A General Method for Visualizing and Explaining Black-Box Regression Models

Erik Štrumbelj and Igor Kononenko

Faculty of Computer and Information Science, University of Ljubljana
Tržaška 25, 1000 Ljubljana, Slovenia
{erik.strumbelj,igor.kononenko}@fri.uni-lj.si

Abstract. We propose a method for explaining regression models and their predictions for individual instances. The method successfully reveals how individual features influence the model and can be used with any type of regression model in a uniform way. We used different types of models and data sets to demonstrate that the method is a useful tool for explaining, comparing, and identifying errors in regression models.

Keywords: Neural networks, SVM, prediction, transparency.

1 Introduction

Explaining prediction models and their predictions is an integral part of machine learning. The purpose of such methods is making models more informative, easier to understand, and easier to use. These benefits are especially welcome when using non-transparent prediction models, such as artificial neural networks and SVM. Some of the most popular learning algorithms (naive Bayes, decision trees) owe a part of their popularity to their ability to produce models which are inherently easy to interpret. For others, model-specific explanation and visualization methods have been developed [3,5,6]. There also exist general methods that can be applied to any model. The latter are the focus of this paper.

Before discussing general explanation methods, we start with a simple example. Figure 1 is an explanation for an instance from the artificial data set *testA*. Instances from this data set describe the situation involving a student in consultation with a professor about his final mark. The teacher can immediately pass the student or may opt to test the student with additional questions in which case it comes down to the student's knowledge to determine whether the student will pass. The model's task is to predict the student's chances of success. The binary feature *teacher* describes the teachers action. The feature *student* describes the student's knowledge and has 6 possible equally spread levels, where 0 means certain failure, 1 means 20% chance,..., and 5 means certain success. In *testA* all combinations of pairs of values the two features are equally probable. The explanation in Figure 1 is consistent with our intuition and helps us understand the model's prediction. Observe how the explanation is given in the form of magnitudes and directions of features' contributions. Assigning a contribution

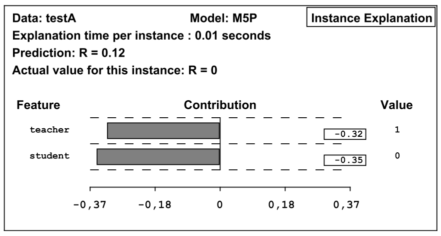

Fig. 1. The decision tree makes a dire prediction (0.12) for this poorly prepared student (student = 0) who will be tested (teacher = 1). The explanation suggests that both features have an approximately equal contribution. Both are negative, speaking against the student's chances.

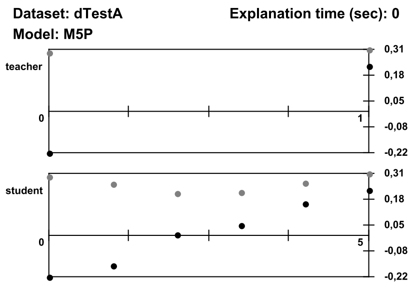

Fig. 2. A general explanation reveals that both features are approximately equally important (grey dots). Studying increases the student's chances. Not being tested is beneficial while being tested has an opposite effect.

(score, rank, etc...) is a common approach and is used in most of the previously mentioned model-specific methods and in all of the general methods that follow.

By using a general method machine learning and data mining practitioners can avoid using a different model-specific explanation method for each different model, which also simplifies comparison. Furthermore, in a practical setting it is very desirable, especially from the end-user's perspective, that the explanation method need not be replaced if the underlying prediction models change. To achieve such generality methods must avoid anything model-specific, essentially treating every model as a black-box, limiting all interaction to changing the inputs (feature values) and observing the outputs. Clearly, going through all possible combinations of input values is infeasible, so each method is forced in some sort of a tradeoff between its time complexity and the complexity of what it can extract from a model.

Some existing methods, such as [7] and [4] use the "one feature at a time" approach. A feature's contribution for a particular instance is defined as the average change in prediction when the feature's value is permuted. While this reduces the time complexity, it, in some cases, does not result in a change that reveals the true importance of a feature. Observe how the value of the expression $1 \vee 1$ does not change if we change either of the 1's to 0. Both must be changed at the same time to achieve a change. A recently published paper introduces FIRM, a method for computing the importance of features for a given model [9]. For each feature the method observes the variance of the conditional expected output of the model, across all possible values of that feature (conditional to the given value of the feature). However, observe how for two uniformly distributed binary variables $E[b_1 \text{ XOR } b_2 | b_1 = 1] = E[b_1 \text{ XOR } b_2 | b_1 = 0] = 0.5$. The conditional expected outputs will be the same and variance will be 0. A clearly important variable will be assigned 0 importance.

A method that solves the problems mentioned in the previous paragraph was recently developed for classification models [8]. The authors' basic idea is to observe changes across all subsets of features (for example, also observing how the value of $1 \vee 1$ changes if we change both values at the same time). The exponential time complexity is resolved by an approximation algorithm. However, unresolved issues remain. First, it is limited to classification models and can not be used to explain a regression model. Second, it can only be used to explain a particular instance (see Figure 1) - users would benefit from a global overview of how features contribute (see Figure 2). And third, the proposed approximation algorithm is based on a very strict assumption that all combinations of feature values are equiprobable. Successfully dealing with the first two issues and loosening the assumption in the third are the main contributions of this paper.

The remainder of the paper is divided into 3 sections. In Section 2 we adapt the explanation method for use with regression models and introduce improvements. Section 3 describes a series of experiments on artificial data sets, followed by an experiment on a real-world data set. With Section 4 we conclude the paper and give some ideas for further work.

2 Explaining Regression Models' Predictions

Let $\mathcal{A} \in \mathcal{A}_1 \times \mathcal{A}_2 \times ... \times \mathcal{A}_n$ be our feature space, where each feature $\mathcal{A}_i$ is a set of values. Let p be the probability mass function defined on the sample space $\mathcal{A}$. Let $f : \mathcal{A} \rightarrow \Re$ be our regression model. No other assumptions are made about f. Let $S = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$. The influence of a certain subset of features $Q \subseteq S$ in a given instance $x \in \mathcal{A}$ is defined as

$$\Delta(Q)(x) = E[f|\text{values of features in } Q \text{ for } x] - E[f]. \tag{1}$$

In other words, the contribution of a subset of feature values in a particular instance is the change in expectation caused by observing those feature values.

Suppose we have $\Delta(Q)(x)$ for every $Q \subseteq S$. How do we combine these values to form contributions of individual feature values? In [8] they propose using the well known game-theoretic solution - the Shapley value - to define $\varphi_i(x)$, the contribution of the $i-$th feature for instance x:

$$\varphi_i(x) = \sum_{Q \subseteq S \setminus \{i\}} \frac{|Q|!(|S| - |Q| - 1)!}{|S|!} (\Delta(Q \cup \{i\})(x) - \Delta(Q)(x)). \tag{2}$$

Eq. 2 has desirable properties. The feature contributions are implicitly normalized (they sum up to the initial difference Δ_S), which makes them easier to interpret. If a feature does not have any impact on the prediction, will be assigned a 0 contribution. And, features with a symmetrical impact will be assigned equal contributions. The work described so far in this section is credited to [8] and only minor modifications were necessary to apply the method to a regression setting (in our case f is a regression model's output, instead of a classification model's probabilistic prediction for a given class value).

2.1 Approximation Algorithm

Eq. 2 reflects any influence the feature might have on the prediction. However, in practice it is often impossible to calculate the Δ-terms due to the time complexity. Even if we could, we still face the exponential time complexity of computing $\varphi_i(x)$. In [8] this is resolved by assuming that $p(x) = \frac{1}{|\mathcal{A}|}$, for all $x \in \mathcal{A}$. For any given feature space this assumption limits the choice of p to a single possibility. The distribution of values plays an important part in how people intuitively explain events. Recall the teacher/student scenario. The concept that students are more likely to pass if they study or are not tested is universal (that is, such a model would perform well on any university with a similar concept, regardless of the distribution of feature values). Our intuitive explanation depends heavily on the distribution of feature values. For example, a student who does not study and is tested will fail. If this teacher tests students most of the time, we would say that it is mostly the student's own fault for not studying. On the other hand, if the teacher almost never tests a student, most would say it was "bad luck" (that is, being tested is a much more important contributor than the amount of study). This example emphasizes the importance of providing more flexibility wrt the choice of p, while still retaining an efficient explanation algorithm.

To loosen the restriction, we assume that p is such that individual features are mutually independent. Then transform Eq. 1 into

$$\Delta(Q)(x) = \sum_{y \in \mathcal{A}} p(y) \left(f(\tau(x, y, Q)) - f(y)\right) \tag{3}$$

Note that $\tau(x, y, W) = (z_1, z_2, ..., z_n)$, where $z_i = x_i$ iff $i \in W$ and $z_i = y_i$, otherwise. We use the alternative formulation of the Shapley value (equivalent to Eq. 2)

$$\varphi_i(x) = \frac{1}{n!} \sum_{\mathcal{O} \in \pi(n)} \Big(\Delta(Pr^i(\mathcal{O}) \cup \{i\})(x) - \Delta(Pr^i(\mathcal{O}))(x)\Big), \tag{4}$$

where $\pi(n)$ is the set of all permutations of n elements and $Pr^i(\mathcal{O})$ is the set of all features which precede the i-th feature in permutation $\mathcal{O} \in \pi(n)$. By combining Eq. 3 and Eq. 4, we get

$$\varphi_i(x) = \frac{1}{n!} \sum_{\mathcal{O} \in \pi(N)} \sum_{y \in \mathcal{A}} p(y) \cdot (f(\tau(x, y, Pr^i(\mathcal{O}) \cup \{i\})) - f(\tau(x, y, Pr^i(\mathcal{O})))), \tag{5}$$

which facilitates the use of random sampling and an efficient approximation algorithm (see Algorithm 1). Note that *at random* refers to drawing each feature's value at random, according to the distribution of that feature's values (usually, by sampling from a data set). Note that due to with replacement features with finite and infinite domains are treated identically. Therefore, it can be applied to both nominal and numeric features.

Observe the same model's prediction for the same instance, but from data set *testB* where the teacher tests the students a vast majority of time (Figure 3) and compare to Figure 1. The explanation now depends on the context and

Algorithm 1 Approximating $\varphi_i(x)$, the importance of the i-th feature's value for instance x and model f. Take m samples.

```
$\varphi_i(x) \leftarrow 0$
for $k = 1$ to $m$ do
    select (at random) permutation $\mathcal{O} \in \pi(n)$ and instance $y \in \mathcal{A}$
    $x_1 \leftarrow \overbrace{\boxed{\text{feat. preceding } i \text{ in } \mathcal{O}}\ \boxed{i}}^{\text{take their values from } x}\ \overbrace{\boxed{\text{feat. succeeding } i \text{ in } \mathcal{O}}}^{\text{take their values from } y}$
    $x_2 \leftarrow \overbrace{\boxed{\text{feat. preceding } i \text{ in } \mathcal{O}}}^{\text{take their values from } x}\ \overbrace{\boxed{i}\ \boxed{\text{feat. succeeding } i \text{ in } \mathcal{O}}}^{\text{take their values from } y}$
    $\varphi_i(x) \leftarrow \varphi_i(x) + f(x_1) - f(x_2)$
end for
$\varphi_i(x) \leftarrow \frac{\varphi_i(x)}{m}$
```

Algorithm 2 Approximating $\psi_{i,j}$, the global importance of the i-th feature's value j for model f. Take m samples.

```
$\psi_{i,j} \leftarrow 0$
for $k = 1$ to $m$ do
    select (at random) instance $y \in \mathcal{A}$
    $x_1 \leftarrow$ set $i$-th feature to $j$, take other values from $y$
    $\psi_{i,j} \leftarrow \psi_{i,j} + f(x_1) - f(y)$
end for
$\psi_{i,j} \leftarrow \frac{\psi_{i,j}}{m}$
```

the proposed explanation method provides us with explanations which are in accordance with our own intuitive explanation.

Figures 1 and 3 show how individual feature values influence the model's prediction *for a given instance*. For a global overview of how a feature contributes, we could observe the contributions across several instances. Instead, we provide the same information within a single visualization. We define the global contribution of the i-th feature's j-th value as the expected value of that feature's contribution (see Eq. 5) for an instance where its value is j:

$$\begin{aligned}\psi_{i,j} &= \sum_{x \in \mathcal{A}, x[i]=j} p(x)\varphi_i(x) = \sum_{x \in \mathcal{A}} p(x)\varphi_i(x) = \\ &= \frac{1}{n!} \sum_{\mathcal{O} \in \pi(N)} \sum_{x \in \mathcal{A}} p(x) \left(f(x') - f(x)\right) = \sum_{x \in \mathcal{A}} p(x) \left(f(x') - f(x)\right),\end{aligned} \tag{6}$$

where x' is x with i-th feature's value set to j). Eq. 6 can be approximated using Algorithm 2.

Figure 2 is a visualization of the global importance of features for our illustrative data set *testA*. Each grey/black point pair is obtained by running Algorithm

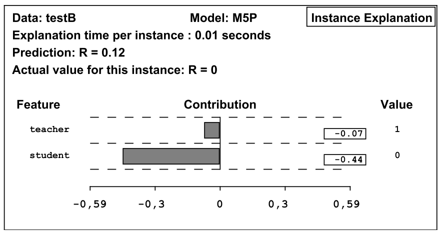

Fig. 3. If it is likely that the teacher will test the student then studying hard (or lack of) becomes much more important.

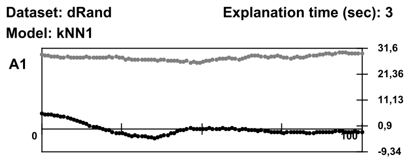

Fig. 4. KNN1 does not perform well, but the features have a strong influence on its predictions. We can conclude it overfits.

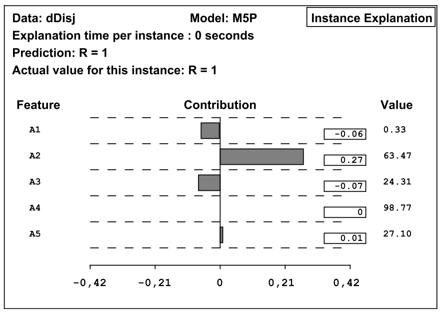

Fig. 5. M5P successfully models dDisj and correctly predicts $R = 1$. The visualization shows that a single feature is responsible for the prediction, while the other two have the opposite effect.

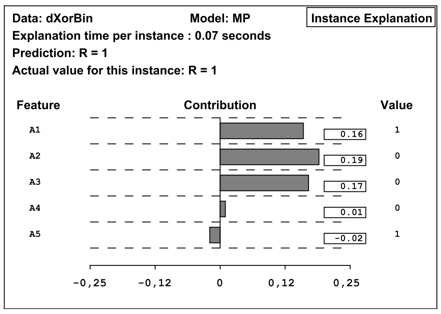

Fig. 6. The neural network successfully models dXorBin and correctly predicts this instance. The explanation reveals that the first three features are important and all three contribute towards 1.

2. The mean of $\psi_{i,j}$ samples (black points) reveals the magnitude and direction of the value's average influence. Standard deviation of $\psi_{i,j}$ samples (grey points) is also included for each value to reveal its global importance.

For an instance explanation, we repeat Algorithm 1 for each feature. To ensure with a certain probability that the approximated contribution will be within a certain distance from the actual contribution we require a constant number of samples. Therefore, for a given error the number of samples m needed to generate the explanation for a single feature does not increase with the number of features. The same applies to global visualizations, although the constant is larger because we repeat the process for each feature value we plot. The total running time for one explanation is: a constant $\times$ the number of features n $\times$ the model's prediction time complexity $\mathcal{T}(f(x))$. The methods time complexity is $O(n \cdot \mathcal{T}(f(x))$. For most regression algorithms $\mathcal{T}(f(x))$ is $O(n)$, which implies quadratic time complexity. Our purpose is to show that the method is a well-founded and useful tool, which can be used to generate explanations in real-time (order of seconds) for data sets with up to a few dozen features (already shown

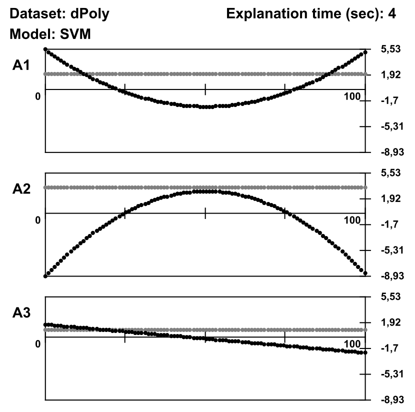

Fig. 7. SVM provides the best fit for *dPoly*. Subsequently, the contributions closely match the actual concepts.

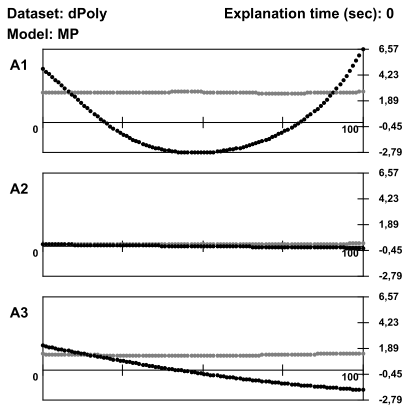

Fig. 8. The visualization shows us that MP learned some but not all of the concepts behind the *dPoly* data set.

in [8]). A more rigorous analysis of the limits of the method wrt the number of features it can handle for a given type of model is delegated to further work.

3 Experimental Verification

We have shown that the method is theoretically well founded and has several desirable properties. But how well does it translate into practice. The time complexity was discussed at the end of Section 2.1. Due length limits, we omit an in-depth analysis of running times in favor of showing more examples.

We tested the method using six different regression algorithms: linear regression (**LR**), a Support Vector Machine for regression (**SVM**), a multi-layer perceptron with a single hidden layer (**MP**), k-nearest neighbors (k = 1 and k = 11), a regression tree (**M5P**), and pace regression (**PR**). The method was implemented in Java using Weka's learning algorithm classes [1]. Default parameters were used, with the exception of SVM, where a 2nd degree polynomial kernel was used. A variety of models (in terms of performance and type) is desirable as we can verify if the explanations reveal why they performed well or poorly.

Artificial data allow us to test if explanations generated for good models are close to those generated for the optimal model (and vice versa). All feature values lie between 0.00 and 100.00, R is the target variable, each data set has 5 features and those that are not explicitly mentioned have no influence on R. Note that 1000 training and 1000 test samples were generated for each data set. Data sets: **dLinear** ($R = A_1 + 2A_2 + 3A_3$), **dRedund** ($R = 2A_1 - 2A_2$; A_3 always has the same value as A_2 to create a redundant feature), **dLocLin** (features A_3 and A_4 are binary and divide the problem space into 4 locally linear subproblems: $R = 5A_1 + A_2, \text{if} A_3 = 0 \wedge A_4 = 0; R = A_1 - 4A_2, \text{if} A_3 = 0 \wedge A_4 = 1; R = 2A_1 + 8A_2, \text{if} A_3 = 1 \wedge A_4 = 0; R = -2A_1 - 3A_2, \text{if} A_3 = 1 \wedge A_4 = 1$), **dTrig** ($R =$

Table 1. RRMSE and distances from the explanation for an optimal model (in parentheses). The correlation coefficients between the two are included for each data set.

	dLinear	dLocLin	dRedund	dTrig	dPoly	dDisj	dXor	dXorBin	dRand
LR	0.00	0.49	0.00	0.83	0.98	0.85	1.00	1.00	1.00
	(3.09)	(112.72)	(2.33)	(0.78)	(4.06)	(0.17)	(0.35)	(0.29)	(1.82)
MP	0.01	0.05	0.02	0.33	0.88	0.72	0.57	0.00	0.99
	(3.10)	(13.75)	(3.01)	(0.20)	(3.25)	(0.13)	(0.17)	(0.05)	(1.72)
SVM	0.01	0.13	0.01	0.50	0.13	1.05	0.81	1.60	1.00
	(3.08)	(24.24)	(2.78)	(0.33)	(0.67)	(0.33)	(0.26)	(0.81)	(3.19)
M5P	0.24	0.08	0.12	0.18	0.30	0.34	0.30	0.00	1.00
	(24.12)	(20.38)	(7.20)	(0.13)	(1.03)	(0.03)	(0.06)	(0.04)	(3.40)
KNN1	0.34	0.11	0.16	0.59	0.66	0.75	0.75	0.00	1.00
	(19.80)	(25.53)	(17.21)	(0.35)	(2.52)	(0.10)	(0.23)	(0.14)	(14.87)
KNN10	0.24	0.11	0.13	0.52	0.60	0.61	0.60	0.26	1.01
	(22.12)	(28.73)	(11.23)	(0.43)	(2.24)	(0.12)	(0.21)	(0.16)	(5.79)
PR	0.00	0.50	0.00	0.79	0.97	0.74	1.00	1.00	1.00
	(2.97)	(114.52)	(3.25)	(0.73)	(4.05)	(0.16)	(0.35)	(0.29)	(1.89)
coeff	**0.942**	**0.998**	**0.927**	**0.958**	**0.991**	**0.911**	**0.992**	**0.913**	**NA**

$sin(\frac{2\pi A_1}{100}) + cos(\frac{2\pi A_2}{100}))$, **dPoly** ($R = 2(\frac{A_1-50}{25})^2 - 3(\frac{A_2-50}{25})^2 - \frac{A_3-50}{25}$), **dDisj** ($R = 1$ if $(A_1 > 50) \vee (A_2 > 40) \vee (A_3 > 60)$; otherwise $R = 0$), **dXor** (an XOR problem, $R = (A_1 > 50)$ XOR $(A_2 > 50)$ XOR $(A_2 > 50)))$, **dXorBin** (similar to $dXor$, all five features are binary.,$R = A_1$ XOR A_2 XOR A_3), **dRand** (R is chosen at random).

First, we investigated if the generated contributions reflect what the model learns. We evaluated the models with the relative root mean squared error (RRMSE). For a distance measure[1] we used the Euclidean *distance* between the vector $(\varphi_1, ..., \varphi_n)$ and the vector generated when using optimal predictions instead of f. Table 1 shows the results for the described experiment. Some models perform better and some data sets are more difficult. Regardless, explanation quality and model performance are highly correlated.

Correlation is not applicable to *dRand*. All models should have a RRMSE of 1 (any deviations are due to noise). However, some models overfit, which results in explanations away from optimal. For example, KNN1 is likely to overfit. Figure 4 reveals that feature A_1 has a substantial influence on the KNN1 model, despite being useless for predicting R.

Results confirm that the explanations reflect, at least in an abstract sense, what the models have learnt. We continue by observing some examples and verifying whether the explanations are useful from a user's perspective. We start with instances from dDisj and dXorBin. Figures 5 and 6 are explanations for M5P on dDisj and MP on dXorBin, respectively. In the introduction we pointed out that these two concepts are representative of what existing general methods are unable to handle correctly. Visualizations show that the proposed method reveals the important features and their contributions.

[1] That is, to describe how much the explanations generated for a given model differ from those generated for an optimal model.

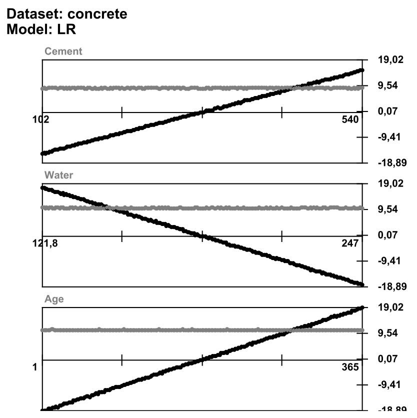

Fig. 9. LR is most influenced by cement, water, and age. Concrete strength increases with age and amount of cement and decreases with the amount of water.

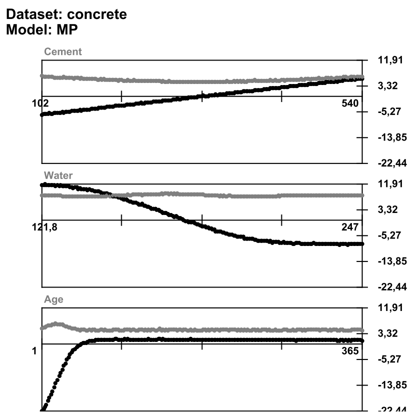

Fig. 10. Similar to LR, cement, water, and age are the most important for the neural network model. However, MP fits the non-linear relationships better.

Now we proceed to global visualizations[2]. The best model for *dPoly* was SVM. The explanation (Figure 7) confirms that it fits the data well. The worst were the linear models, which can not fit the polynomial. The MP model is somewhere in between and Figure 8 reveals why. The model learned only a part of the concept, missing the relevance of feature A_2.

We conclude the section with a more realistic example of what data mining practitioners encounter in practice. The *concrete* data set has 9 numeric features - concrete mixture components (in kg/m^3) and age (in days), and one target feature - compressive strength of the mixture (in MPa). The data were obtained from the UCI repository, where it was made available by prof. I-Cheng Yeh [2].

The compression strength is a highly non-linear problem [2]. Using LR and MP we achieved mean squared errors of 109 and 55, respectively, while predicting with the mean value results in a mean squared error of 279 (we used 10-fold cross-validation). The minimum, maximum, mean, and standard deviation of the compressive strength class variable are 2.3, 82.6, 35.8, and 16.706, respectively.

Figures 9 and 10 are visualizations for LR and MP. These are used to reveal the overall importance of individual features and their contribution to the model's predictions. When interested in a specific prediction, we observe the corresponding instance explanation. For example, Figure 11 is an instance explanation for MP's prediction for a particular concrete mixture. MP's prediction for is close to the actual concrete compressive strength, while LR overestimates the compressive strength for this instance and predicts 60 MPa. The explanation reveals which features contribute towards/against compressive strength.

[2] We left some irrelevant features out of the visualizations, to conserve space.

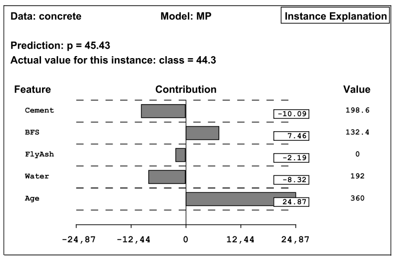

Fig. 11. For this particular prediction age contributes positively. The amount of water and cement have a negative contribution. Construction experts agree with the explanation and elaborate that the mixture suffers from a high water-to-cement ratio. Least important features were removed.

4 Conclusion

The proposed explanation method is simple to implement and can be applied to any regression model. It can explain both the model and its predictions. Results across different regression models and data sets confirmed that the method's explanations reflect what the models learn, even in cases where existing general explanation methods would fail. The examples presented throughout the paper illustrate that the method is a useful tool for visualizing models, comparing them, and identifying potential errors. With emphasis on the theoretical properties and the method's usefulness, less attention was given to measuring and optimizing running times. We delegate this to further work, together with an in-depth analysis of running times across different types of models.

References

1. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. SIGKDD Explorations 11(1), 1–3 (2009)
2. I-Cheng, Y.: Modeling of strength of high performance concrete using artificial neural networks. Cement and Concrete Research 28(12), 1797–1808 (1998)
3. Jakulin, A., Možina, M., Demšar, J., Bratko, I., Zupan, B.: Nomograms for visualizing support vector machines. In: KDD '05: ACM SIGKDD. pp. 108–117 (2005)
4. Lemaire, V., Fraud, R., Voisine, N.: Contact personalization using a score understanding method. In: IJCNN 2008 (2008)
5. Možina, M., Demšar, J., Kattan, M., Zupan, B.: Nomograms for visualization of naive bayesian classifier. In: PKDD 2004. pp. 337–348. Springer-Verlag (2004)
6. Poulet, F.: Svm and graphical algorithms: A cooperative approach. In: 4th IEEE ICDM. pp. 499–502 (2004)
7. Robnik-Šikonja, M., Kononenko, I.: Explaining classifications for individual instances. IEEE TKDE 20, 589–600 (2008)
8. Štrumbelj, E., Kononenko, I.: An efficient explanation of individual classifications using game theory. Journal of Machine Learning Research 11, 1–18 (2010)
9. Zien, A., Krämer, N., Sonnenburg, S., Rätsch, G.: The feature importance ranking measure. In: ECML PKDD 2009, Part II. pp. 694–709. Springer-Verlag (2009)