by Equation (1) below:

$$Y_i = (b_0 + b_1 X_i) + \varepsilon_i \tag{1}$$

where Y_i is the dependent variable, b_0 represents the intercept with the vertical axis, b_1 is the slope of the regression line, and X_i is the independent variable. The value of ε_i represents the amount of residual. Generally, the residual value is calculated as the difference between the observed value and the estimated value of the regression model. Small residuals correspond to a good fit of the regression model, while the opposite implies that the model does not fit well to the examined data. The entities b_0 and b_1 are characterized as "regression coefficients" and are necessary for the least squares method, since we need to identify their values and, thus, the regression line, so that the following quantity (2) is minimized.

$$1\sum_i \varepsilon_i^2 = \sum_i (Y_i - b_0 - b_1 X_i)^2 \tag{2}$$

The following amounts of total sum of squares, residual sum of squares, and model sum of squares ((3)–(5), respectively) contribute to the evaluation of good fit of the regression line to the examined data. Sum of squares (SS) indicates the deviation from the mean and is calculated as the sum of the squares of the differences from the mean [49].

$$1SS_T = \sum_i (Y_i - \overline{Y})^2 \tag{3}$$

$$1SS_R = \sum_i (Y_i - b_0 - b_1 X_i)^2 \tag{4}$$

$$1SS_M = SS_T - SS_R. \tag{5}$$

A rather useful and simple interpretation of the sums of squares would be that SS_T and SS_R represent the deviation of the examined data from the "worst model" (mean value) and the "best model" (line), respectively, while SS_M denotes the difference between the "worst model" and the "best model". The bigger the value of SS_M, the more important the contribution of the model to the prediction of the independent variable Y. The smaller the value of SS_M, the lower the contribution of the model to the improvement of the "worst prediction" of the mean value. The quality of the model fitting can be calculated as the percentage of the improvement in prediction (6) which is introduced by the model. This implies the percentage of the independent variable's volatility, which is explained by the model and is named "coefficient of determination", corresponding to the square of Pearson's coefficient [53,54].

$$1R^2 = \frac{SS_M}{SS_T} = \frac{SS_T - SS_R}{SS_T} = 1 - \frac{SS_R}{SS_T}. \tag{6}$$

R-squared (*R-Sq*) or R^2 defines the good fit of a statistical model to the examined data. The higher its value, the better fit the model has. Although the significance of this coefficient is crucial for all regression models, we should always take into consideration its two main drawbacks. It has been reported that with the addition of one or more new parameter(s) in a model, the respective R^2 value increases. This fact explains why taking only the R-squared value into account can not secure the good fit of a model. This is one of the reasons why the R-squared value alone cannot guarantee the good fit of a model. Moreover, the metric may be affected by random noise of the dataset, especially when there is a high amount of parameters and higher order polynomials in the examined model. In such cases, we have an "over-fitting" problem, which results to misleadingly high R^2 values and makes the model unsuitable for prediction purposes [55]. An additional measure for evaluation of the regression model is the *F*-test, which is calculated by the mean sums of squares as shown in Equations (7)–(9) below. The mean squares (*MS*) amount is calculated by dividing the respective sum of squares by the degrees of freedom. It estimates the population variance. The mean squares are useful for defining whether the parameters of a regression model are significant [49].

$$1MS_M = \frac{SS_M}{Degrees\ of\ Freedom} = \frac{SS_M}{Number\ of\ Variables} \tag{7}$$

$$1MS_R = \frac{SS_R}{Degrees\ of\ Freedom} = \frac{SS_R}{n - Number\ of\ Regression\ Coefficients} \tag{8}$$

$$1F = \frac{MS_M}{MS_R}. \tag{9}$$

Degrees of freedom is the number of values in the final calculation of a statistic that are free to vary. The concept of this metric was introduced by Student in 1908 [56], while the specific naming belongs to Fisher, who used it some years later in 1922 [57]. Although R^2 estimates the relationship strength between a regression model and the dependent variable, it does not support any formal hypothesis test for this relationship. This is the reason for the significance of *F* and the corresponding *P* values. The *F*-test determines whether this relationship is statistically significant or not. More specifically, if the *P*-value for the *F*-test is lower than the defined significance level, the regression model is statistically significant for predictive purposes [48]. In the case the statistical significance is <0.001, we can safely conclude that the model highly contributes to the prediction of the independent variable.

Moreover, the statistical significance of the regression coefficients is crucial for the validity of the regression model and the evaluation of its quality. More specifically, the value of b_0 defines the change upon the dependent variable if the respective independent variable changes by one unit. To examine the statistical significance of b_1 we apply a *t*-test with significance < 0.05 [56]. The standard error of the coefficient (*SE Coef*) is the standard deviation of the estimate of a coefficient in a regression model. It measures the precision of the model's estimation about the coefficient's unknown value. The *SE Coef* value is always positive and the smaller it is, the more precise the estimate. The division of the coefficient by the respective standard error results in a specific t-value (*T*), which is also known as the t-statistic. It measures the likelihood that the actual value of the parameter is not zero. The larger the absolute value of *T* is, then the less possible it is for the real value of the parameter to be zero. If (*P*), that is related to the t-statistic, is lower than the defined significance level, we conclude that the coefficient is different from zero [48]. An introduction to multiple regression would be the extension of the linear model with more than one independent variable (10). In the case of two independent variables, the regression line's equation extends to a plane, while in the case of more than two independent variables, to a hyperplane.

$$1Y_i = (b_0 + b_1X_i + ... + b_kX_k) + \varepsilon_i. \tag{10}$$

In multiple regression, the amounts of SS_T, SS_R, and SS_M are calculated in a more complicated way but their meaning and significance remain the same. The fact that multiple independent variables are involved in the regression makes it imperative to calculate a coefficient of multiple correlation that reveals the strength of the relationship of the dependent variable with all independent ones. The value of R^2 is calculated similarly to the simple linear regression, as the volatility percentage of the independent variable, which is actually explained by the model. One fundamental issue that needs to be clarified before initiating the creation of a multiple regression model is how the independent variables will be selected. Taking into consideration that, during examination of a specific dataset, we can use particular attributes and features as independent variables, it is clear that the latter are usually correlated to each other. However, there are several methodologies for the selection of the most appropriate variables for the regression model, such as forced entry, when all variables enter the model simultaneously, stepwise, where the order of variables is defined by mathematical criteria, forward, backward, etc. [58]. In general, the researcher should have a good understanding of the dataset that needs to be examined, so that the most appropriate methodology can be selected. Another matter that concerns researchers is the model's accuracy, since it is crucial that the model can achieve a good fit to the data and its behavior is not affected by a few extreme instances. Such instances, called "outliers", are cases which differ significantly from the rest of the dataset. They can stand as a "diagnostics" measure of the model's fitting, since they may have a great impact on the regression coefficients' values. Outliers can be detected by their large residual values. For better comprehension and comparison of residuals, these can be standardized by dividing their value by their standard deviation. Standardized residuals with an absolute value > 3 may be concerning, while in the case over 1% or 5% of the standardized residuals are >2.5 or >2, respectively, then this is a indication of poor fitting. Other measures of checking for outliers are the adjusted predicted value, which is calculated for each case separately, by removing it from the sample and estimating it with the new regression model that is formed, Cook's distance, which is a measure of overall impact of a data point on the model (e.g., data points with a value > 1 may be concerning), etc. [59,60]. Detailed explanation of additional metrics which are examined for the selection of best-fitting regression models is provided below:

1. Adjusted R-squared ($R\text{-}Sq(adj)$): adjusted coefficient of determination. This metric proves to be useful during the comparison of models with different numbers of predictors (i.e., independent variables), since it is adjusted according to the number of predictors in a model. In more detail, its value increases only if a new predictor improves the model more than was anticipated by chance, while it decreases when a predictor improves the model less than anticipated by chance. Interestingly enough, its value turns out to always be less than the R-squared value [55].
2. Predicted R-squared ($R\text{-}Sq(pred)$): the predicted R-squared explains the predictability of a regression model, i.e., predicting responses for new observations. A regression model that seems to fit the original data may not be capable of providing valid predictions for new observations. Similarly to adjusted R-squared, predicted R-squared is always lower than R-squared and there are times when even a negative value has been observed. Perhaps the most important benefit of this metric is that it can "prevent" researchers from using models which over-fit. Since it is impossible to predict random noise, the value of predicted R-squared would drop in case of an over-fitted model. Kutner et al. explain that if $R\text{-}Sq(pred)$ is much lower than the model's $R\text{-}Sq$ value, the model is probably over-fitted [49].
3. Standard error of regression (S): standard error of regression measures the units of the "response" (dependent variable) and stands for the standard distance between data values and the estimated regression line. As the S value decreases, the model's predictability increases. When comparing different models, the model with the lowest S value reflects the best fit [61].

One more metric is root mean square error (RMSE), which is the standard deviation of the residuals (prediction errors). RMSE measures the residuals' spread around the line of best fit. It is a measure of accuracy used for comparison of different models, which are generated for a specific dataset. This metric has nonnegative values, while a value of 0 would indicate a perfect fit to the data, however, this is quite impossible to achieve in practice. In general, the lower the RMSE value is, the better. It is important to note that this metric should not be used between different datasets, as it is scale-dependent [62] and comparisons of different data types would, thus, not be valid. Apart from these metrics, we also measured the mean absolute error (MAE), which measures the average magnitude of the errors in a set of predictions, without considering their direction, and the median absolute error (MedAE) that is insensitive to outliers [48,49,55].

3.2. Regression

During our study on the interior point method [35], MLPRegressor was applied, resulting in the selection of the most appropriate model to predict the execution time of the algorithm. The prediction model for the interior point method achieved an R^2 value of 78% (training set) and 72% (test set). The percentage in both sets (training and test), along with the values in the rest metrics considered for the model evaluation, proved that the regression model for the interior point method has a very good fit on the data and, thus, can further be used for prediction of the algorithm's efficiency. The training and test sets were formed through cross validation, as supported for MLPRegressor by the scikit-learn library [47].

Table 2 shows the results of the regression model for the interior point method execution time, using MLPRegressor ANN, while Figures 1 and 2 show some examples of the comparative analysis we performed before deciding on the selected model. More details on the complete analysis can be found in [35].

Based on the above results, we decided to move on with our effort to generate respective models for the primal and dual algorithms as well. Evaluating the metrics of each model for the primal and dual simplex algorithm separately, our models were formed, using the parameters shown in Table 3. To split our dataset into training and test sets, a ratio of 75 to 25 was selected and the formulated models were evaluated on the standard metrics, which were analyzed earlier in this section.

Table 2. Metrics for MLPRegressor model for the execution time of interior point method.

	Training Set	**Test Set**
RMSE	123.32	296.73
MAE	54.31	97.54
MedAE	7.12	9.49
R^2	0.78	0.72

Table 4 presents the results of our neural networks, both for primal and dual simplex algorithms on the examined dataset. For the primal simplex algorithm, the model that showed the best performance, compared to the rest of the models that we formed and tested, achieved an RMSE value of 342.08 and an R^2 value of 0.79, while for the test set the model achieved an RMSE value of 1302.25 and an R^2 value of 0.21. The model was set to work with one hidden layer of size equal to 30 neurons, logistic activation function, and the *lbfgs* solver. As for the dual simplex algorithm, the results we achieved with the best fitting model were an RMSE value of 345.28 and an R^2 value of 0.66 in the training set, while in the test set, these values reached 1260.39 and 0.05, respectively. In this case, there was again one hidden layer with 20 neurons, while the activation function and solver were logistic and *lbfgs*, similarly to the model formulated for the primal simplex algorithm. Taking into account the variability in the features of the 295 LPs of our dataset and the metrics' values, our models performed below our initial expectations since it was shown that they cannot explain the data reasonably well, showing a significant discrepancy between the metrics'

values of the training and the test set. An R^2 value of 1 would indicate a perfect fit of the data, so the current R^2 values of the training sets for both algorithms prove a certain level of goodness of fit of our models, which, however, cannot be verified/validated further. This is confirmed by the R^2 values of the respective test sets, which drop significantly, while the corresponding RMSE values increase tremendously, compared to the ones of the training set. At this point, it is interesting to keep in mind what Hillier and Lieberman defined in [63], i.e., relationships between primal and their respective dual problems must be symmetric, since "the dual of this dual problem is this primal problem". Therefore, in the future, we are interested in examining the corresponding dual version of each LP, to identify whether there is any impact on the results we achieved through regression.

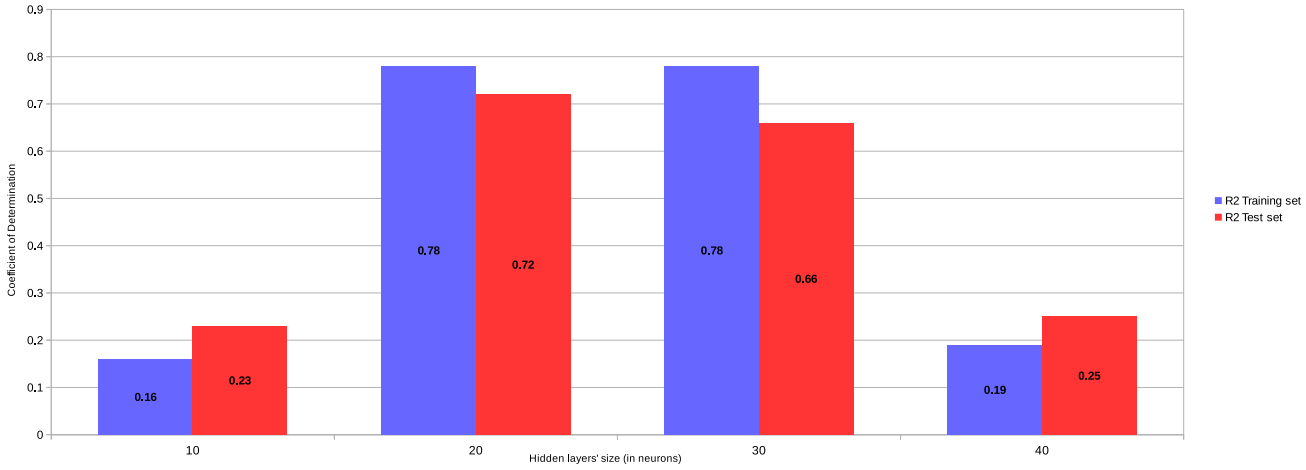

Figure 1. Regression model for interior point method—tuning the number of neurons (1 hidden layer).

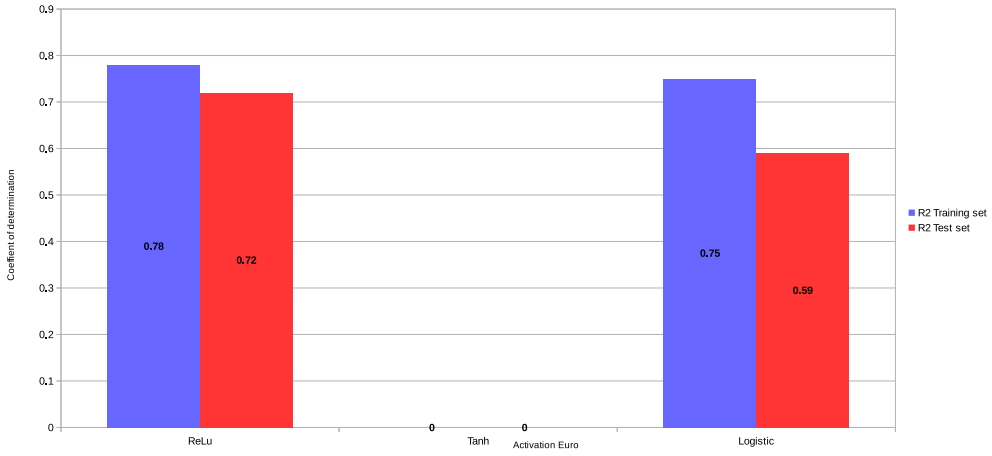

Figure 2. Regression model for interior point method—tuning the activation function.

Table 3. MLPRegressor model parameters used for primal and dual simplex algorithms.

Algorithm	Primal, Dual
Hidden layers	1–3
Hidden layer sizes	10–100 neurons/layer
Activation function	relu, tanh, logistic
Solver	lbfgs, sgd
Alpha value	1×10^{-5}
Maximum iterations	1000
Tolerance	0.0001

Table 4. MLPRegressor model for the execution time of the primal and dual simplex algorithms.

	Primal		Dual	
	Training Set	**Test Set**	**Training Set**	**Test Set**
RMSE	342.08	1302.25	345.28	1260.39
MAE	9.60	25.07	11.35	25.16
MedAE	3.26	14.75	3.93	16.05
R^2	0.79	0.21	0.66	0.05

These values reveal models that cannot be further utilized for prediction of the execution time needed for the solution of LPs by the primal and dual simplex algorithms. It is quite interesting to show that these results were considered the "best" after extensive and thorough testing, with different numbers of hidden layers and neurons per layer (1–3 and 10–100, respectively), different activation functions (*relu*, *tanh*, and *logistic*), and solvers (*lbfgs* and *sgd*). A graphical representation of the results we received with only some of the different models formed with MLPRegressor follows below. The models we tested showed worse performance with some of them even characterized by negative values of R^2 for the test set, which could not be interpreted to lead to meaningful and useful results. More specifically, since R^2 compares the fit of the chosen model with that of a horizontal straight line (the null hypothesis), if the model fits worse than a horizontal line, then R^2 is negative, meaning that the chosen model does not follow the trend of the data, so would not be useful for prediction purposes. Moreover, models formed with the *sgd* solver showed poor goodness of fit (R^2 values below 0.06 and 0.20 for the primal and dual simplex algorithms, respectively), thus they are not included in the graphical representation. A few of our models are presented in the respective graphs in Figures 3 and 4. The example of Figure 3 presents the R^2 value for several different numbers of neurons in models for the primal simplex algorithm, using one hidden layer, the logistic activation function, and the *lbfgs* solver. The R^2 value is given both for the training and test sets. In Figure 4, the R^2 value of several models is given, using the *relu*, *tanh*, and *logistic* activation functions and having 30 neurons in one hidden layer with *lbfgs* as the solver. Furthermore, similar results are presented for the dual simplex algorithm in Figures 5 and 6. Figure 5 presents the R^2 value for several different numbers of neurons in models for the dual simplex algorithm, using one hidden layer, the logistic activation function, and the *lbfgs* solver. In Figure 6, the R^2 value of several models is given, using the *relu*, *tanh*, and *logistic* activation functions and having 20 neurons in one hidden layer with *lbfgs* as the solver. The R^2 value is given both for the training and test sets below, for all presented samples of our comparative analysis.

Elaborating further on the concept of regression, we extended our analysis to more regression algorithms, such as Decision Tree, ElasticNet, Lasso, Random Forest, Ridge, Support Vector, and Linear Regressor. Although scikit's GridSearch function was utilized to identify the best regression model generated from each algorithm, the models that were eventually formulated could not be used for prediction purposes. A remarkable exception was reported for the Random Forest ANN model, which may result in better values of the evaluation metrics than MLPRegressor, but shows the same significant discrepancies between the training and test set. More specifically, the reported R^2 values of the training set for primal and dual simplex algorithms show goodness of fit for the relevant Random Forest ANN models, in combination with the values of the rest metrics, as well. However, this fact cannot be validated through the test set, since the R^2 values decrease significantly, along with the remaining error metrics' values which increase to a large extent, compared to the ones of the training set. The values of the metrics which were used to evaluate the respective ANN regression models for primal and dual simplex algorithms are included in Tables 5 and 6, separated for the training and test set.

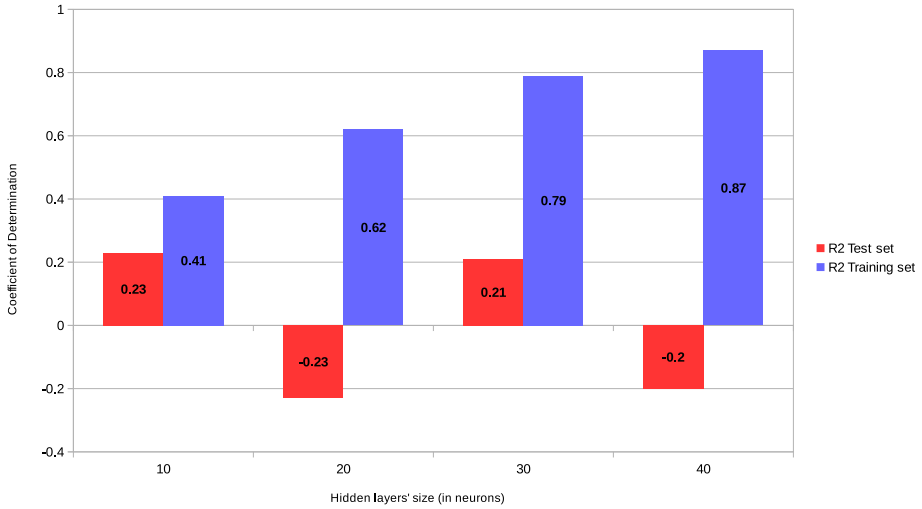

Figure 3. Regression model for primal method—tuning the number of neurons in hidden layers.

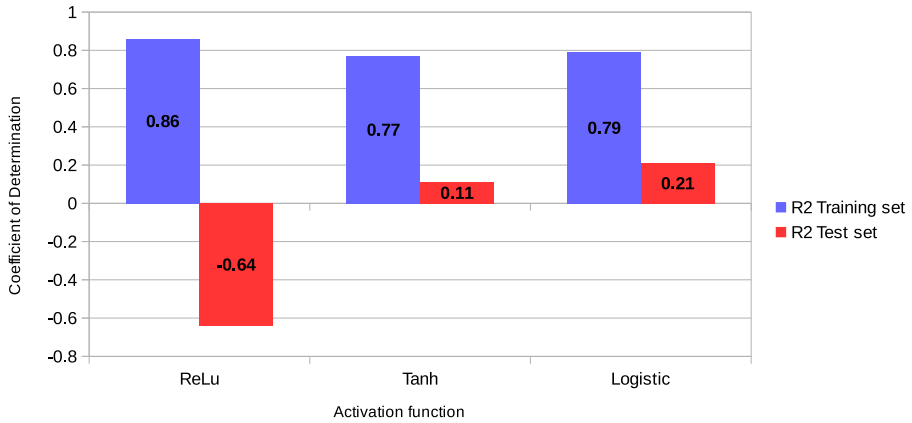

Figure 4. Regression model for primal method—tuning the activation function.

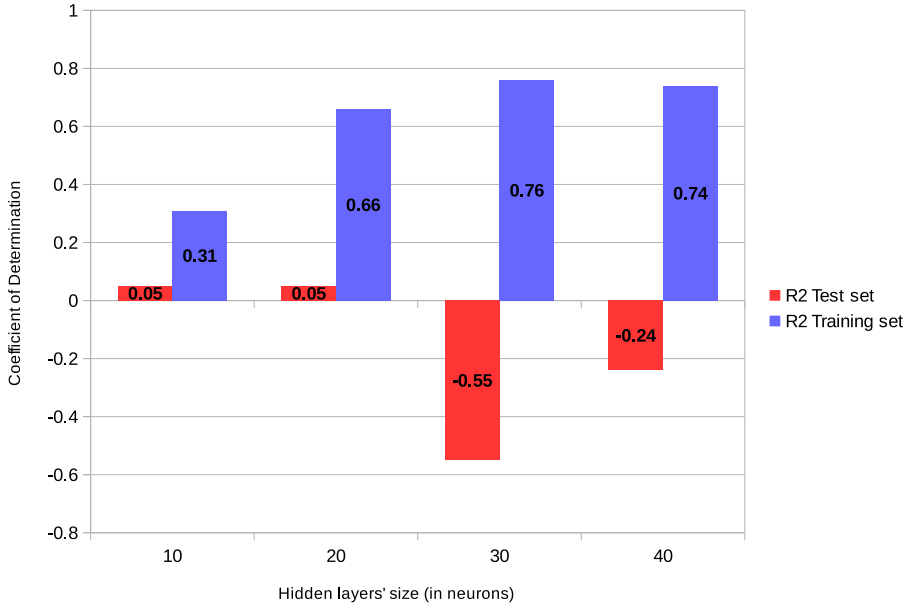

Figure 5. Regression model for dual method—tuning the number of neurons in hidden layers.

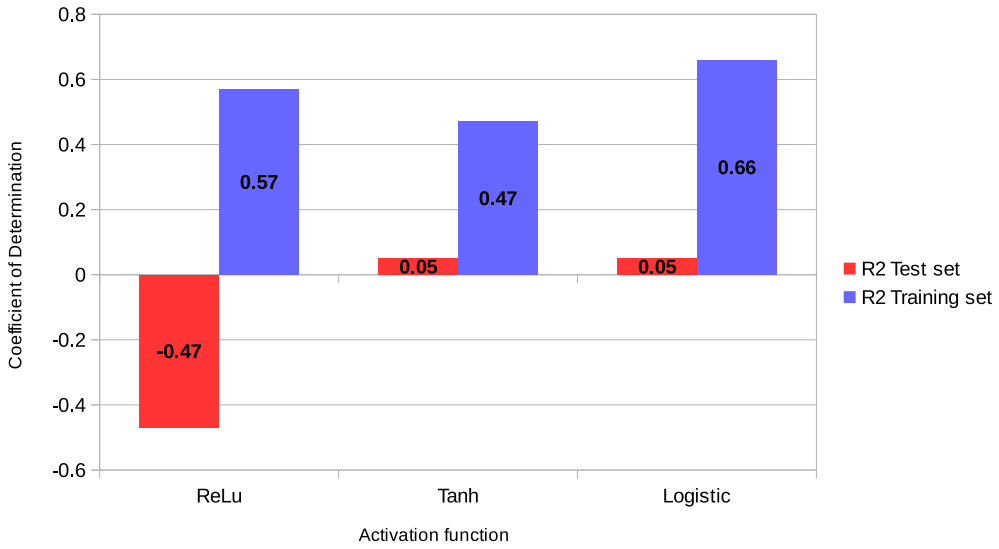

Figure 6. Regression model for dual method—tuning the activation function.

Table 5. Other regression models for the execution time of the primal simplex algorithm.

	Training Set				Test Set			
	RMSE	MAE	MedAE	R^2	RMSE	MAE	MedAE	R^2
Decision Tree	1441.9	31.77	26.02	0.01	2098.2	37.06	28.00	−0.03
ElasticNet	1483.2	32.11	24.28	0.07	1717.2	35.04	27.27	−0.01
Lasso	1470.6	32.24	35.39	0.05	1797.1	35.17	24.92	−0.002
Linear	1669.9	34.95	27.82	0.05	1177.7	29.90	24.39	−0.03
Random Forest	209.4	10.28	7.57	0.88	954.20	22.52	17.53	0.15
Ridge	1491.6	32.10	24.50	0.04	1809.10	36.41	28.00	−0.01
Support Vector	1663.0	27.55	17.89	−0.08	2138.28	31.92	18.46	−0.14

Table 6. Other regression models for the execution time of the dual simplex algorithm.

	Training Set				Test Set			
	RMSE	MAE	MedAE	R^2	RMSE	MAE	MedAE	R^2
Decision Tree	1125.2	28.02	23.90	0.02	998.6	26.98	25.90	−0.08
ElasticNet	1014.7	26.16	20.90	0.03	1206.8	27.90	20.76	0.04
Lasso	940.2	24.01	18.77	0.09	1327.0	28.93	20.92	−0.08
Linear	1040.6	25.87	17.59	0.10	1006.7	25.99	22.91	−0.10
Random Forest	148.4	8.68	6.77	0.87	618.9	17.29	13.12	0.34
Ridge	1033.3	26.21	19.78	0.11	997.9	24.84	18.60	−0.10
Support Vector	1299.2	21.26	8.06	−0.17	1356.96	24.15	12.20	−0.30

These findings enhanced our focus towards classification techniques in order to experiment with our dataset further. Now, instead of trying to predict an exact value, such as the execution time of an algorithm, we will concentrate on predicting the class under which the value of the execution time may fall. As shown in the following sections, classification techniques seem to work fine for our dataset.

3.3. Classification

Having tested regression models thoroughly, we concluded that keeping the models that showed the best performance as prediction models for the execution time of the primal and dual simplex algorithm would not be a safe or valid choice. Therefore, we decided to experiment with classification techniques and identify models that could be used for the prediction of certain classes of instances. For the purpose of this study, we tested two of the most commonly used classification algorithms supported by the scikit-learn toolkit,

such as Multi-layer Perceptron Classifier (MLPClassifier) and KNeighborsClassifier. In contrast to other classification algorithms, such as Naive Bayes Classifier, MLPClassifier performs the task of classification, based on an underlying neural network. The process of classification using ANNs may seem theoretically complex and difficult to implement and interpret and it surely requires extensive testing to tune (offering a plethora of tuning options to prevent over- or under-fitting, though). However, this still cannot change the fact that it can prove to be a powerful tool for dealing with complex relations and functions that connect the examined input and output variables, while it is also effective for high dimensionality problems. The parameters described earlier for MLPRegressor are also present in the use of MLPClassifier. Therefore, we proceeded with exhaustive testing of several models using different numbers of hidden layers and numbers of neurons, different activation functions, and solvers. The exact ranges of values are presented above in Table 3. To measure the validity and accuracy of the models, which were generated by MLPClassifier and the remaining classifiers, we analyzed the confusion matrix and the accuracy value of each model, along with the classification report that is created upon testing of the model. In statistical classification and machine learning, a confusion matrix supports the visualization of a supervised learning algorithm's performance, by showing the instances in a predicted class in each row of the matrix and the instances in an actual class in each column of the matrix (or vice versa). The confusion matrix can show how many instances were misclassified for each class. Accuracy is another significant metric for evaluating classification models, which, in general, can be considered as the number of predictions that the examined model identified correctly. Detailed explanation of the concepts of confusion matrix, accuracy, etc. can be found in [64]. More specifically, accuracy could be represented by the following definition:

$$\text{Accuracy} = (\text{Total number of correct predictions}/\text{Total number of predictions}). \quad (11)$$

We could explain the concept of accuracy in classification problems with the help of some simple but useful terms such as true and false positives and true and false negatives. A true positive (TP) is an instance that exists in an actual class of our dataset and has also been correctly predicted by our examined model. A true negative (TN) is an instance that does not exist in the actual class of our dataset and it is also correctly predicted by our examined model. A false positive (FP) is an instance that does not exist in the class, but our model has predicted it incorrectly, while a false negative (FN) refers to an instance that exists in a class of the examined dataset, however, it is incorrectly predicted (i.e, that it does not exist). As a fraction, accuracy could be expressed as follows:

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}). \quad (12)$$

However, it would not be safe to consider that the confusion matrix and accuracy value can stand alone as proof of the validity and good performance of the examined models, thus we proceeded with further analysis of the generated classification reports. A classification report includes the precision, recall, F_1, and support scores of a classification model. Compared to a plain accuracy value, we could say that the classification report offers a deeper intuition of the classifier's behavior and can also help select the most effective model for the examined dataset (for instance, the model with the "strongest" values of classification metrics). Before presenting the results of our models in this section, we include a brief description of the metrics we used to compare our results. The precision value is representative of the classifier's ability to avoid marking a negative instance as positive. For each class of a given dataset, the precision is defined as the ratio of TPs to the sum of TPs and FPs, as shown below:

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}). \quad (13)$$

Moving on, we examine the recall value which depicts the classifier's ability to find all positive instances. For each class of the examined dataset, recall is calculated by the ratio of TPs to the sum of TPs and FNs, as shown below:

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}). \quad (14)$$

Furthermore, the F_1 score is the harmonic mean of precision and recall, with its best value reaching 1 (i.e., perfect precision and recall) and its worst at 0. Although F_1 cannot be used alone to describe the accuracy of a classification model, it can certainly be useful while comparing several models. Last, but not least, the support value stands for the number of actual instances of each class in the examined dataset. This value provides us with a clear picture of the "balance" in our dataset, meaning how balanced the separation of the instances among the classes of our dataset is. Unbalanced training data may result in weaknesses in the reported scores of the classifier, which would result in the need for stratified sampling or even re-balancing [64].

3.4. Classification Model for the Primal and Dual Simplex Algorithms

This subsection includes the classification models for the primal and dual simplex algorithms, along with examples from our comparative analysis to select the most efficient models for the examined dataset. The selected model for the primal simplex algorithm uses the *tanh* activation function, *lbfgs* solver, and two hidden layers of 100 neurons each. Similarly, the selected model for the dual simplex algorithm uses the *lbfgs* solver and two hidden layers of 100 neurons each, with the only exception being the activation function, which is *relu* instead of *tanh*. The execution time of primal and dual simplex algorithms was separated into four classes, which are defined as shown in Table 7. The classes were formulated after extensive sampling of the given dataset and experimenting with different numbers of classes. Class 0 represents LPs that are easy to solve, with the time needed for their solution being less than 0.1 s, for both algorithms. Class 1 consists of LPs that are relatively easy to solve, with the execution time falling in ranges 0.1–0.5 and 0.1–1 s, for primal and dual simplex algorithms, respectively. Class 2 stands for the LPs that seem to require more time to solve (i.e., execution time for primal and dual simplex reported in ranges of 0.5–4 and 1–10 s, respectively). Finally, Class 3 consists of LPs that can be considered rather difficult and time-consuming, with the relevant execution time exceeding 4 and 10 seconds for primal and dual simplex algorithms, respectively. The generated classification model for the execution time of the primal simplex algorithm reaches an accuracy value of 0.83, while the generated classification model for the execution time of the dual simplex algorithm has an accuracy value of 0.84. The respective confusion matrices and classification reports are available in Tables 8–10. It is shown that the model for the primal simplex algorithm misclassifies only 2 instances in Class 0, 3 instances in Class 1 and Class 2, while 7 instances are misclassified in Class 3. The model for the dual simplex algorithm classifies all 33 instances correctly in Class 0, misclassifies 4 instances in Class 1 and 6 instances in Class 2, while Class 3 turns out to be the most challenging one with 4 out of a total of 11 instances misclassified. The precision, recall, and F_1 scores for each examined class are quite satisfying, with the average scores confirming the accuracy of the generated models.

Table 7. Classes of the primal and dual simplex algorithm execution time (in seconds).

Class	Primal	Dual
0	$0 < \text{time} < 0.1$	$0 < \text{time} < 0.1$
1	$0.1 \leq \text{time} < 0.5$	$0.1 \leq \text{time} < 1$
2	$0.5 \leq \text{time} < 4$	$1 \leq \text{time} < 10$
3	$4 \leq \text{time}$	$10 \leq \text{time}$

Table 8. Confusion matrix for the primal simplex algorithm execution time.

		Actual Class			
		0	**1**	**2**	**3**
Predicted Class	0	22	2	0	0
	1	2	21	0	1
	2	0	0	20	3
	3	0	0	7	11

Table 9. Confusion matrix for the dual simplex algorithm execution time.

		Actual Class			
		0	**1**	**2**	**3**
Predicted Class	0	33	0	0	0
	1	4	20	0	0
	2	0	3	15	3
	3	0	2	2	7

Table 10. Classification reports for the primal and dual simplex algorithm execution time.

Class (Primal)	**Precision**	**Recall**	F_1	**Support**
0	0.92	0.92	0.92	24
1	0.91	0.88	0.89	24
2	0.74	0.87	0.80	23
3	0.73	0.61	0.67	18
avg/total	0.83	0.83	0.83	89
Class (Dual)	**Precision**	**Recall**	F_1	**Support**
0	0.89	1.00	0.94	33
1	0.80	0.83	0.82	24
2	0.88	0.71	0.79	21
3	0.70	0.64	0.67	11
avg/total	0.84	0.84	0.84	89

These results were extracted after extensive testing with several combinations of activation functions, solvers, different numbers of hidden layers and neurons, and different classification algorithms (e.g., KNeighborsClassifier). The following figures (Figures 7–12) include a graphical representation of examples from various tests which were performed and used for comparative analysis, before we reach the final models of this study. As shown, although we have an accuracy score of 0.88 in Figure 10, the respective model is not selected. The reason for this is that the rest of its characteristics (precision, recall, F_1) indicate a unsuitable model for our dataset, i.e., precision and F_1 are ill-defined and, thus, they are set to 0.0 in labels with no predicted samples.

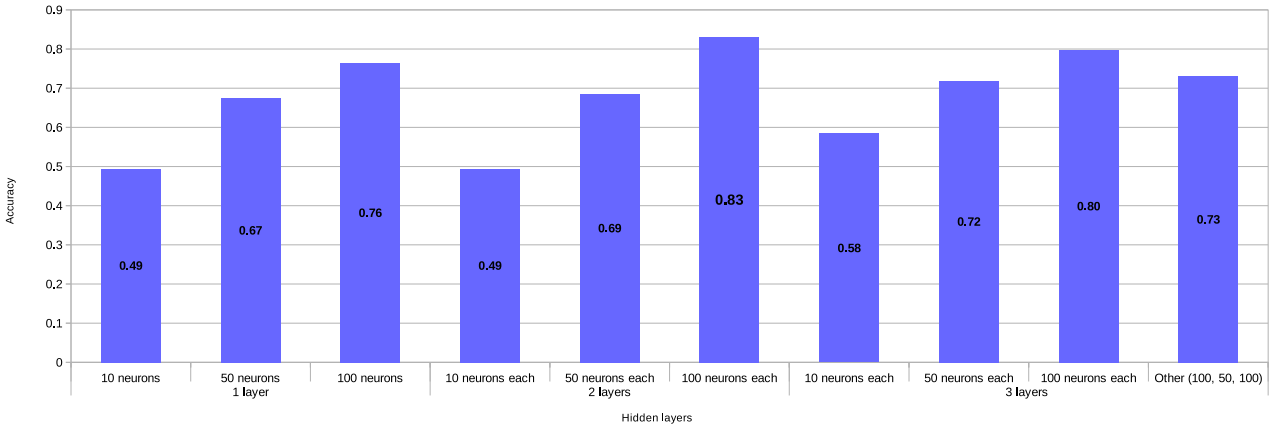

Figure 7. Classification model for primal method—tuning the number of hidden layers and neurons (*tanh* activation function, *lbfgs* solver).

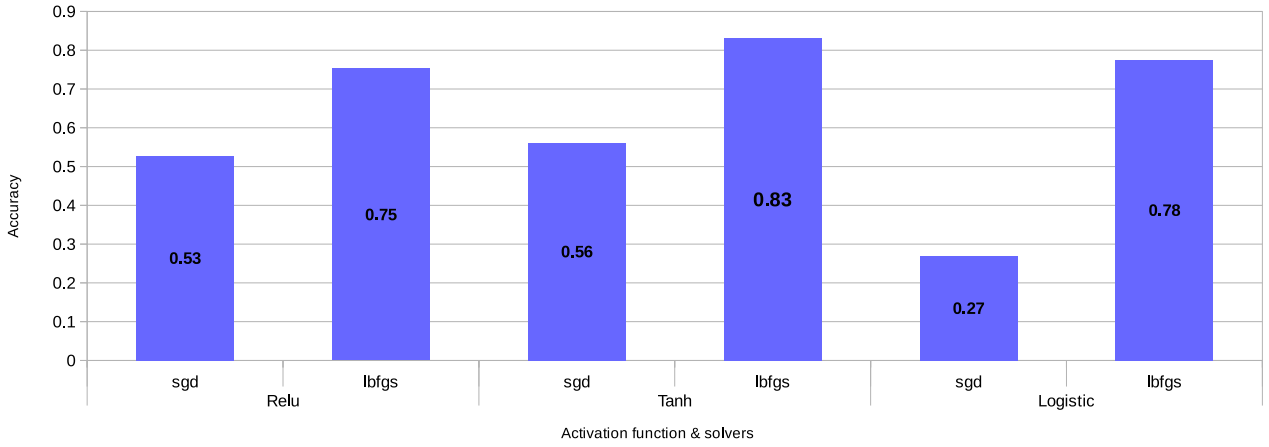

Figure 8. Classification model for primal method—tuning the activation function and solver (2 hidden layers, 100 neurons each).

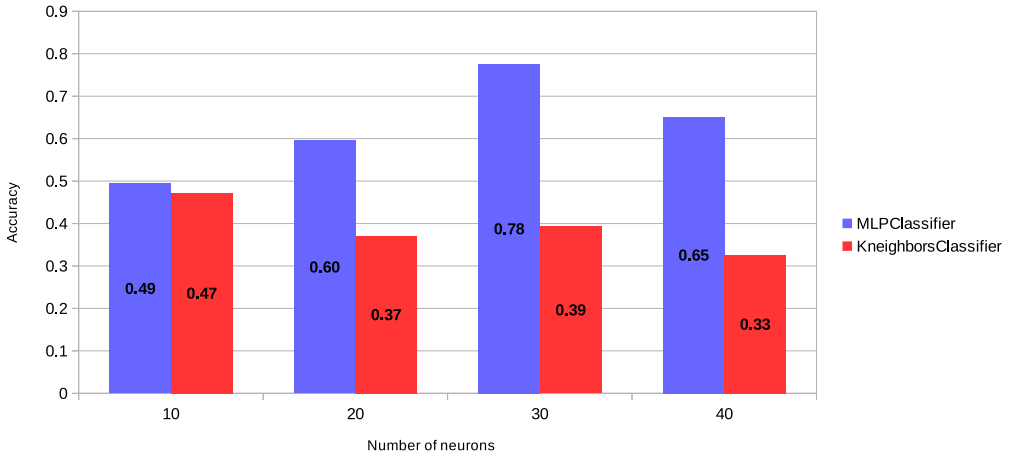

Figure 9. Classification model for primal method—testing different classification algorithms (1 hidden layer, *tanh* activation function, *lbfgs* solver).

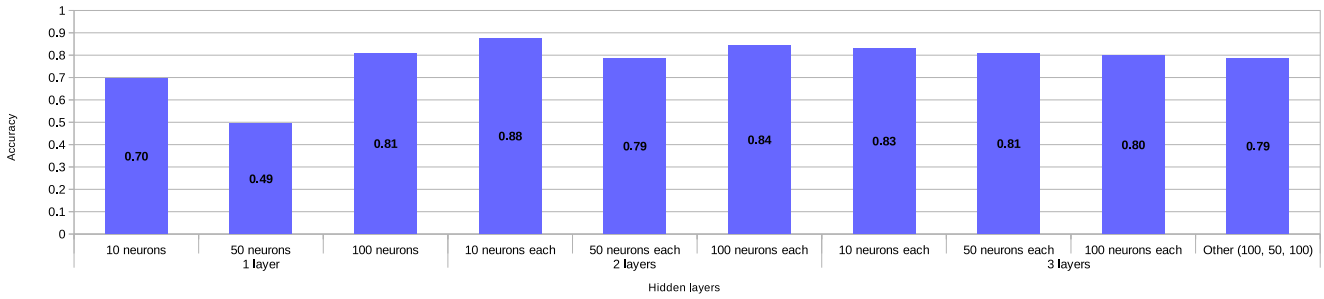

Figure 10. Classification model for dual method—tuning the number of hidden layers and neurons (*relu* activation function, *lbfgs* solver).

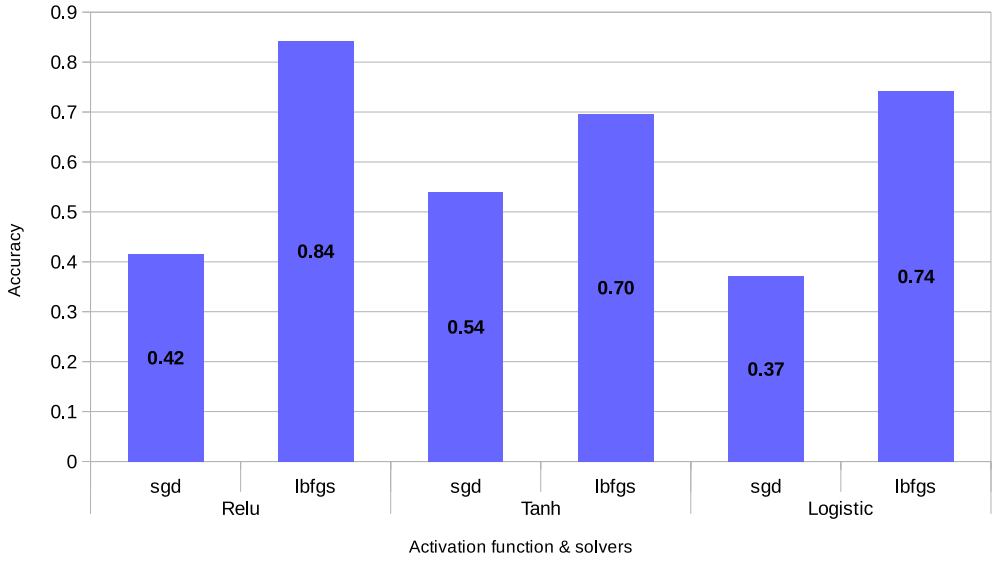

Figure 11. Classification model for dual method—tuning the activation function and solver (2 hidden layers, 100 neurons each).

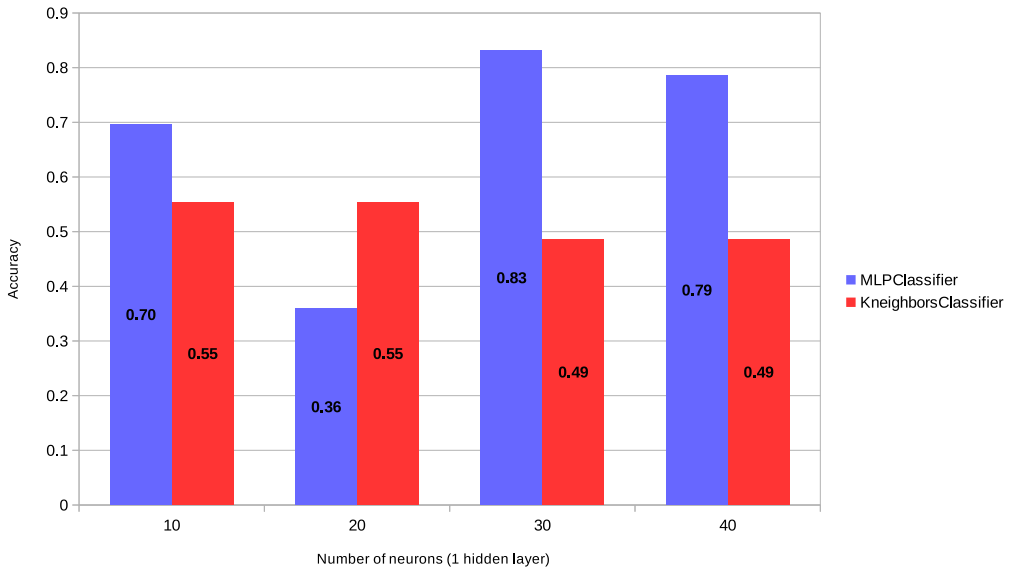

Figure 12. Classification model for dual method—testing different classification algorithms.

4. Conclusions

As stated at the beginning of this paper, an important step in solving linear programming problems is the selection of the most efficient algorithm. To select the most suitable algorithm, the majority of linear programming solvers use a heuristic procedure, based on the characteristics of the input linear programming problem. In this study, we examined the use of neural networks for predicting the execution time of CPLEX's primal and dual simplex algorithms. The results we received from the regression process were not satisfying enough to support a prediction model for the execution time of each method. Thus, we further experimented with classification approach, which led to meaningful results about the generated models, with the accuracy of our model for the primal method reaching 0.83, while for dual it was 0.84. Through classification, we may not be able to predict the exact value for the primal method and dual method's execution time, but we gain in predicting the class under which a specific problem can be classified. This action alone generates adequate information about the time needed for the solution of a problem, enabling the solver to select the most efficient of the examined simplex algorithms.

Author Contributions: Conceptualization, N.S. and N.P.; methodology and validation, N.S., N.P. and S.V.; formal analysis and investigation, N.P. and S.V.; writing—original draft preparation, S.V.; writing—review and editing, N.P. and N.S.; supervision, N.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Python source code and training data used in this study are available upon request. If interested please reach out to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bertsimas, D.; Tsitsiklis, J.N. *Introduction to Linear Optimization*; Athena Scientific: Belmont, MA, USA, 1997; Volume 6.
2. Ploskas, N.; Samaras, N. *Linear Programming Using MATLAB*; Springer: Cham, Switzerland, 2017.
3. Dantzig, G.B. Programming in linear structure. *Econometrica* **1949**, *17*, 73–74.
4. Dantzig, G.B. *Linear Programming and Extensions*; Princeton University Press: Princeton, NJ, USA, 1963.
5. Dongarra, J.; Sullivan, F. Guest editors' introduction: The top 10 algorithms. *Comput. Sci. Eng.* **2020**, *2*, 73–74. [CrossRef]
6. Lemke, C.E. The dual method of solving the linear programming problem. *Nav. Res. Logist. Q.* **1954**, *1*, 36–47. [CrossRef]
7. Frisch, K.R. *The Logarithmic Potential Method of Convex Programming*; Memorandum from the Institute of Economics, University of Oslo: Oslo, Norway, 1955.
8. Hoffman, A.; Mannos, M.; Sokolowsky, D.; Wiegmann, N. Computational experience in solving linear programs. *J. Soc. Ind. Appl. Math.* **1953**, *1*, 17–33. [CrossRef]
9. Neumann, J.V. *On a Maximization Problem*; Technical Report; Institute for Advanced Study: Princeton, NJ, USA, 1947.
10. Baz, M.; Hunsaker, B. *Automated Tuning of Optimization Software Parameters*; Technical Report; Department of Industrial Engineering, University of Pittsburgh: Pittsburgh, PA, USA, 2007.
11. Baz, M.; Hunsaker, B.; Prokopyev, O. How much do we "pay" for using default parameters? *Comput. Optim. Appl.* **2011**, *48*, 91–108. [CrossRef]
12. Franzin, A.; Cáceres, L.P.; Stützle, T. Effect of transformations of numerical parameters in automatic algorithm configuration. *Optim. Lett.* **2018**, *12*, 1741–1753. [CrossRef]
13. Barry, M.; Abgottspon, H.; Schumann, R. Solver tuning and model configuration. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*; Springer: Cham, Switzerland, 2018; pp. 141–154.
14. Audet, C.; Orban, D. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM J. Optim.* **2006**, *17*, 642–664. [CrossRef]
15. Chen, W.; Shao, Z.; Wang, K.; Chen, X.; Biegler, L. Random sampling-based automatic parameter tuning for nonlinear programming solvers. *Ind. Eng. Chem. Res.* **2011**, *50*, 3907–3918. [CrossRef]
16. Hutter, F.; Hoos, H.; Leyton-Brown, K. Automated configuration of mixed integer programming solvers. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*; Springer: Cham, Switzerland, 2010; pp. 186–202.

17. Hutter, F.; Hoos, H.; Leyton-Brown, K.; Stützle, T. ParamILS: An automatic algorithm configuration framework. *J. Artif. Intell. Res.* **2009**, *36*, 267–306. [CrossRef]
18. Liu, J.; Ploskas, N.; Sahinidis, N. Tuning BARON using derivative-free optimization algorithms. *J. Glob. Optim.* **2019**, *74*, 611–637. [CrossRef]
19. Sauk, B.; Ploskas, N.; Sahinidis, N. GPU parameter tuning for tall and skinny dense linear least squares problems. *Optim. Methods Softw.* **2020**, *35*, 638–660. [CrossRef]
20. Voulgaropoulou, S.; Samaras, N.; Sifaleras, A. Computational complexity of the exterior point simplex algorithm. *Oper. Res. Springer* **2019**, *19*, 297–316. [CrossRef]
21. Özaltın, O.; Hunsaker, B.; Schaefer, A. Predicting the Solution Time of Branch-and-Bound Algorithms for Mixed-Integer Programs. *INFORMS J. Comput.* **2011**, *23*, 392–403. [CrossRef]
22. Vanschoren, J. Meta-learning: A survey. *arXiv* **2018**, arXiv:1810.03548.
23. Wolpert, D.H.; Macready, W.G. *No Free Lunch Theorems for Search*; Technical Report; Technical Report SFI-TR-95-02-010; Santa Fe Institute: Santa Fe, New Mexico, 1995.
24. Priya, R.; de Souza, B.F.; Rossi, A.L.; de Carvalho, A.C. Predicting execution time of machine learning tasks using metalearning. In Proceedings of the 2011 World Congress on Information and Communication Technologies, Mumbai, India, 11–14 December 2011; pp. 1193–1198.
25. Brazdil, P.; Carrier, C.G.; Soares, C.; Vilalta, R. *Metalearning: Applications to Data Mining*; Springer-Verlag: Berlin/Heidelberg, Germany, 2009.
26. Matsunaga, A.; Fortes, J.A. On the use of machine learning to predict the time and resources consumed by applications. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, VIC, Australia, 17–20 May 2010; pp. 495–504.
27. Sun, J.; Sun, G.; Zhan, S.; Zhang, J.; Chen, Y. Automated performance modeling of HPC applications using machine learning. *IEEE Trans. Comput.* **2020**, *69*, 749–763. [CrossRef]
28. Pietri, I.; Juve, G.; Deelman, E.; Sakellariou, R. A performance model to estimate execution time of scientific workflows on the cloud. In Proceedings of the 2014 9th Workshop on Workflows in Support of Large-Scale Science, New Orleans, LA, USA, 16–21 November 2014; pp. 11–19.
29. Amaris, M.; de Camargo, R.Y.; Dyab, M.; Goldman, A.; Trystram, D. A comparison of GPU execution time prediction using machine learning and analytical modeling. In Proceedings of the 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 31 October–2 November 2016; pp. 326–333.
30. Krishnaswamy, S.; Loke, S.; Zaslavsky, A. Estimating computation times of data-intensive applications. *IEEE Distrib. Syst. Online* **2004**, *5*, 8374521. [CrossRef]
31. Smith, W. Prediction services for distributed computing. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium, Long Beach, CA, USA, 26–30 March 2007; pp. 1–10.
32. Smith, W.; Foster, I.; Taylor, V. Predicting application run times with historical information. *J. Parallel Distrib. Comput.* **2004**, *64*, 1007–1016. [CrossRef]
33. Tsafrir, D.; Etsion, Y.; Feitelson, D. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Trans. Parallel Distrib. Syst.* **2007**, *18*, 789–803. [CrossRef]
34. IBM ILOG CPLEX. CPLEX 12.6.0 User Manual. Available online: http://www-01.ibm.com/support/knowledgecenter/SSSA5P_12.6.1/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html?lang=en (accessed on 15 June 2021).
35. Voulgaropoulou, S.; Samaras, N.; Ploskas, N., Predicting the execution time of the interior point method for solving linear programming problems using artificial neural networks. In *Learning and Intelligent Optimization (LION 13)*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2019; pp. 319–324.
36. Carolan, W.J.; Hill, J.E.; Kennington, J.L.; Niemi, S.; Wichmann, S.J. An Empirical Evaluation of the KORBX® Algorithms for Military Airlift Applications. *Oper. Res. Informs* **1990**, *38*, 240–248. [CrossRef]
37. Mészáros, C. Mészáros Linear Programming Test Set. Available online: http://old.sztaki.hu/~meszaros/public_ftp/lptestset/ (accessed on 15 June 2021).
38. Mittelmann, H. Mittelmann's Linear Programming Test Set. Available online: http://plato.asu.edu/ftp/lptestset/ (accessed on 15 June 2021).
39. Netlib Repository. NETLIB Linear Programming Test Set. Available online: https://netlib.org/lp/ (accessed on 15 June 2021).
40. Cireşan, D.C.; Meier, U.; Schmidhuber, J. Transfer learning for Latin and Chinese characters with deep neural networks. In Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, 10–15 June 2012.
41. Alwosheel, A.; van Cranenburgh, S.; Chorus, C.G. Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis. *J. Choice Model.* **2018**, *28*, 167–182. [CrossRef]
42. Chauhan, V.; Joshi, K.D.; Surgenor, B. Machine vision for coin recognition with ANNs: Effect of training and testing parameters. In *Engineering Applications of Neural Networks, Proceedings of the 18th International Conference, EANN 2017, Athens, Greece, 25–27 August 2017*; Communications in Computer and Information Science; Boracchi, G., Iliadis, L., Jayne, C., Likas, A., Eds.; Springer: Cham, Switzerland, 2017; Volume 744, pp. 523–534.
43. Jain, A.K.; Chandrasekaran, B. Dimensionality and sample size considerations in pattern recognition practice. *Handb. Stat* **1982**, *39*, 835–855.

44. Kavzoglu, T.; Mather, P.M. The use of backpropagating artificial neural networks in land cover classification. *Int. J. Rem. Sens.* **2003**, *24*, 4907–4983. [CrossRef]
45. Raudys, S.J.; Jain, A.K. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Trans. Pattern Anal. Mach. Intell.* **1991**, *13*, 252–264. [CrossRef]
46. Maros, I.; Khaliq, M. Advances in Design and Implementation of Optimization Software. *Eur. J. Oper. Res.* **2002**, *140*, 322–337. [CrossRef]
47. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
48. Draper, N.R.; Smith, H. *Applied Regression Analysis*, 3rd ed.; John Wiley and Sons: Hoboken, NJ, USA, 1998.
49. Kutner, M.H.; Neter, J.; Nachtsheim, C.J.; Wasserman, W. *Applied Linear Statistical Models*, 5th ed.; McGraw-Hill/Irwin: New York, NY, USA, 2004.
50. Silva, I.N.; Spatti, D.H.; Flauzino, R.A.; Liboni, L.H.B.; dos Reis Alves, S.F. *Artificial Neural Networks. A Practical Course*; Springer International Publishing: Cham, Switzerland, 2017.
51. Bretscher, O. *Linear Algebra with Applications*, 3rd ed.; Prentice Hall: Upper Saddle River, NJ, USA, 1995.
52. Stigler, S.M. Gauss and the Invention of Least Squares. *Ann. Stat.* **1981**, *9*, 465–474. [CrossRef]
53. Pearson, K. Notes on regression and inheritance in the case of two parents. *Proc. R. Soc. Lond.* **1895**, *58*, 240–242. [CrossRef]
54. Stigler, S.M. Francis Galton's Account of the Invention of Correlation. *Stat. Sci.* **1989**, *4*, 73–79. [CrossRef]
55. Rao, C.R. *Coefficient of Determination, Linear Statistical Inference and Its Applications*, 2nd ed.; Wiley: New York, NY, USA, 1973.
56. Student. The probable error of a mean. *Biometrika* **1908**, *6*, 1–25. [CrossRef]
57. Fisher, R.A. On the interpretation of χ^2 from contingency tables and the calculation of P. *J. R. Stat. Soc.* **1922**, *85*, 87–94. [CrossRef]
58. Castle, J.; Doornik, J.; Hendry, D. Evaluating automatic model selection. *J. Time Ser. Econom.* **2011**, *3*, 1–33. [CrossRef]
59. Cook, D. Detection of Influential Observations in Linear Regression. *Technometrics Am. Stat. Assoc.* **1977**, *19*, 15–18.
60. Cook, D. Influential Observations in Linear Regression. *J. Am. Stat. Assoc. Am. Stat. Assoc.* **1979**, *74*, 169–174. [CrossRef]
61. Hosmer, D.; Jovanovic, B.; Lemeshow, S. Best subsets logistic regression. *Biometrics* **1989**, *45*, 1265–1270. [CrossRef]
62. Hyndman, R.; Koehler, A. Another look at measures of forecast accuracy. *Int. J. Forecast.* **2006**, *22*, 679–688. [CrossRef]
63. Hillier, F.S.; Lieberman, G.J. *Introduction to Operations Research*; Holden-Day Inc.: San Francisco, CA, USA, 1967.
64. Fawcett, T. An Introduction to ROC Analysis. *Pattern Recognit. Lett.* **2006**, *27*, 861–874. [CrossRef]