12-2016

A Modification of Backpropagation Enables Neural Networks to Learn Preferences

Martine Ceberio
University of Texas at El Paso, mceberio@utep.edu

Vladik Kreinovich
University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-16-102
To appear in *Journal of Uncertain Systems*

A Modification of Backpropagation Enables Neural Networks to Learn Preferences

Martine Ceberio and Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
mceberio@utep.edu, vladik@utep.edu

Abstract

To help a person make proper decisions, we must first understand the person's preferences. A natural way to determine these preferences is to learn them from the person's choices. In principle, we can use the traditional machine learning techniques: we start with all the pairs (x, y) of options for which we know the person's choices, and we train, e.g., the neural network to recognize these choices. However, this process does not take into account that a rational person's choices are consistent: e.g., if a person prefers a to b and b to c, this person should also prefer a and c. Since the usual learning algorithms do not take this consistency into account, the resulting choice-prediction algorithm may be inconsistent. It is therefore desirable to explicitly take consistency into account when training the network. In this paper, we show how this can be done.

1 Formulation of the Problem

Need to learn preferences. To help a person make decisions, we must first understand this person's preferences. Sometimes, a person can describe his or her preferences in precise terms. However, in many cases, a person cannot describe these preferences in precise terms, so we must elicit such preferences from him or her.

Elicitation such preferences is an important task in decision making; see, e.g., [5, 6, 8]. Elicitation such preferences is also an important part of recommender systems.

It is natural to use machine learning to learn preferences. The only way to learn a person's preferences is to provide this person with several pairs of options, record the person's preferences for all these pairs, and then use this information to predict how this person will react to other pairs.

Computers have been designed to process numbers. Computers are still much better in processing numbers than in processing any other type of information. Therefore, a reasonable way to describe each option is to describe it by a tuple of numbers $x = (x_1, \ldots, x_n)$, namely, as a tuple consisting of different numerical quantities that characterize this option.

The preference can also be described by some number z. For example, for each pair (x, y), we can use $z = 1$ if the person preferred x and $z = -1$ if the person preferred y.

In these terms, the problem of learning a person's preferences takes the following form:

- we are given a finite list of pairs of tuples (x, y) for each of which we know the preference z;
- we would like to use this information to predict the values z corresponding to all other pairs (x, y).

In this form, the problem becomes a particular case of the general machine learning problem; see, e.g., [2]. And indeed, machine learning techniques have been effectively used to elicit preferences.

Limitations to a straightforward application of machine learning. The above formulation does not take into account that for a rational person, preferences corresponding to different pairs are related to each other. Namely, if a person preferred x to y and y to u, then we expect this person to prefer x to u as well. In other words, preferences must be *transitive:* if $z(x, y) = z(y, u) = 1$, then we should have $z(x, u) = 1$.

The above formulation does not take this transitivity into account. As a result, at each stage of learning, we may have the current state of the learned function $z(x, y)$ to be non-transitive. This leads to the following two limitations:

- The fact that on the intermediate stages, we go through un-realistic functions before getting to the correct one makes the system wander more than needed and thus, take longer time to learn – and for machine learning techniques, learning time is, in general, rather long [2].
- It is also possible that some non-transitivity will remain for the learning result as well – in which case the resulting system clearly makes wrong predictions.

To speed up the learning process and to make its result more adequate, it is therefore desirable to modify the machine learning algorithms so that they explicitly take transitivity into account.

What we do in this paper. In this paper, we propose exactly such a modification. We explain our idea on the example of back-propagation neural networks – since as of now such networks are the most efficient machine learning tools.

Specifically, the efficient tools are deep networks, with a large number of layers. To simplify our exposition, we only provide the corresponding formulas for the simpler case of the traditional 3-layer networks; however, these formulas can be easily generalized to any number of layers.

2 Main Idea

General idea. A preference relation can be usually described by a function $f(x)$ such that x is preferred to y if and only $f(x)$ is larger than or equal to $f(y)$. We will therefore use a neural network not to learn $z(x,y)$, but instead, to learn the corresponding function $f(x)$.

This way, on each intermediate stage of learning, the corresponding relation $f(x) \geq f(y)$ is clearly transitive.

How can we implement this idea? To decide how to implement this idea, let us recall how neural networks work. If for a given neural network, for some example (x,y), the predicted value $z(x,y)$ is different from the observed value z, then we modify the parameters of the neural network so as to decrease this difference – and thus, get it closer to 0.

Similarly, in our case, if for some pairs (x,y), the person prefers x, but at the current stage of learning, with the learned-so-far function $f(x)$, we get $f(x) < f(y)$, then we should modify the parameters of the neural network so as to decrease the difference $f(y) - f(x)$ – and thus, get it closer to a desired negative value.

Of course, if the values of $f(x)$ and $f(y)$ are very close, a person may not notice the difference. So, it makes sense to say that if x is preferred to y, then not only we should have $f(x) \geq f(y)$, but the difference $f(x) - f(y)$ describing this preference should be larger than or equal to some positive threshold $\delta > 0$.

To describe the corresponding algorithm, let us recall the derivation of the usual back-propagation algorithm.

3 The Usual Back-Propagation Algorithm: A Brief Reminder

Main idea behind the usual back-propagation. The result of applying a 3-layer neural network to inputs $x_1, \ldots, x_n$ has the form

$$f(x) = \sum_{k=1}^{K} W_k \cdot X_k - W_0 \tag{1}$$

where K is the total number of neurons in the hidden layer, and the output of each of the K hidden neurons has the form

$$X_k = s_0\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right), \tag{2}$$

the activation function $s_0(z)$ has the form

$$s_0(z) = \frac{1}{1+\exp(-z)}, \tag{3}$$

and W_k and w_{ki} are parameters that need to be determined in the process of training the network.

For each tuple $x = (x_1, \ldots, x_n)$, we want to result $f(x)$ of applying the neural network to be as close to the observed value z as possible. A natural way to describe this requirement "to be as close as possible" is to minimize the square

$$J = (\Delta z)^2 \tag{4}$$

of the difference

$$\Delta z \stackrel{\text{def}}{=} f(x) - z. \tag{5}$$

The simplest way to minimize a function J is to use gradient descent, in which, for every parameter a, we replace its original value with the new value $a + \Delta a$, where

$$\Delta a = -\lambda \cdot \frac{\partial J}{\partial a}, \tag{6}$$

for some value λ.

This is exactly back-propagation. To be more precise, back-propagation is an algorithm that enables us to efficiently compute all these changes in parameters. Let us describe how this algorithm works.

From the main idea to exact formulas. Let us start with the parameter W_0. Because of the chain rule, we have

$$\frac{\partial J}{\partial W_0} = 2 \cdot \Delta z \cdot \frac{\Delta z}{\partial W_0}. \tag{7}$$

Here, $\Delta z = f(x) - z$, where

$$\frac{\partial f(x)}{\partial W_0} = -1$$

and z does not depend on W_0 at all. Thus,

$$\frac{\Delta z}{\partial W_0} = -1,$$

and the formula (7) takes the form

$$\frac{\partial J}{\partial W_0} = -2 \cdot \Delta z. \tag{8}$$

Thus,

$$\Delta W_0 = -\lambda \cdot \frac{\partial J}{\partial W_0} = 2 \cdot \lambda \cdot \Delta z. \tag{9}$$

This formula can be simplified if we denote $\alpha \stackrel{\text{def}}{=} 2\lambda$, then

$$\Delta W_0 = \alpha \cdot \Delta z. \tag{10}$$

Next, let is consider the parameter W_k. Here,

$$\frac{\partial J}{\partial W_k} = 2 \cdot \Delta z \cdot \frac{\Delta z}{\partial W_k} = 2 \cdot \Delta z \cdot \frac{\Delta f(x)}{\partial W_k} = 2 \cdot \Delta z \cdot X_k. \tag{11}$$

By comparing the formulas (8) and (11), we conclude that

$$\frac{\partial J}{\partial W_k} = -X_k \cdot \frac{\partial J}{\partial W_0}.$$

Multiplying both sides of this equality by $-\lambda$, we conclude that

$$\Delta W_k = -X_k \cdot \Delta W_0. \tag{12}$$

Let us now consider the parameter w_{k0}. In the formula for $f(x)$ – and thus, in the formula for $\Delta z = f(x) - z$ – the only term that depends on w_{k0} is the term X_k. Thus, we get

$$\frac{\partial J}{\partial w_{k0}} = \frac{\partial J}{\partial X_k} \cdot \frac{\partial X_k}{\partial w_{k0}}. \tag{13}$$

Here,

$$\frac{\partial J}{\partial X_k} = 2 \cdot \Delta z \cdot \frac{\partial f(x)}{\partial X_k} = 2 \cdot \Delta z \cdot W_k. \tag{14}$$

On the other hand, due to the chain rule,

$$\frac{\partial X_k}{\partial w_{k0}} = s_0' \left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0} \right) \cdot (-1). \tag{15}$$

By differentiating the function $s_0(z)$, we conclude that $s_0'(z) = s_0(z) \cdot (1 - s_0(z))$. Since here $s_0 \left(\sum\limits_{i=1}^{n} w_{ki} \cdot x_i - w_{k0} \right) = X_k$, the formula (14) takes the form

$$\frac{\partial X_k}{\partial w_{k0}} = -X_k \cdot (1 - X_k). \tag{16}$$

Substituting formulas (14) and (16) into the formula (13), we conclude that

$$\frac{\partial J}{\partial w_{k0}} = -2 \cdot \Delta z \cdot W_k \cdot X_k \cdot (1 - X_k). \tag{17}$$

By comparing the formulas (17) and (11), we conclude that

$$\frac{\partial J}{\partial w_{k0}} = \frac{\partial J}{\partial W_k} \cdot W_k \cdot (1 - X_k). \tag{18}$$

Multiplying both sides of this equality by $-\lambda$, we conclude that

$$\Delta w_{k0} = -W_k \cdot (1 - X_k) \cdot \Delta W_k. \tag{19}$$

Finally, let us consider each of the remaining parameters w_{ki}. Here,

$$\frac{\partial J}{\partial w_{ki}} = \frac{\partial J}{\partial X_k} \cdot \frac{\partial X_k}{\partial w_{ki}}, \tag{20}$$

where

$$\frac{\partial X_k}{\partial w_{ki}} = s_0' \left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0} \right) \cdot x_i. \tag{21}$$

By comparing the formulas (15) and (21), we conclude that

$$\frac{\partial X_k}{\partial w_{ki}} = -x_i \cdot \frac{\partial X_k}{\partial w_{k0}}. \tag{22}$$

Multiplying both sides of this equality by $\dfrac{\partial J}{\partial X_k}$, and taking into account formulas (20) and (13), we conclude that

$$\frac{\partial J}{\partial w_{ki}} = -x_i \cdot \frac{\partial J}{\partial w_{k0}}.$$

Multiplying both sides of this equality by $-\lambda$, we conclude that

$$\Delta w_{ki} = -x_i \cdot \Delta w_{k0}. \tag{23}$$

Thus, we arrive at the following algorithm.

Resulting formulas. We start with some values of W_k and w_{ki}.

Then, we process all the tuples x for which we know the desired result z one by one. The processing of each tuple consists of two stages.

First, we perform *forward computation:*

- first, for each k from 1 to K, we compute

$$X_k = s_0\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right); \tag{24}$$

- then, we compute $f(x) = \sum\limits_{k=1}^{K} W_k \cdot X_k - W_0$.

After that, we perform *backward computation*:

- first, we compute $\Delta z = f(x) - z$;
- then, we compute $\Delta W_0 = \alpha \cdot \Delta z$;
- after that, we compute $\Delta W_k = -X_k \cdot \Delta W_0$;
- then, we compute $\Delta w_{k0} = -W_k \cdot (1 - X_k) \cdot \Delta W_k$;
- after that, we compute $\Delta w_{ki} = -x_i \cdot \Delta w_{k0}$;
- finally, we update the values of the weights:

$$W_0^{\text{new}} = W_0 + \Delta W_0, \quad W_k^{\text{new}} = W_k + \Delta W_k,$$

$$w_{k0}^{\text{new}} = w_{k0} + \Delta w_{k0}, \text{ and } w_{ki}^{\text{new}} = w_{ki} + \Delta w_{ki}.$$

We repeat this two-stage procedure for every tuple.

Once we have cycled through all the tuples, we cycle through each tuple again and again – until the process converges, i.e., until for each tuple, the absolute value of the difference $f(x) - z$ is smaller than some small number δ.

4 A Modification of the Back-Propagation Algorithm Enabling It To Learn Preferences

Main idea. We would like the neural network to learn the person's preferences. Specifically, we would like to learn the person's objective function $f(x)$ for which x is preferred to y if and only if $f(x) - f(y) \geq \delta$.

As the input to the desired learning algorithm, we have a list of pairs of tuples (x, y) for which the person prefers x. If for this tuple, we have $f(x) - f(y) < \delta$, then we need to modify the parameters of the neural network so as to increase the difference $f(x) - f(y)$.

The results of applying a 3-layer neural network to the tuples $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ have the form

$$f(x) = \sum_{k=1}^{K} W_k \cdot X_k - W_0 \tag{24}$$

and

$$f(y) = \sum_{k=1}^{K} W_k \cdot Y_k - W_0, \tag{25}$$

where

$$X_k = s_0 \left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0} \right) \tag{26}$$

and

$$Y_k = s_0 \left(\sum_{i=1}^{n} w_{ki} \cdot y_i - w_{k0} \right). \tag{27}$$

The difference $J \stackrel{\text{def}}{=} f(x) - f(y)$ that we want to increase has the form

$$J = f(x) - f(y) = \sum_{k=1}^{K} W_k \cdot (X_k - Y_k). \tag{28}$$

We see that this difference does not depend on W_0. Thus, it make sense to ignore W_0, e.g., to take $W_0 = 0$. This makes sense since $f(x)$ is the objective function whose only purpose is to describe preferences, and adding a constant W_0 to the objective function does not change the corresponding order between the alternatives.

To increase the function J, we will use the gradient ascent, in which, for every parameter a, we replace its original value with the new value $a + \Delta a$, where

$$\Delta a = \lambda \cdot \frac{\partial J}{\partial a}, \tag{29}$$

for some value λ. Here, $J = f(x) - f(y)$, so

$$\frac{\partial J}{\partial a} = \frac{\partial f(x)}{\partial a} - \frac{\partial f(y)}{\partial a}. \tag{30}$$

Thus, the formula (29) takes the form

$$\Delta a = \Delta_x a - \Delta_y a, \tag{31}$$

where

$$\Delta_x a = \lambda \cdot \frac{\partial f(x)}{\partial a} \tag{32}$$

and

$$\Delta_y a = \lambda \cdot \frac{\partial f(y)}{\partial a}. \tag{33}$$

Let us show how, similarly to the usual back-propagation, we can efficiently compute all these changes in parameters.

From the main idea to exact formulas. Let us start with a parameter W_k. Here,

$$\frac{\partial f(x)}{\partial W_k} = X_k, \tag{34}$$

and thus,

$$\Delta_x W_k = \lambda \cdot X_k. \tag{35}$$

Similarly,

$$\Delta_y W_k = \lambda \cdot Y_k. \tag{36}$$

Let us now consider the parameter w_{k0}. In the expression for $f(x)$, the only term depending on w_{k0} is the term $W_k \cdot X_k$. Thus,

$$\frac{\partial f(x)}{\partial w_{k0}} = W_k \cdot \frac{\partial X_k}{\partial w_{k0}}. \tag{37}$$

From the formula (16) for the usual back-propagation, we know that

$$\frac{\partial X_k}{\partial w_{k0}} = -X_k \cdot (1 - X_k), \tag{37}$$

and thus,

$$\frac{\partial f(x)}{\partial w_{k0}} = -W_k \cdot X_k \cdot (1 - X_k). \tag{38}$$

Comparing the expressions (38) and (34), we conclude that

$$\frac{\partial f(x)}{\partial w_{k0}} = -W_k \cdot (1 - X_k) \cdot \frac{\partial f(x)}{\partial W_k}. \tag{39}$$

Multiplying both sides of this equality by λ, we conclude that

$$\Delta_x w_{k0} = -W_k \cdot (1 - X_k) \cdot \Delta_x W_k. \tag{40}$$

Similarly,

$$\Delta_y w_{k0} = -W_k \cdot (1 - Y_k) \cdot \Delta_y W_k. \tag{41}$$

Finally, let us consider each of the remaining parameters w_{ki}. Here,

$$\frac{\partial f(x)}{\partial w_{ki}} = W_k \cdot \frac{\partial X_k}{\partial w_{k0}}. \tag{42}$$

From the formula (22), we know that

$$\frac{\partial X_k}{\partial w_{ki}} = -x_i \cdot \frac{\partial X_k}{\partial w_{k0}}. \tag{43}$$

Multiplying both sides of this equality by W_k and taking into account formulas (42) and (37), we conclude that

$$\frac{\partial f(x)}{\partial w_{ki}} = -x_i \cdot \frac{\partial f(x)}{\partial w_{k0}}. \tag{44}$$

Multiplying both sides of this equality by λ, we conclude that

$$\Delta_x w_{ki} = -x_i \cdot \Delta_x w_{k0}. \tag{45}$$

Similarly,

$$\Delta_y w_{ki} = -y_i \cdot \Delta_y w_{k0}. \tag{46}$$

Thus, we arrive at the following algorithm.

Resulting formulas. We start with some values of W_k and w_{ki}.

Then, we process all the pairs of pairs (x, y) for which we know that the person prefers x to y. The processing of each pair consists of two stages.

First, we perform *forward computation:*

- first, for each k from 1 to K, we compute

$$X_k = s_0\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) \tag{47}$$

  and

$$Y_k = s_0\left(\sum_{i=1}^{n} w_{ki} \cdot y_i - w_{k0}\right); \tag{48}$$

- then, we compute $f(x) = \sum\limits_{k=1}^{K} W_k \cdot X_k$ and $f(y) = \sum\limits_{k=1}^{K} W_k \cdot Y_k$.

After that, if $f(x) - f(y) < \delta$, we perform *backward computation*:

- first, for each k, we compute $\Delta_x W_k = \lambda \cdot X_k$ and $\Delta_y W_k = \lambda \cdot Y_k$;
- then, we compute

$$\Delta_x w_{k0} = -W_k \cdot (1 - X_k) \cdot \Delta_x W_k \text{ and } \Delta_y w_{k0} = -W_k \cdot (1 - Y_k) \cdot \Delta_y W_k;$$

- after that, we compute $\Delta_x w_{ki} = -x_i \cdot \Delta_x w_{k0}$ and $\Delta_y w_{ki} = -y_i \cdot \Delta_y w_{k0}$;
- finally, we update the values of the weights:

$$W_k^{\text{new}} = W_k + \Delta_x W_k - \Delta_y W_k,$$

$$w_{k0}^{\text{new}} = w_{k0} + \Delta_x w_{k0} - \Delta_y w_{k0}, \text{ and } w_{ki}^{\text{new}} = w_{ki} + \Delta_x w_{ki} - \Delta_y w_{ki}.$$

We repeat this two-stage procedure for every pair of tuples.

Once we have cycled through all the pairs, we cycle through each pairs again and again – until the process converges.

Comment. Ideally, we should cycle until we get $f(x) - f(y) \geq \delta$ for all the pairs. However, a person may have been somewhat inconsistent, and there may be situations in which this person preferred x to y, y to u, and u to x. In such cases, it is not possible to have a function $f(x)$ for which always x is preferred to y if $f(x) > f(y)$. With this possibility in mind, it is better to stop the iterations when the weights W_k and w_{ki} stop changing, i.e., when the values of the weights at the end of the cycle are sufficiently close to the values of these weight at the beginning of this cycle.

5 Alternative Algorithms

Idea. Let $a_1, \ldots, a_m$ be quantities that characterize each alternative. We want to describe an objective function $f(a_1, \ldots, a_m)$ that describes the person's preferences: a is better than b if and only if $f(a) > f(b)$.

In the previous section, we described a general neural network-based algorithm for finding this objective function. An alternative idea is to take into account that usually, the dependence of the objective function on the quantities a_i is smooth. Thus, we can expand the unknown function in Taylor series and keep only small-order terms in this expansion. For example, if we only keep linear terms, we get a general expression $f(a) = c_0 + \sum\limits_{i=1}^{m} c_i \cdot a_i$. If we keep quadratic terms, we get an expression

$$f(a) = c_0 + \sum_{i=1}^{m} c_i \cdot a_i + \sum_{i \leq j} c_{ij} \cdot a_i \cdot a_j,$$

etc. In general, we get an expression of the type

$$f(x) = \sum_{i=1}^{n} C_i \cdot x_i, \tag{49}$$

where x_i are the corresponding monomials:

- expressions a_i in the linear case,
- expressions a_i and $a_i \cdot a_j$ in the quadratic case, etc.

In this case, to determine the objective function, we need to find the values of the corresponding parameters C_i.

In terms of the objective function (49), the condition $f(x)-f(y) \ge \delta$ becomes a linear inequality

$$\sum_{i=1}^{n} C_i \cdot (x_i - y_i) \ge \delta. \tag{50}$$

Thus, we can use linear programming [7, 9] – a known method for solving systems of linear inequalities – to find the corresponding values C_i. Thus, we arrive at the following algorithm.

First alternative algorithm. Once we have listed the quantities $a_1, \ldots, a_m$ that describe each alternative, we select an order $d \ge 1$, and describe each alternative by the values $x_1, \ldots, x_n$ of all possible monomials $x_i = a_{j_1} \cdot \ldots \cdot a_{j_q}$ of order $q \le d$ in terms of the quantities a_j; here, $j_1 \le \ldots \le j_q$.

To each pairs (x, y) for which x was preferred to y, we form a linear inequality

$$\sum_{i=1}^{n} C_i \cdot (x_i - y_i) \ge \delta, \tag{50}$$

with unknown values C_i. We then use linear programming to find the values $C_1, \ldots, C_n$ that satisfy all these inequalities; see, e.g., [3, 4].

Once the values C_i are found, we predict that an alternative x will be preferred to an alternative y if the inequality (50) holds.

Need to go beyond the first alternative algorithm. The above first alternative algorithm assumes that the person always makes rational preferences. In real life, as we have mentioned earlier, people sometimes make inconsistent choices. In this case, it is not possible to find the coefficients C_i for which all the inequalities (50) will be satisfied.

To deal with such realistic situations, we can use the gradient ascent approach similar to the one that we use in the neural networks case. For the expression

$$f(x) - f(y) = \sum_{i=1}^{n} C_i \cdot (x_i - y_i),$$

the gradient ascent method takes the form

$$C_i \to C_i + \lambda \cdot \frac{\partial J}{\partial C_i} = \lambda \cdot (x_i - y_i).$$

Thus, we arrive at the following algorithm.

Second alternative algorithm. We start with some values of the parameters $C_1, \ldots, C_n$.

Then, we process all the pairs of tuples (x, y) for which we know that the person prefers x to y. For each pair, if

$$\sum_{i=1}^{n} C_i \cdot (x_i - y_i) < \delta,$$

then we replace each value C_i with the new value

$$C_i^{\text{new}} = C_i + \lambda \cdot (x_i - y_i).$$

Once we have cycled through all the pairs, we cycle through each pairs again and again – until the process converges, i.e., until the values C_i do not change much from the end of one cycle to the end of another cycle.

Third alternative algorithm: using linear discriminant analysis. We would like to find the coefficients C_i for which $C \cdot (x - y) > 0$ for all pairs for which x is preferred to y, i.e., where we denoted $C = (C_1, \ldots)$, $x = (x_1, \ldots)$, $y = (y_1, \ldots)$, and $a \cdot b \stackrel{\text{def}}{=} a_1 \cdot b_1 + a_2 \cdot b_2 + \ldots$

Similarly, we should have $C \cdot (y - x) < 0$ for all such pairs (x, y). From the mathematical viewpoint, this problem is similar to the *linear discriminant analysis* (see, e.g., [1, 3, 4]), when we have two sets $\mathcal{S}$ and $\mathcal{S}'$ and we need to find a hyperplane that separates them, i.e., a vector C such that $C \cdot S \geq 0$ for all $S \in \mathcal{S}$ and $A \cdot S' \leq 0$ for all $S' \in \mathcal{S}'$. In our case, S is the set of all vectors $x - y$, and S' is the set of all vectors $y - x$.

The standard way of solving this problem is to compute the mean μ of all the vectors $S \in \mathcal{S}$, the covariance matrix Σ, and then to take $C = \Sigma^{-1}\mu$. So, in our case, we should do the following:

- compute all the vectors $x - y$ corresponding to the pairs (x, y) in which the person preferred x to y; let M be the total number of such pairs;
- compute the average $\mu = \dfrac{1}{M} \cdot \sum (x - y)$ of these vectors;
- compute the corresponding covariance matrix Σ with components

$$\Sigma_{ab} = \frac{1}{M} \cdot \sum_x (x_a - y_a - \mu_a) \cdot (x_b - y_b - \mu_b);$$

- compute the vector C formed by the desired coefficients C_i as $C = \Sigma^{-1}\mu$, i.e., as a solution to a linear system $\Sigma C = \mu$.

Acknowledgments

This work was supported by the National Science Foundation grants HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721, and by an award "UTEP and Prudential Actuarial Science Academy and Pipeline Initiative" from Prudential Foundation.

References

[1] A. Afifi and S. May, *Practical Multivariate Analysis*, Chapman & Hall/CRC, Boca Raton, Florida, 2011.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.

[3] S. M. Escarzaga, C. Tweedie, O. Kosheleva, and V. Kreinovich, “How to predict nesting sites and how to measure shoreline erosion: fuzzy and probabilistic techniques for environment-related spatial data processing”, *Proceedings of the 2016 World Conference on Soft Computing*, Berkeley, California, May 22–25, 2016.

[4] S. M. Escarzaga, C. Tweedie, and V. Kreinovich, “How to predict nesting sites?”, *Journal of Uncertain Systems*, 2017, Vol. 11, to appear.

[5] P. C. Fishburn, *Utility Theory for Decision Making*, John Wiley & Sons Inc., New York, 1969.

[6] R. D. Luce and R. Raiffa, *Games and Decisions: Introduction and Critical Survey*, Dover, New York, 1989.

[7] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*, Springer, Cham, Switzerland, 2016.

[8] H. Raiffa, *Decision Analysis*, Addison-Wesley, Reading, Massachusetts, 1970.

[9] P. J. Vanderbei, *Linear Programming: Foundations and Extensions*, Springer, New York, 2014.