Evolutionary Algorithm for Large Margin Nearest Neighbour Regression

Florin Leon[1(✉)] and Silvia Curteanu[2]

[1] Department of Computer Science and Engineering, "Gheorghe Asachi" Technical University of Iaşi, Bd. Mangeron, 700050 Iaşi, Romania
fleon@cs.tuiasi.ro
[2] Department of Chemical Engineering, "Gheorghe Asachi" Technical University of Iaşi, Bd. Mangeron, 700050 Iaşi, Romania
scurtean@ch.tuiasi.ro

Abstract. The concept of a large margin is central to support vector machines and it has recently been adapted and applied for nearest neighbour classification. In this paper, we suggest a modification of this method in order to be used for regression problems. The learning of a distance metric is performed by means of an evolutionary algorithm. Our technique allows the use of a set of prototypes with different distance metrics, which can increase the flexibility of the method especially for problems with a large number of instances. The proposed method is tested on a real world problem – the prediction of the corrosion resistance of some alloys containing titanium and molybdenum – and provides very good results.

Keywords: Large margin · Nearest neighbour regression · Evolutionary algorithm · Prototypes

1 Introduction

Regression analysis includes any technique for modelling different kinds of processes with the goal of finding a relationship between a dependent variable and one or more independent variables, given a set of training instances or vectors in the form of ($\mathbf{x}_i$, y_i) pairs, where $\mathbf{x}_i$ are the inputs and y_i is the output of a sample. The general regression model is [8, 10, 11]:

$$y_i = f(\mathbf{x}_i) + e_i \,, \tag{1}$$

where f is the regression model (the approximating function), y_i is the desired output value that corresponds to the $\mathbf{x}_i$ input of the training set and e_i is a residual whose expected error given the sample point $\mathbf{x}_i$ is $E(e_i \mid \mathbf{x}_i) = 0$.

Presently, there are many methods that can be used for regression. Beside analytical models, where the task is to find adequate values for the coefficients usually by means of differential optimization, we can mention several machine learning techniques such as neural networks, support vector machines (ε-SVR, ν-SVR), decision

M. Núñez et al. (Eds.): ICCCI 2015, Part I, LNAI 9329, pp. 305–315, 2015.
DOI: 10.1007/978-3-319-24069-5_29

trees (M5P, Random Forest, REPTree) or rules (M5, Decision Table), etc. *k*-Nearest Neighbour (kNN) [5] is a simple, efficient way to estimate the value of the unknown function in a given point using its values in other points. Let $S = \{\mathbf{x}_1, ..., \mathbf{x}_m\}$ be the set of training points. The kNN estimator can be simply defined as the mean function value of the nearest neighbours [13]:

$$\tilde{f}(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}' \in N(\mathbf{x})} f(\mathbf{x}'), \tag{2}$$

where $N(\mathbf{x}) \subseteq S$ is the set of *k* nearest points to **x** in dataset *S*.

Another, more elaborate version of the method considers a weighted average, where the weight of each neighbour depends on its proximity to the query point:

$$\tilde{f}(\mathbf{x}) = \frac{1}{z} \sum_{\mathbf{x}' \in N(\mathbf{x})} w_d(\mathbf{x}, \mathbf{x}') \cdot f(\mathbf{x}'), \tag{3}$$

where *z* is a normalization factor, and the inverse of the squared Euclidian distance is usually employed to assess the weights:

$$w_d(\mathbf{x}, \mathbf{x}') = \frac{1}{d(\mathbf{x}, \mathbf{x}')^2} = \frac{1}{\sum_{i=1}^{n} (x_i - x'_i)^2}. \tag{4}$$

A comprehensive review of locally weighted regression and classification methods is given in [3]. Even if kNN is a very simple method, it usually performs very well for a wide range of problems, especially when the number of instances is large enough and there is little noise in the data. One of the most important aspects of the method is the choice of the distance function. While Euclidian distance is the most commonly encountered, other particularisations of the general Minkowski distance can be used, e.g. the Manhattan distance. Some experiments in cognitive psychology suggest that humans use an exponential negative distance function for certain types of classification tasks [1]. However, the classical approach does not take into account any information about a particular problem. Beside the common practice of normalizing the instance values independently on each dimension, there is little domain knowledge incorporated into the method. The present work investigates the use of the concept of a large margin (best known in the context of support vector machines) for regression problems, by adapting its existing formulation for classification problems. We organize our paper as follows. Section 2 presents the large-margin nearest neighbour method and some of its extensions. Section 3 describes the proposed method, including the use of prototypes and evolutionary algorithms for optimization. Section 4 focuses on a case study, and section 5 contains the conclusions.

2 Related Work

Because of the importance of the distance metric, researchers sought to find ways to adapt it to the problem at hand in order to yield better performance. This is the issue of distance metric learning [15, 14, 4, 6]. The idea of a large margin, one of the fundamental ideas of support vector machines, was transferred to the kNN method for classification tasks [16-18], resulting in the large margin nearest neighbour method (LMNN). In this case, learning involves the optimization of a convex problem using semidefinite programming. The LMNN technique was also extended to incorporate invariance to multivariate polynomial transformations [9]. Since our regression method builds on the LMNN method for classification [18], we will present it with more details as follows.

In general, distance metric learning aims at finding a linear transformation $\mathbf{x}' = \mathbf{L}\mathbf{x}$, such that the distance between two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ becomes:

$$d_L(\mathbf{x}_i, \mathbf{x}_j) = \left\| \mathbf{L}\left(\mathbf{x}_i - \mathbf{x}_j\right) \right\|_2 . \tag{5}$$

Since all the operations in k-nearest neighbour classification or regression can be expressed in terms of square distances, an alternative way of stating the transformation is by means of the square matrix: $\mathbf{M} = \mathbf{L}^T\mathbf{L}$, and thus the square distance is:

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i - \mathbf{x}_j\right)^T \mathbf{M}\left(\mathbf{x}_i - \mathbf{x}_j\right). \tag{6}$$

For a classification problem, the choice of **L**, or equivalently **M**, aims at minimizing the distance between a vector $\mathbf{x}_i$ and its *k* target neighbours $\mathbf{x}_j$, where a target signifies a neighbour that belongs to the same class. At the same time, the distance between a vector and the impostors $\mathbf{x}_l$, i.e. neighbours that belong to a different class, should be maximized. In order to establish a large margin between the vectors that belong to different classes, the following relation is imposed:

$$d_M(\mathbf{x}_i, \mathbf{x}_l) \geq 1 + d_M(\mathbf{x}_i, \mathbf{x}_j) . \tag{7}$$

Here, the value of 1 is arbitrary; the idea is to have some minimum value for the margin that separates the classes. However, it was proven that other minimum values for the margin would not change the nature of the optimization problem, but will only result in the scaling of the matrix **M**.

Overall, the optimization problem is defined as follows:

$$\begin{aligned} \text{min} \quad & \sum_{ij} \eta_{ij} d_{ij} + \lambda_h \sum_{ijl} \nu_{ijl} \xi_{ijl} \\ \text{such that} \quad & \left(\mathbf{x}_i - \mathbf{x}_j\right)^T \mathbf{M}\left(\mathbf{x}_i - \mathbf{x}_j\right) = d_{ij} \\ & \left(\mathbf{x}_i - \mathbf{x}_l\right)^T \mathbf{M}\left(\mathbf{x}_i - \mathbf{x}_l\right) - d_{ij} \geq 1 - \xi_{ijl} \\ & \mathbf{M} \succeq 0, \ \xi_{ijl} \geq 0, \forall i, j, l \end{aligned} \tag{8}$$

where $\eta_{ij} \in \{0,1\}$ is 1 only when $\mathbf{x}_j$ is a target neighbour of $\mathbf{x}_i$, $\nu_{ijl} = \eta_{ij}$ if and only if $y_i \neq y_l$ and 0 otherwise, d_{ij} is the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$, ξ_{ijl} is the hinge loss [12] and $\lambda_h \geq 0$ is a constant.

The two objectives, of minimizing the distance to targets and maximizing the distance to impostors, are conflicting. Following an analogy to attraction and repulsion forces in physics, Weinberger and Saul [18] introduce two forces, ε_{pull} and ε_{push}, whose balance is reached by setting the value of λ_h. In their work, the authors consider the importance of these forces to be equal, i.e. $\lambda_h = 1$.

Other researchers (e.g. [9]) suggest the use of regularization to avoid the overfitting caused by large values of the elements of **L** or **M**. Regularization can be performed for example by minimizing the Frobenius norm of the parameters, and thus adding a third term to the optimization problem:

$$\min \sum_{ij} \eta_{ij} d_{ij} + \lambda_h \sum_{ijl} \nu_{ijl} \xi_{ijl} + \lambda_r \sqrt{\sum_p \sum_q m_{pq}^2} , \tag{9}$$

where $\lambda_r \geq 0$ is another constant.

Although there are several studies concerned with LMNN classification, so far not many researchers have addressed the issue of regression. One recent paper that presents some modifications of the LMNN metric learning for regression is [2].

3 Description of the Proposed Method

A binary classification problem can be considered as a special case of a regression problem where the desired function only takes two values: 0 and 1. In this section, we will generalize the concepts of the LMNN method and adapt it for regression. As an optimization tool we will consider an evolutionary algorithm, which can provide more flexibility in situations when we want to learn not a single distance metric, but several.

3.1 Distance Metric Learning Using Prototypes

In our work, we considered **L** (and thus **M**) to be diagonal matrices. This increases the clarity of the results, because in this case the elements m_{ii} can be interpreted as the weights of the problem input dimensions, and is also a form of regularization. Given the fact that an evolutionary algorithm is used for finding the elements of the matrix, it can be easily applied to find a full matrix **L**. In the unrestricted case, the elements of **M** can be directly found only while satisfying some constraints, because they should comply with the relation $\mathbf{M} = \mathbf{L}^T\mathbf{L}$, i.e. **M** should be symmetric and positive semidefinite.

By using the **M** matrix, the relation between the neighbour weights and the distance in equation (4) still holds, but now we have:

$$w_{d_M}(\mathbf{x},\mathbf{x}') = \frac{1}{d_M(\mathbf{x},\mathbf{x}')} = \frac{1}{\sum_{i=1}^{n} m_{ii} \cdot (x_i - x'_i)^2}. \tag{10}$$

In this formulation, there is a single, global matrix **M** for all the instances. However, it is possible to have different distance metrics for the different instances or groups of instances. We introduce the use of *prototypes*, which are special locations in the input space of the problem, such that each prototype P has its own matrix $\mathbf{M}(P)$. When computing the distance weight to a new point, an instance will use the weights of its nearest prototype, i.e. $m_{ii}(P)$ instead of m_{ii} in equation (10).

3.2 The Evolutionary Algorithm

The following parameters of the evolutionary algorithm were used in the experiments: 40 chromosomes in the population, tournament selection with 2 candidates, arithmetic crossover with a probability of 95% and mutation with a probability of 5% in which a gene value is reinitialized to a random value in its corresponding domain of definition. Elitism was also used such that the best solution in a generation is never lost. As a stopping criterion, a maximum number of generations, 500 in our case, was used. The number of genes in a chromosome depends on the number of prototypes and the dimensionality of the problem, namely: $n_g = n_p \cdot n_i$, where n_g is the number of genes, n_p is the number of prototypes and n_i is the number of inputs.

The domain of the genes, which represent the elements of **L**, is $[10^{-3}, 10]$. All the operations in the software implementation described later deal with the elements of **M**, i.e. their squares. Thus it can be considered that the corresponding values of the **M** elements lie in the $[10^{-6}, 10^{2}]$ domain.

The fitness function F, which is to be minimized, takes into account 3 criteria:

$$F = w_1^F \cdot F_1 + w_2^F \cdot F_2 + w_3^F \cdot F_3, \tag{11}$$

where the weights of the criteria are normalized: $w_1^F + w_2^F + w_3^F = 1$.

In order to simplify the expressions of the F_i functions, let us make the following notations, where d_M means the weighted square distance function using the weights we search for: $d_{ij} = d_M(\mathbf{x}_i, \mathbf{x}_j)$, $d_{ik} = d_M(\mathbf{x}_i, \mathbf{x}_k)$, $g_{ij} = \left|f(\mathbf{x}_i) - f(\mathbf{x}_j)\right|$ and $g_{ik} = \left|f(\mathbf{x}_i) - f(\mathbf{x}_k)\right|$.

Thus, the first criterion is:

$$F_1 = \sum_{i=1}^{n} \sum_{j \in N(i)} d_{ij} \cdot (1 - g_{ij}), \tag{12}$$

where $N(i)$ is the set of the nearest k neighbours of instance i, in our case $k = 3$. Basically, this criterion says that the nearest neighbours of i should have similar values to the one of i, and more distant ones should have different values. This criterion tries to

minimize the distance between an instance *i* and its neighbours with similar values. If a neighbour *j* has a dissimilar value, the second factor, $1 - g_{ij}$, becomes small and the distance is no longer necessary to be minimized.

The second criterion is expressed as follows:

$$F_2 = \sum_{i=1}^{n} \sum_{j \in N(i)} \sum_{l \in N(i)} \max\left(1 + d_{ij} \cdot \left(1 - g_{ij}\right) - d_{il} \cdot \left(1 - g_{il}\right), 0\right). \tag{13}$$

It takes into account a pair of neighbours, *j* and *l*, by analogy to a target and an impostor. However, for our regression problem we do not have these notions because we do not have a class which could be the same or different. We can only take into account the real values of the instance outputs. The reasoning is the same as for the first criterion, but now we try to minimize the distance to the neighbours with close values (the positive term), while simultaneously trying to maximize the distance to the neighbours with distant values (the negative term). By analogy to equation (8), we consider that a margin of at least 1 should be present between an instance with a close value and another with a distant value. The *max* function is used by analogy to the expression of the hinge loss. If we consider that *j* has a value close to the value of *i* and *l* has a distant one, the corresponding hinge loss will be 0 only when $d_{il} \geq 1 + d_{ij}$. The condition $j \neq l$ is implicit because when the terms are equal they cancel each other out.

The third criterion is used for regularization, to avoid large values of the weights:

$$F_3 = \sum_{j=1}^{n_p} \sum_{i=1}^{n_i} m_{ii}(j) . \tag{14}$$

However, for our case study presented in section 4, the weights are very small and this term is not needed. Overall, we use the following values for the weights of the criteria: $w_1^F = w_2^F = 0.5$ and $w_3^F = 0$.

The advantage of using an evolutionary algorithm for optimization is that prototypes can be used, with different weight values, instead of a single, global set of weights. As mentioned above, when computing the distance from a certain training instance, the distance is weighted by the values corresponding to the nearest prototype of the training instance.

We compute the positions of the prototypes using k-means, a simple clustering algorithm which tends to favour (hyper)-spherical clusters. This is why it is very appropriate for our problem, where distances are computed with variations of the Euclidian metric. However, the scope of the evolutionary algorithm can be expanded to also find the optimal number of clusters as well as their positions in the input space.

4 Case Study

In this section we will present the results of applying the regression method to a practical problem, namely the prediction of the corrosion of some alloys containing titanium and molybdenum (TiMo).

Titanium (Ti) and its alloys are widely used in dental applications due to the excellent corrosion resistance and mechanical properties. However, it has been reported that Ti is sensitive to fluoride (F^-) and lactic acid. Consequently, the samples were examined using electrochemical impedance spectroscopy (EIS) in acidic artificial saliva with NaF and/or caffeine. The material corrosion was quantified by the polarization resistance (R_p) of the TiMo alloys (output of the model) which was modelled depending on the immersion time, caffeine concentration, NaF concentration, type of alloy (Ti content), and solution pH (inputs of the model). The experimental data covered a large domain, corresponding to the following conditions: 12, 20 and 40 wt.% of Mo, 0.1, 0.3 and 0.5 wt.% NaF, 0, 0.5, 1 and 1.5 mg/mL caffeine and 3-8 for solution pH.

In order to compare different algorithms and models, we consider 2 separate problems. The first is to assess the performance on the testing set, after randomizing the dataset and selecting 2/3 of the instances as the training set and 1/3 as the testing set. The second one is to perform cross-validation with 10 folds.

We use the coefficient of determination (r^2), the squared coefficient of correlation, as a metric to compare the performance of different models. Table 1 presents the results obtained with various algorithms implemented in the popular collection of machine learning algorithms Weka [7], for the training-testing split and for cross-validation. The results are sorted in decreasing order of the cross-validation results. On each column, the maximum value is emphasised with bold characters.

Table 1. Performance of different algorithms implemented in Weka

Algorithm	Training Set	Testing Set	Cross-validation
REPTree	0.9789	0.8789	**0.8983**
ν-SVR, RBF	0.9109	0.8140	0.8962
M5 Rules	0.8962	0.8127	0.8953
Random Forest	0.9833	**0.9181**	0.8915
kNN, k = 10	0.9994	0.8974	0.8363
ε-SVR, RBF	0.7583	0.6726	0.7910
ν-SVR, P2	0.8187	0.7268	0.7739
ε-SVR, P2	0.7903	0.7022	0.7533
NN	**1.0000**	0.8516	0.6655
Additive Regression	0.6336	0.5869	0.5895

When using our software implementation for the regression problem, 10 neighbours are considered when computing the output, because this value gave the best results for the kNN method in Weka. Fewer neighbours do not provide enough information to compute a precise value. If more neighbours are used, the influence of the more distant ones becomes negligible, since the neighbour weights are proportional to the inverse of the square distance.

In the computation of the fitness function, 3 reference neighbours are used. This number should be at least 2, in order to provide some contrast between an instance with a close value and another with a distant value (by analogy with the target and impostor instances for classification). With 2 reference neighbours, the results are far

worse than with 3, with the best coefficient of determination of 0.8984. With more reference neighbours, the computation time greatly increases, while the results are no better than in case of 3 reference neighbours.

Table 2 shows the average and best results for the case of the training set – testing set split and also for cross-validation with 10 folds, while considering 4 different configurations: with 1, 2, 5 and 10 prototypes. Theoretically, each instance could have its own metric, but in our case, with a dataset of 1152 instances, the computation time would be too high for any practical purpose. The positions of the clusters are obtained with the k-means algorithm taking into account only the inputs of the instances. The results presented are obtained after 50 runs. The results with $r^2 < 0.5$ were eliminated because they were considered to be outliers. The results on the training set are omitted, because they are 1 in all cases.

Table 2. Performance of the proposed method with different numbers of prototypes

No. Prototypes	Testing Set		Cross-validation	
	Average	Maximum	Average	Maximum
1	0.8436	**0.9703**	0.8056	0.9160
2	0.8349	0.9394	0.8123	**0.9207**
5	0.7914	0.8517	0.7432	0.8273
10	0.6997	0.8270	0.6439	0.7917

Table 3. The best weights obtained for the training-testing problem with: a) 1 prototype; b) 2 prototypes

Input	m_{ii} x 1000
x_1	0.0719
x_2	0.0589
x_3	0.8551
x_4	0.0463
x_5	0.1479

Input	Prototype 1 position	Prototype 2 position	$m_{ii}(1)$ x 1000	$m_{ii}(2)$ x 1000
x_1	0.4507	0.4612	0.3889	0.0902
x_2	0.3450	0.3639	1.8994	0.0421
x_3	0.4987	0.5142	0.0193	0.0027
x_4	0.5109	0.4833	0.0716	2.5454
x_5	0.0987	0.7964	0.2239	0.0205

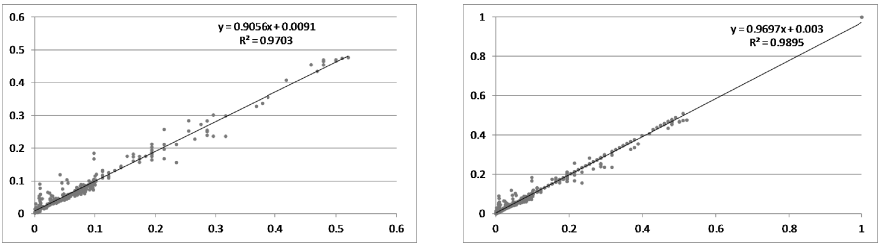

Fig. 1. Comparison between the predictions of the best model and the expected data: a) for the test set; b) for the whole dataset

While cross-validation is a standard way to compare several algorithms, it cannot provide a unique set of "good" values for the internal parameters, because it generates 10 different models. Therefore, the single split into training and testing sets is useful in this respect.

The best performance for the testing set is given by the configuration with 1 prototype, i.e. a global set of weights for all the instances of the problem. The actual values of the weights are presented in table 3a.

Figures 1a and 1b present the correlation between the desired values and the values provided by the model, for the testing set alone and for the whole dataset. Beside the high value of the coefficient of determination, one can also graphically observe the good fit of the data.

Table 3b presents the weights for the best fit on the training-testing problem, corresponding to 2 prototypes.

Beside the variation in performance between different runs, which is normal especially since the number of generations (500) and individuals in the population (40) may be a little too small for our problem, we hypothesise that there may be another reason why 1 prototype provides the best results for the training-testing case and 2 prototypes provide the best results for cross-validation. In one fold of cross-validation 90% of the data are used, compared to 67% for training in the first case. Therefore, the data diversity is larger and 2 prototypes provide more flexibility. Also the positions of the prototypes are currently fixed, pre-computed, taking into account the whole dataset, and thus the 90% situation better matches the distribution of the full data. However, when the number of prototypes increases, the problem space becomes too finely partitioned, because the evolutionary algorithm evolves the weights independently and it is almost certain that the sets of weights corresponding to different prototypes will be different, although the actual topology of the problem may not require such distinctions in the distance metric of the instances.

5 Conclusions

The results obtained for a rather difficult problem using a large margin nearest neighbour regression method are quite promising. The weights are obtained using an evolutionary algorithm, which provides simplicity and flexibility and allows the use of several distance metrics in different regions of the problem space, corresponding to different prototypes. The evolutionary paradigm can be easily applied to also find the optimal number of prototypes and their positions, at the expense of a large increase in the search space and thus computation time. The presented method allows the user to change the number of neighbours that are considered for distance calculation and the weights of the criteria of the composite fitness function, in order to adapt these parameters to the particular characteristics of the problem. The modelling procedure applied for the evaluation of the polarization resistance of the TiMo alloys as a function of process conditions contributes to a better understanding of the process and can partially replace a series of experiments that are time, material and energy consuming.

Acknowledgment. This work was supported by the "Partnership in priority areas – PN-II" program, financed by ANCS, CNDI - UEFISCDI, project PN-II-PT-PCCA-2011-3.2-0732, No. 23/2012.

References

1. Aha, D.W., Goldstone, R.L.: Concept learning and flexible weighting. In: Proceedings of the 14th Annual Conference of the Cognitive Science Society, pp. 534–539 (1992)
2. Assi, K.C., Labelle, H., Cheriet, F.: Modified Large Margin Nearest Neighbor Metric Learning for Regression. IEEE Signal Processing Letters **21**(3), 292–296 (2014). doi:10.1109/LSP.2014.2301037
3. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally weighted learning. Artificial Intelligence Review **11**(1-5), 11–73 (1997)
4. Chopra, S., Hadsell, R., LeCun, Y.: Learning a similarity metric discriminatively, with application to face verification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2005, pp. 349–356, San Diego, CA, USA (2005)
5. Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. IEEE Transactions in Information Theory, IT-13, 21–27 (1967)
6. Goldberger, J., Roweis, S., Hinton, G., Salakhutdinov, R.: Neighbourhood components analysis. In: Advances in Neural Information Processing Systems, vol. 17, pp. 513–520. MIT Press, Cambridge (2005)
7. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. ACM SIGKDD Explorations **11**(1), 10–18 (2009)
8. Hansen, B.E.: Nearest Neighbor Methods. NonParametric Econometrics. University of Wisconsin-Madison, pp. 84-88 (2009). http://www.ssc.wisc.edu/~bhansen/718/NonParametrics10.pdf (accessed March 15, 2015)
9. Kumar, M.P., Torr, P.H.S., Zisserman, A.: An invariant large margin nearest neighbour classifier. In: Proceedings of the IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil (2007). doi: 10.1109/iccv.2007.4409041
10. Leon, F., Piuleac, C.G., Curteanu, S., Poulios, I.: Instance-Based Regression with Missing Data Applied to a Photocatalytic Oxidation Process. Central European Journal of Chemistry **10**(4), 1149–1156 (2012). doi:10.2478/s11532-012-0038-x
11. Mareci, D., Sutiman, D., Chelariu, R., Leon, F., Curteanu, S.: Evaluation of the corrosion resistance of new TiZr binary alloys by experiment and simulation based on regression model with incomplete data. Corrosion Science, Elsevier **73**, 106–122 (2013). doi:10.1016/j.corsci.2013.03.030
12. Moore, R.C., DeNero, J.: L1 and L2 regularization for multiclass hinge loss models. In: Proceedings of the Symposium on Machine Learning in Speech and Language Processing (2011)
13. Navot, A., Shpigelman, L., Tishby, N., Vaadia, E.: Nearest neighbor based feature selection for regression and its application to neural activity. In: Advances in Neural Information Processing Systems, vol. 19 (2005)
14. Shalev-Shwartz, S., Singer, Y., Ng, A.Y.: Online and batch learning of pseudo-metrics. In: Proceedings of the 21st International Conference on Machine Learning, ICML 2004, pp. 94–101, Banff, Canada (2004)
15. Shental, N., Hertz, T., Weinshall, D., Pavel, M.: Adjustment learning and relevant component analysis. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) ECCV 2002, Part IV. LNCS, vol. 2353, pp. 776–790. Springer, Heidelberg (2002)

16. Weinberger, K.Q., Blitzer, J., Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. In: Advances in Neural Information Processing Systems, vol. 18, pp. 1473–1480. MIT Press, Cambridge (2006)
17. Weinberger, K.Q., Saul, L.K.: Fast solvers and efficient implementations for distance metric learning. In: Proceedings of the 25th International Conference on Machine Learning, pp. 1160–1167, Helsinki, Finland (2008)
18. Weinberger, K.Q., Saul, L.K.: Distance Metric Learning for Large Margin Nearest Neighbor Classification. Journal of Machine Learning Research **10**, 207–244 (2009)