# Expert Data Mining:

## Violations of Monotonicity

## Contents

Introduction ........ 2
Input ........ 2
Violation of the Monotonicity Assumption of the Domain Expert ........ 2
  One Below Upper Zero ........ 3
  Zero Above Upper One ........ 3
  Both ........ 4
Violation of Monotonicity in Real Data ........ 4
Fixing Violations ........ 4
  Changing Class ........ 5
    Monotonicity Preservation ........ 5
    Monotonicity Reaffirmation ........ 5
    Examples of Fixing Violations ........ 6
  Fix by Adding Attributes ........ 7
    Example of Adding Attributes ........ 8
  Non-Monotonic Function ........ 9
Violation of Monotonicity in Real Data ........ 9
Output ........ 10
Examples ........ 10

## Introduction

This document will describe how the Expert Data Mining program works with violations of monotonicity. Note: due to the complexity of the program, the previous README is also attached.

## Input

The input of the program is the same as its previous iteration. However, once all questions are asked, the program will ask the user to specify where any violations of monotonicity in the outputted dataset should be. Before going forward, it is important to note that the expert dataset does not contain violations of monotonicity due to the user assumption that the data should be monotonic for the given attributes. The user was asked some questions, but the remaining questions were answered through an expansion of monotonicity. However, a user can review these expanded values to find some expansion that does not correspond to his/her knowledge of the domain or on real experimental data. Once reviewed, the user may change the answer from one to zero or zero to one so that the answer corresponds to his/her knowledge or real data. Therefore, the user is essentially indicating where violations of monotonicity *should* occur in the expert dataset.

Where the description above is about the user's intention to change values of monotonically expanded values of cases, it is also possible that the user, after reviewing all the cases, would like to change his/ her answers to some of the cases on the assumption that monotonicity not violated at all. In this scenario, the values of the cases are updated. The user may need to answer some additional questions about any cases that were expanded by the case whose value was changed. This is because those other cases will no longer be monotonically related to the changed case.

Furthermore, there are two forms of violations of monotonicity: violation of **the monotonicity assumption of the expert**, and violation of **monotonicity in real data**. First, we will consider the former.

## Violation of the Monotonicity Assumption of the Domain Expert

A violation of the monotonicity assumption is a violation that occurs in the expert data set. This violation occurs because the data were assumed to be monotone, but there are case(s) where it is not. For example, the monotonicity will require having the expert answered that F(100) = 1 also to have F(101) = 1 by monotonicity expansion. In fact, a user can say that despite F(100) = 1 but F(101) = 0.

Once the expert dataset has been created and all classes are assigned, the user can select a case or cases where the values need to be changed. It can be a misassigned class or a violation of monotonicity. For example, an upper zero can be changed to a one and a lower one can be changed to a zero without a violation of monotonicity occurring. Let's look at the two cases,

101 = 1

100 = 0

The user can change (101), the lower one to have a value of 0. This case is simply called an <u>update of monotonicity</u> instead of a <u>violation of monotonicity</u> because the monotone Boolean function is simply updated.

Furthermore, there are two types of violation of the monotonicity assumptions: "one below upper zero" and "zero above upper one."

## One Below Upper Zero

For this type of violation, the source of the violation causes a violation between itself and the case above it. For example, we have the three cases:

1110 = 1

1010 = 0

1000 = 1

The violation of monotonicity is the two highlighted cases. However, we can say that the source of the violation is due to the case (1000) because it is the reason why the case above it breaks the property of monotonicity. The upper two cases, which are adjacent, do not form a violation of monotonicity. For that reason, this situation is called a "one below upper zero."

## Zero Above Upper One

For this type of violation, the source of the violation causes a violation between itself and the case below it. For example, we have the three cases:

1110 = 0

1010 = 1

1010 = 1

The violation of monotonicity is the two highlighted cases. The can say that the source of the violation is (1110) because it breaks the property of monotonicity. The two other cases, which are adjacent to each other, are monotonically sound.

### Both

The above situations are not mutually exclusive. For example:

> 1111 = 1
>
> 1110 = 0
>
> 1010 = 1
>
> 1000 = 1

It is unclear if it we can say which are the cases that are a violation of monotonicity is the source of the violation. The zero is not above the upper one. The one is below the upper zero, but since there are more ones below it, it is hard to say if that upper zero should be there or not. In these situations, what we call the situation would be up to how we decide to rectify it, if we rectify it at all.

## Violation of Monotonicity in Real Data

A violation of monotonicity in real data occurs when a case in the real dataset does not match the class of the corresponding case in the expert dataset. Of course, this also does not strictly have to be a violation. For example, in the real dataset, the upper zero should actually be the lower one. This case is simply called a mismatch.

There are two options presented to the user to fix violations of monotonicity in the real data: to apply the monotone Boolean function that was restored from the expert dataset, or to indicate that the case itself is wrong. Then, an attribute of that case will be changed. In the case that there is a single case in the expert dataset that has the same class and is directly monotonically related (one level apart), then it is assumed that the expert case will replace the real case. If this is not true, then there would be an additional violation of monotonicity that was missed. If there are no or multiple cases with the same class and are directly monotonically related, then the user will have to manually specify which attribute in the case to change.

## Fixing Violations

To fix violations of monotonicity, the user will be asked to either change the prior answer to the currently assigned, or to add a new attribute to the dataset.

## Changing Class

When a user chooses to change the class of the case, then the class will be changed to the opposite of what it was originally given. At this stage, in the expert dataset, it is important to note that the violation of monotonicity *has been* made. Remember that there are no violations in the expert dataset in the first place. Next, monotonicity has to be reinstated to fix the violation. There are two methods of doing this. First, the user is asked if they would like to preserve monotonicity. If the answer is yes, then monotonicity preservation occurs. If the answer is no, then the monotonicity reaffirmation occurs.

## Monotonicity Preservation

The first method is monotonicity preservation (a user said yes to preserving monotonicity). Due to the assumption of monotonic dependencies between *n*-D points, the change of the class in the specified case will change the classes of some other *n*-D points which are monotonically related to it: children, parents, grandparents, grandchildren and so on. This means that cases will have to be re-asked if they were expanded and were monotonically related to the case whose class has been changed. This is due to the change of class in the specified case: the source of expansion does not exist anymore for those monotonically related and expanded cases. Some questions may not have to be re-asked if a true attribute was given and the related case does not contain that true attribute (the related case would be false in this case).

## Monotonicity Reaffirmation

The second method is monotonicity reaffirmation (a user said no to preserving monotonicity). An adjacent case to the source of the violation of monotonicity will need to be checked if it was asked before. For a source of a violation of monotonicity that has a class of zero, then the adjacent case is the case below it. For a source of a violation of monotonicity that has a class of one, then the adjacent case is the case above it. Monotonicity must be reaffirmed in the direction of possible expansion. This is because the values of the case in the opposite direction, no matter if the value is one or zero, will not cause a violation of monotonicity. For example, let's look at two cases:

101 = 0

100 = 0 (original); 1 (changed due to violation)

000 = 0

If the user specifies that there is a violation in (100), then F(100) = 1. The adjacent case that needs to be checked is the case above it, (101), because the case below the violation, (000), can be of any value and not cause a violation. However, the case above it must be checked because if the user changes the case above it to one as well, then there was simply an update of monotonicity. Otherwise, a violation truly occurred and either attributes must be added or a non-monotonic function must be used.

Furthermore, if the adjacent case was asked before, it is assumed that the user is confident in that answer, so the user will be asked to add a new attribute(s), or else the restored function will be a non-monotonic function (more on these in subsequent sections). If the adjacent case was expanded by something else, it is assumed that the class is now in question. Therefore, the user is asked to give the class to this adjacent case. If the new class solves the violation of monotonicity, then monotonicity preservation will be used. If the new class reaffirms the violation of monotonicity, the user will be asked to add a new attribute(s), or else the restored function will be a non-monotonic function (more on these in subsequent sections).

## Examples of Fixing Violations

For the subsequent examples, the "Given" column are the cases as they originally were in the expert dataset. The red cases are cases which were specified to have violations of monotonicity. The "Situation" columns are the results of attempting to fix the monotonicity by using the Changing Class method.

*Example 1*

| Given | Situation 1 | Situation 2 | Situation 3 |
|---|---|---|---|
| F(110) = 1 | F(110) = 0 | F(110) = 0 | F(110) = 1 |
| F(100) = 1 | F(100) = 1 | F(100) = 0 | F(100) = 0 |

In the first situation, the violation was specified in the first case of the chain. The user did not preserve monotonicity, change the class of the adjacent case and did not add new attributes, so a non-monotonic function must be used. In the second situation, the violation was also specified in the first case of the chain. The user preserved monotonicity, so adjacent case's class was changed. In the third situation, the violation was specified in the last case of the chain. This is simply an update of monotonicity, so nothing else is done.

*Example 2*

| Given | Situation 1 | Situation 2 | Situation 3 | Situation 4 |
|---|---|---|---|---|
| F(110) = 1 | F(110) = 0 | F(110) = 0 | F(110) = 1 | F(110) = 0 |
| F(100) = 0 | F(100) = 1 | F(100) = 0 | F(100) = 1 | F(100) = 1 |

In the first situation, the violation was specified in the first case of the chain. The user did not preserve monotonicity, changed the class of the adjacent case, and did not add new attributes, so a non-monotonic function must be used due to the violation of monotonicity. In the second situation, the violation was also specified in the first case of the chain. The user preserved monotonicity, so the adjacent case's class was kept the same, so it ended up being an update of monotonicity. In the third situation, the violation was specified in the last case of the chain. The user preserved monotonicity, so the adjacent case's class was kept the same, so it ended up being an update of monotonicity, just like the previous situation. In the last situation, the user specified that the violation was in the second case of the chain. The user did not preserve monotonicity and did not change the answer to the case above it, so a non-monotonic function must be used due to the violation of monotonicity.

*Example 3*

| Given | Situation 1 | Situation 2 | Situation 3 |
|---|---|---|---|
| F(110) = 0 | F(110) = 1 | F(110) = 1 | F(110) = 0 |
| F(100) = 0 | F(100) = 0 | F(100) = 1 | F(100) = 1 |

In the first situation, an update was specified in the first case of the chain. There was no violation. In the second situation, an update was specified in the second case of the chain. The user preserved monotonicity or answered "1" to the question about the case above it, so this ended up being an update of monotonicity. In the third situation, the violation was specified in the second case of the chain. The user did not preserve monotonicity and did not change the answer to the case above the violation, so a non-monotonic function must be used due to the violation of monotonicity.

## Fix by Adding Attributes

In the case that the user decides to add a new attribute, then new Hansel Chains must be generated. Before asking any new questions about the new chains, it is possible to expand some of the new n-D points with the old **n-D points**.

Here we have can have two situations:

> **Situation 1**: Let F(**a**) = 1 and F(**b**) = 1, where **b** is the next n-D point greater than **a**. Assume there is a user told us that the F(**a**) = 1 is incorrect and change it to F(**a**`) = 0 with adding a new attribute $x_{n+1}$. For example, an old **a** = (101) and F(**a**) = 1 with a new class of 0 will expand to a new n-D point **a**` = (1010) and F(**a**`) = 0.   This change of value does not create a violation of monotonicity but still changes the function, which needs to be adjusted for all *n*-D points that are monotonically related to this point.

**Situation 2:** Let F(**a**) = 0 and F(**b**) = 0, where **b** is the next n-D point greater than **a**. Assume the user told us that the F(**a**) = 0 is incorrect, changing it to F(**a`**) = 1 by adding a new attribute $x_{n+1}$. For example, an old **a** = (101) and F(**a**) = 0 with a new class of 1 will expand to a new n-D point **a`** = (1011) and F(**a`**) = 1. This creates a violation of monotonicity. To remove a violation of monotonicity the algorithm will assign F(**b**) = 1. It will be conducted not only for this **b** but for all n-D points, which are greater than **a**, meaning for all n-D points that are monotonically related to **a.**

The two situations are also used to expand from old cases to the new cases in the new expert dataset by using the same process.

Furthermore, real data can be input as well. The "filename" variable at line 81 of the header file gives the name of the required dataset file, which is assumed to be a CSV. The first line of the CSV gives the name of the attributes, and it is assumed that the last column is the class column (otherwise, there is no point in including real data). Each case will have a 1 appended to it, as well as a duplicate with a 0 appended to it, and this repeats for however many new attributes there are. The real data is doubled in count for each new attribute added.

### Example of Adding Attributes

Let's look at a dataset with a small dimension of just two.

11 = 1

10 = 1

01 = 0

00 = 0

Now we will add an attribute, so

111 = 1

110 = ?

101 = 1

100 = ?

011 = ?

010 = 0

001 = ?

000 = 0

Notice that the attribute that was added corresponds to the class of the case. In the situation that there is no correspondence between the attribute that was added and the class, we cannot expand by monotonicity, so the user must be asked  questions about those cases which have a question mark for its class.

### Non-Monotonic Function

If the user does not add new attributes and the changing class method did not work, then there is no possibility of having a monotone function. Therefore, an "AND NOT" clause must be added to the function, where the "NOT" represents a violation of monotonicity. For example, we have two lower ones:

100 = 1

010 = 1

The monotone Boolean function that corresponds to these two lower ones is "x1 v x2." However, let's say that there was an unresolved violation of monotonicity in the chain that corresponds to the second case. The violation was a "zero above a lower one" and it was F(110) = 0. Since the previous function does not apply to the violation (and the violation is the only case where the function does not work, the function can be modified to account for the violation. The new function is "x1 v (x2 & !x1x2)." The "x1x2" corresponds to the case (110), and there is a not symbol before "x1x2" because the function is only true for "x2" when "x1" is not true. In the NOT clause, the "x2" is not necessary, but it is included for clarity of what the violation stands for.

## Violation of Monotonicity in Real Data

Moreover, for violations of monotonicity that the user wants to rectify with the latest class that was assigned, these can be immediately fixed. However, should the user want to add a new attribute, we cannot do this immediately because there may be more violations of monotonicity that would be "lost" due to creating a new data set. They would be lost because those cases would not exist anymore. Therefore, if the user decided to add attributes to fix the monotonicity, then they will be done after finding the other violations. Plus, multiple attributes can be added by using this method.  The only caveat with this method is that the "Order of Questions" and "Total Questions" columns will be for the questions asked about the new cases in the new dataset. Order of Questions" will show the sequence for asking the questions that determine if there is a violation of monotonicity, and "Total Questions" will be the number of questions in that sequence. However, this difference is not a problem because the middlemost case is always used, or the one above it in cases of Hansel Chains with an odd number of cases.

## Output

The output of the program is very similar to its previous iteration, except there are two tables.

The first table is the same as before, but the second table is a trimmed down table that shows the results of rectifying violations of monotonicity. Several irrelevant columns were taken out (such as the columns for ordering because the ordering for the rectification is the same every time). An additional column was added, “Monotonicity Fixed,” which signifies if a case had a violation of monotonicity that was fixed. This also applies even when a new attribute was added because case **a** goes to **a`** as showed in situation 1 and 2.

Also, a new version of the real dataset, if applicable will be printed to a different file. The name is simply a concatenation of “new_” and the given filename variable.

## Examples

```
Enter the class for this data point:
a                       = true (1)
b                       = true (1)
c                       = false (0)
d                       = true (1)
e                       = false (0)
Enter Class: 1
Your answer was different for this question this time: 1 instead of 0.
Do you want to add a new attribute to the dataset or decide which answer is correct? Enter (1/0): 1
Does the current attribute need a change of class? (1/0):
```

When checking a violation of monotonicity, this is the dialogue that will occur. If a violation is detected, then the program will prompt the user to enter “1” to add a new attribute or “0” to use the value which they last entered as the new class. If the user adds a new attribute, then the program will ask questions on the new cases of the new dataset in the same format that has always been done. This interaction is the only new interaction in the program.

Here is an example of the secondary table:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Results After Resolving Any Violations of Monotonicity | | | | | | | | |
| Monotone Boolean Function Simplified: x1x2x4 v x2x3x5 v x4x5 | | | | | | | | |
| Monotone Boolean Function Non-simplified:x1x2x4 v x1x2x3x4 v x2x3x5 v x2x4x5 v x1x4x5 v x4x5 | | | | | | | | |
| Order of C | 1.2 | 2.2 | 3.2 | 4.2 | 5.2 | 6.3 | 7.3 | 8.3 |
| Answers: | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Total Questions: 10 | | | | | | | | |
| | | | | | | | | |
| Reference | Vector | Class | Was Fixed | Expanded By | Expanded 1-1 | Expanded 0-0 | Expandable 1-1 | Expandable 0-0 |
| 1.1 | 0;1;0;1;0 | 0 | 0 | | | 6.1;8.1; | 1.2;6.3;8.3; | 6.1;8.1; |
| 1.2 | 1;1;0;1;0 | 1 | 1 | | 6.4; | | 6.4;8.4; | 1.1;3.1;5.1; |
| 2.1 | 0;1;1;0;0 | 0 | 0 | | | 7.1; | 2.2;6.3;7.3; | 7.1;8.1; |
| 2.2 | 1;1;1;0;0 | 0 | 0 | | | 4.1; | 6.4;7.4; | 2.1;4.1;5.1; |
| 3.1 | 1;0;0;1;0 | 0 | 0 | 1.2 | | 9.1; | 1.2;3.2;9.3; | 6.1;9.1; |
| 3.2 | 1;0;1;1;0 | 0 | 0 | | | 6.2; | 6.4;9.4; | 6.2;3.1;4.1; |
| 4.1 | 1;0;1;0;0 | 0 | 0 | 2.2 | | | 2.2;3.2;4.2; | 7.1;9.1; |
| 4.2 | 1;0;1;0;1 | 0 | 1 | | | | 7.4;9.4; | 7.2;9.2;4.1; |

Needless to say, the “Class,” “Expanded By,” “Expanded 1-1,” and “Expanded 0-0” columns will be different than the table above if there were any violations of monotonicity.

Finally, there are several results files that are attached. “results_resolved.csv” is fixing a violation of monotonicity without adding any new attributes, “results_resolved_add_1.csv” is fixing a violation of monotonicity by adding one attribute, and “results_resolved_add_2.csv” is fixing two violations of monotonicity by adding two attributes. These files were included in the last iteration of this README as well. Additional files are a sample dataset with just two cases and its modified version after being fixed.

```
Checking for violations of monotonicity:

Number          Datapoint               Class
1.1             0, 1, 0, 1, 0           0
1.2             1, 1, 0, 1, 0           0
2.1             0, 1, 1, 0, 0           0
2.2             1, 1, 1, 0, 0           0
3.1             1, 0, 0, 1, 0           0
3.2             1, 0, 1, 1, 0           0
4.1             1, 0, 1, 0, 0           0
4.2             1, 0, 1, 0, 1           1
5.1             1, 1, 0, 0, 0           0
5.2             1, 1, 0, 0, 1           0
6.1             0, 0, 0, 1, 0           0
6.2             0, 0, 1, 1, 0           0
6.3             0, 1, 1, 1, 0           0
6.4             1, 1, 1, 1, 0           0
7.1             0, 0, 1, 0, 0           0
7.2             0, 0, 1, 0, 1           0
7.3             0, 1, 1, 0, 1           1
7.4             1, 1, 1, 0, 1           1
8.1             0, 1, 0, 0, 0           0
8.2             0, 1, 0, 0, 1           0
8.3             0, 1, 0, 1, 1           1
8.4             1, 1, 0, 1, 1           1
9.1             1, 0, 0, 0, 0           0
9.2             1, 0, 0, 0, 1           0
9.3             1, 0, 0, 1, 1           1
9.4             1, 0, 1, 1, 1           1
10.1            0, 0, 0, 0, 0           0
10.2            0, 0, 0, 0, 1           0
10.3            0, 0, 0, 1, 1           1
10.4            0, 0, 1, 1, 1           1
10.5            0, 1, 1, 1, 1           1
10.6            1, 1, 1, 1, 1           1
Please enter the number of any vectors which are a violation of monotonicity in a comma-separalited list (e.g. 1.1, 3.2, ..., 7.4): 1.1, 7.2

Do you want to fix the violation by adding a new attribute to the expert dataset or by changing the class? Enter (1/0):0
Do you want to fix the violation by adding a new attribute to the expert dataset or by changing the class? Enter (1/0): 0
```

In the above photo, the user specified two violations of monotonicity, 1.1 and 7.2. The user decided to change the class of the two cases as shown in the bottom two lines.

```
Enter the class for this data point:
a                       = false (0)
b                       = false (0)
c                       = false (0)
d                       = false (0)
e                       = true (1)
Enter Class: 0
Does the user want to change the attributes of the real data or to apply the Boolean function to the real data? (1/0): 0
```

In this photo, the last few lines of console print-outs are given. There was one last question that was asked due to it being expanded, but its parent’s class was changed. Then, the user specifies that they want to apply the Boolean function to the real data instead of changing the attributes of the real data.