Artificial Intelligence for Modelling Dengue Transmission in Bello

Elisabet Lobo-Vesga
Olga-Lucía Quintero-Montoya

December 2014

Data

The main idea of this work is to establish a geographic (and temporal) relationship between breeding sites of *Ae. aegypti* and cases of Denge Fever (DF) in Bello. Then we have the coordinates and week occurrence of the breeding sites (578 observations) and the cases (1667 observations) from 2008 to 2012.

Note: The data of breeding sites in 2009 are not available.

Neighbors

Initially, trying to figure out how the breeding sites and the cases are related we created the neighbors algorithm. This algorithm takes a breeding site data (Longitude, Latitude and Week) and determines which cases could be occasioned by it depending on a radius and a week delay. This is what we call "The neighbors of a breeding site".

Biologically, the infection radius and the delay week fluctuate according to the *El Niño* phenomenon. To simulate such behavior we determined yearly the neighbors of every breeding site changing the delay from 3 to 6 weeks and the radius from 50 to 200 meters. Then, we transform the original data given an specific delay and radius as show in Fig 1.

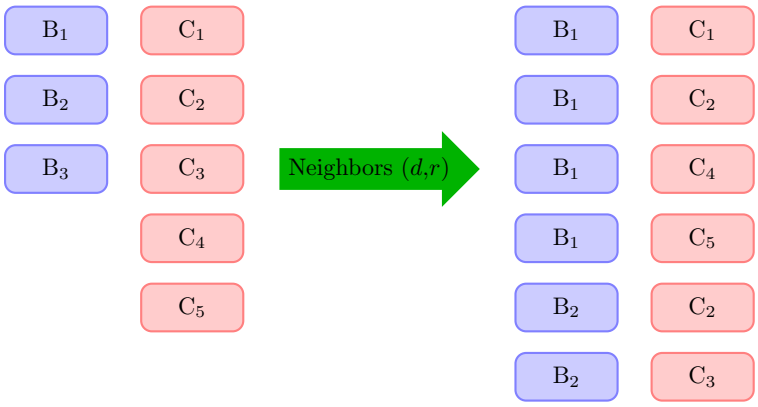

Figure 1: Data transformation using Neighbors algorithm

The results of our data transformation by year changing radius and delay are show in next figures.

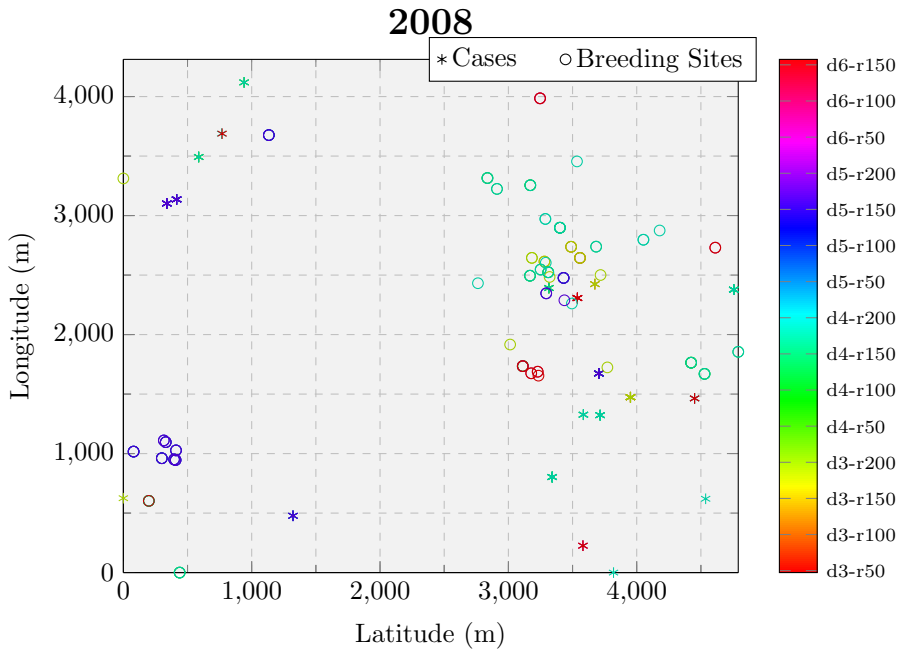

Figure 2: Neighbors 2008

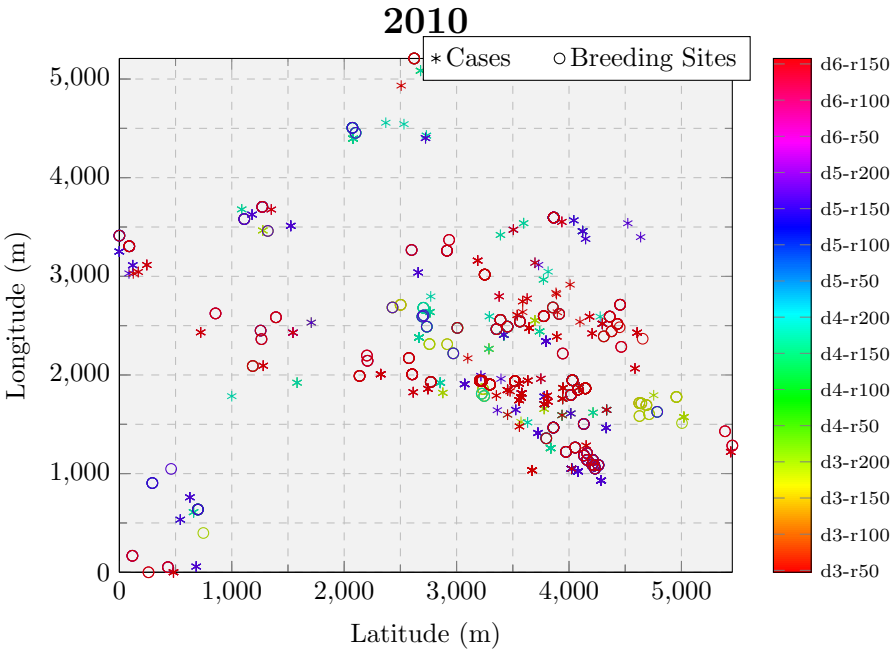

Figure 3: Neighbors 2010

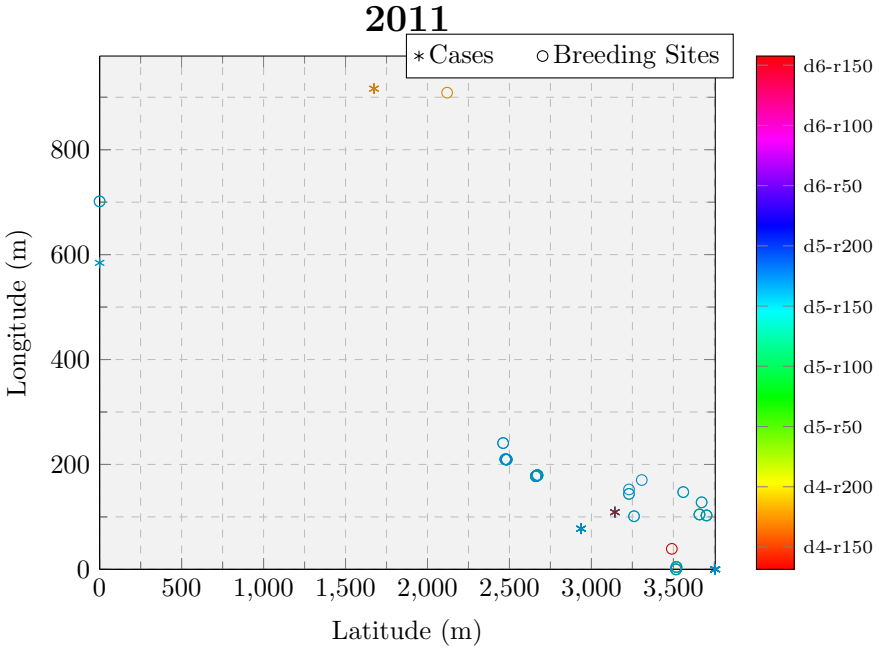

Figure 4: Neighbors 2011

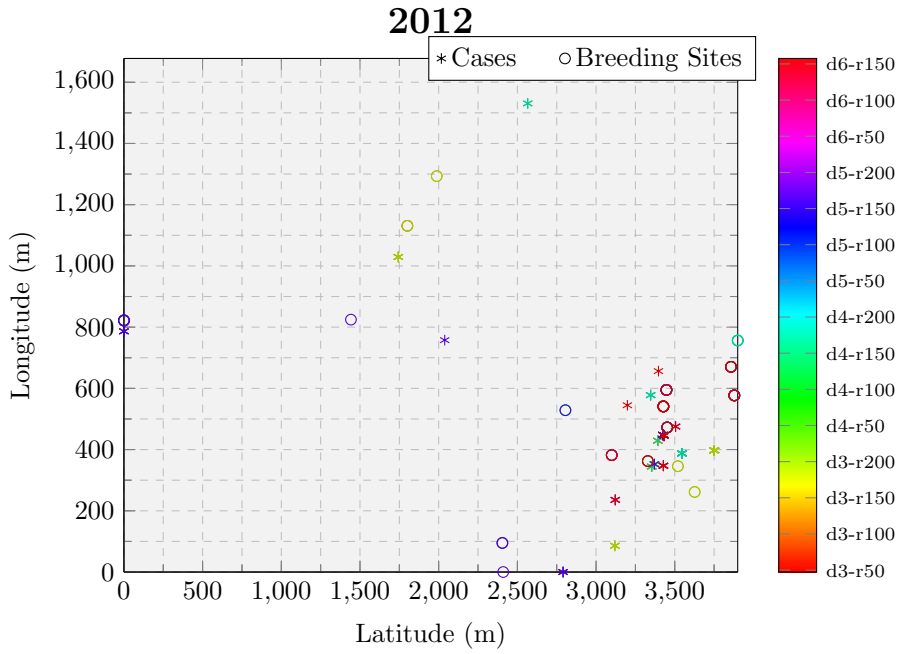

Figure 5: Neighbors 2012

Clustering

The previous simulation generates 16 subgroups of data by year, all of them are grouped by a combination of clustering techniques (subclust and fuzzy c-means). To find the optimum number of centers we use the *subclust* MatLab function, subsequently we take that number as a parameter for the *fcm* function to find the optimum location of the centers. Moreover we improve the *fcm* function given it the initialization of the centers (the result of *subclust*), that reduce the number of iterations and generally minimize the objective function.

The modified function (see Algorithm 1) takes a data set, the initial location of the centers and a vector of options (exponent for the partition matrix U, maximum number of iteration, minimum amount of improvement and a binary variable to display info during iteration) and returns the membership function matrix U (which contains the grade of membership of each data point in each cluster), the optimum location of the centers and a vector that contains the value of the objective function at each iteration.

Algorithm 1 follows the same procedure of *fcm* MatLab function, the principal change is the "Initial fuzzy partition" section where, with the initial centers we determined the Euclidean distance (*distfcm* function) of each data to each center, subsequently we initialized the U matrix as follows (note that the *fcm* function of MatLab initialized U randomly):

$$U_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ij}}{d_{kj}}\right)^{(2/(m-1))}} \tag{1}$$

Where d_{ij} is the Euclidean distance between i^{th} cluster center and j^{th} data point, c is the number of clusters and $m \in [1, \infty)$ is the weighting exponent.

Having an initial U matrix our modified function follows the general steps (*stepfcm* function):

1. Calculates c fuzzy clusters
2. Computes the cost (objective) function
3. Computes a new U by Equation 1
4. Goes to step 1

It stops if either it exceeds a maximum of iterations or if the improvement of the actual cost function over previous iteration is below a certain threshold.

```
input  : data, centers 0, options
output: U, centers, objFnc

n ← length (centers 0);
objFnc ← zeros (maxIter,1);
defaultOpt ← [2, 100, 1E − 5, 1];
if options is null then
    options ← defaultOpt;
end

expo ← options (1);
maxIter ← options (2);
minImpro ← options (3);
display ← options (4);

Initial fuzzy partition;
dist ← distfcm (centers 0, data);
tmp ← dist^(−2/(expo−1)));
U ← tmp/(ones(n, 1) ∗ ∑ tmp);
U (isnan (U)) ← 1;

Main loop;
for i ← 1 to maxIter do
    [U, centers, objFnc(i)] ← stepfcm(data, U, n, expo);
    if display then
        print "Iteration count = i, obj.fcn = objFnc (i)";
    end
    Check termination condition;
    if i > 1 then
        if |objFnc(i) − objFnc(i − 1)| < minImpro then
            break;
        end
    end
end
```

Algorithm 1: Main function of modified *fcm*

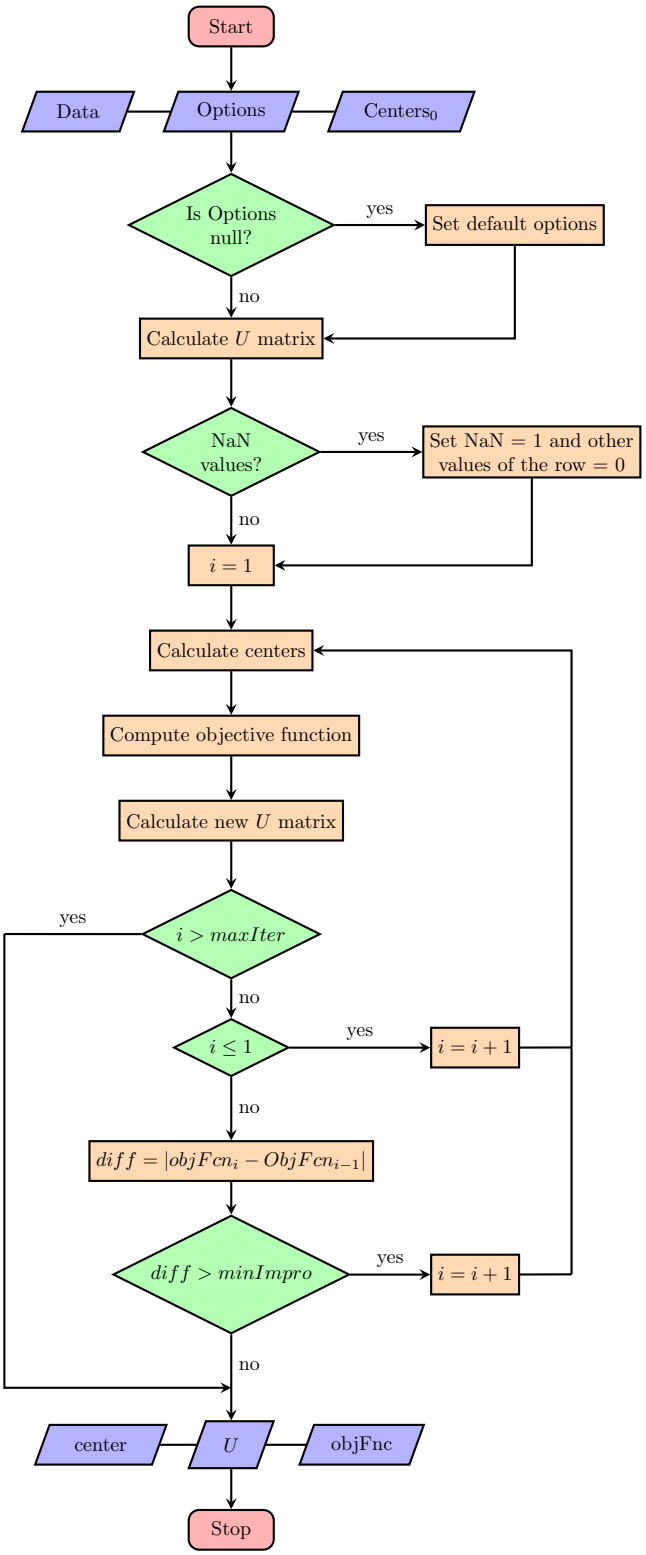

Figure 6: FCM Flowchart

The results of the clustering section are reported in the next tables, each column corresponds to:

- Index: number of the subgroup
- Delay: delay week
- Radius: radius of action (in meters)
- Size: data size, number of observations
- Clusters: optimum number of centers, this is given by the *subclust* function
- FCM: Objective function value of the *fcm* method.
- FCM2: Objective function value of the improved *fcm* algorithm.

Index	Delay	Radius	Size	Clusters	FCM	FCM2
1	3	50	1	0	0	0
2	3	100	3	3	2.12E-007	NaN
3	3	150	4	4	5.23E-007	NaN
4	3	200	11	6	1.67E-005	1.62E-005
5	4	50	3	3	1.21E-012	NaN
6	4	100	10	7	2.98E-005	3.07E-005
7	4	150	15	8	3.17E-005	5.12E-005
8	4	200	26	9	0.0001623551	0.0001305956
9	5	50	4	3	6.52E-004	NaN
10	5	100	8	3	8.58E-006	0.0002654141
11	5	150	10	3	1.11E-005	1.11E-005
12	5	200	11	3	1.22E-005	1.22E-005
13	6	50	0	0	0	0
14	6	100	2	2	1.09E-026	NaN
15	6	150	5	3	7.13E-007	7.13E-007
16	6	200	7	4	1.06E-006	0.0006128854

Table 1: Clustering 2008

Index	Delay	Radius	Size	Clusters	FCM	FCM2
1	3	50	1	0	0	0
2	3	100	18	10	0.0001715234	0.0002592176
3	3	150	58	7	0.0006638709	0.0006943074
4	3	200	86	8	0.001170078	0.0010761235
5	4	50	6	5	5.88E-005	NaN
6	4	100	26	5	0.0006173628	0.000582129
7	4	150	51	3	0.0020991648	0.0020985399
8	4	200	89	6	0.001859613	0.0017673659
9	5	50	11	1	9.67E-007	NaN
10	5	100	45	6	0.0005095857	0.0009989919
11	5	150	73	5	0.0017237401	0.0028274043
12	5	200	108	6	0.0030788256	0.0035099061
13	6	50	8	6	1.03E-006	8.59E-005
14	6	100	46	4	0.0026861811	0.0026893429
15	6	150	87	4	0.0043727996	0.0043944189
16	6	200	146	6	0.0036385615	0.0039475001

Table 2: Clustering 2010

Index	Delay	Radius	Size	Clusters	FCM	FCM2
1	3	50	0	0	0	0
2	3	100	0	0	0	0
3	3	150	0	0	0	0
4	3	200	0	0	0	0
5	4	50	0	0	0	0
6	4	100	0	0	0	0
7	4	150	1	0	0	0
8	4	200	2	2	4.85E-012	NaN
9	5	50	2	2	3.95E-010	NaN
10	5	100	8	4	3.68E-006	8.56E-008
11	5	150	17	4	1.65E-005	0.0004055341
12	5	200	21	4	2.04E-005	0.0004067676
13	6	50	0	0	0	0
14	6	100	0	0	0	0
15	6	150	0	0	0	0
16	6	200	1	0	0	0

Table 3: Clustering 2011

Index	Delay	Radius	Size	Clusters	FCM	FCM2
1	3	50	4	3	3.14E-006	NaN
2	3	100	5	4	1.65E-007	NaN
3	3	150	15	8	5.78E-006	2.98E-006
4	3	200	16	8	2.04E-005	4.40E-006
5	4	50	1	0	0	0
6	4	100	4	4	3.29E-006	NaN
7	4	150	9	6	4.39E-006	2.49E-006
8	4	200	11	7	5.82E-006	1.90E-006
9	5	50	1	0	0	0
10	5	100	2	2	3.80E-008	NaN
11	5	150	7	5	1.54E-005	4.06E-005
12	5	200	11	7	1.84E-005	1.39E-006
13	6	50	3	3	3.01E-007	NaN
14	6	100	5	5	7.20E-006	NaN
15	6	150	8	8	5.62E-006	NaN
16	6	200	10	10	6.79E-006	NaN

Table 4: Clustering 2012

As stated above, in the Tables 1, 2, 3, 4 it can be seen that generality our fcm algorithm minimizes the objective function, unfortunately at the same time is remarkable that it has problems when the data size is small. This is due to the origin of the initial centers because the *subclust* function takes each data point as a possible center and groups them with a a measure of closeness, but always the center is a data point. This implies that the result is a subset of the data set. Now, in the fcm method, when U is initialized, we have some distances that are zero (because there are data points that are centers) and in the Equation 1 it will generate division by zero. This phenomenon is controlled replacing the NaN values by 1 and the others values of the row by zero, but when the amount of centers is approximately the number of observations we obtain a diagonal matrix of ones as U, and this is an useless matrix.

Neural Network

However, the previous clusters do not have a clear interpretation, for that reason in this section we consider only the data sets obtained by the Neighbors algorithm. For each year we selected the 4th largest data sets (highlighted in Table 1, 2, 3, 4) and with them an unique data set by year was created, every set contained the coordinates (Longitude, Latitude and Week) of the Breeding Sites and the possible Cases associated to them.

With the selected data we created a Neural Network (by year) that takes as inputs the coordinates of the Breeding Sites and generates as output (predicts) the coordinates of the Cases. The Networks were created using the MatLab command *nftool*.

The general architecture of our neural networks have three inputs, one hidden

layer with n neurons and three outputs (see Figure 7). In every year the variables of the neural network will be the number of neurons in the hidden layer and the size of data (number of observations).

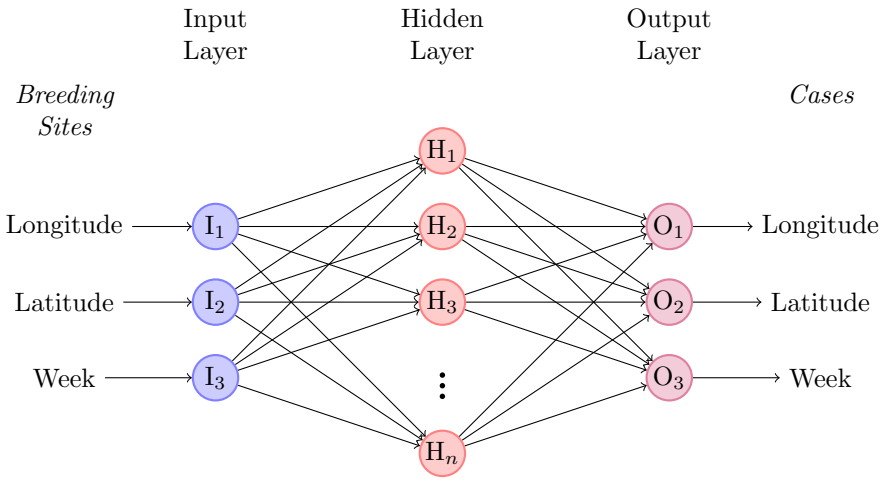

Figure 7: General structure of a Neural Network

All our neural network are a two-layer feed-forward network with sigmoid hidden neurons and linear output neurons. The networks are trained with Levenberg-Marquardt backpropagation algorithm, unless there is not enough memory, in which case scaled conjugate gradient backpropagation is used (non of our neural networks required it).

For 2008 we have 63 observations that are divided, 60% to training, 20% to testing and 20% to validation. The neural network has 15 neurons in the hidden layer.

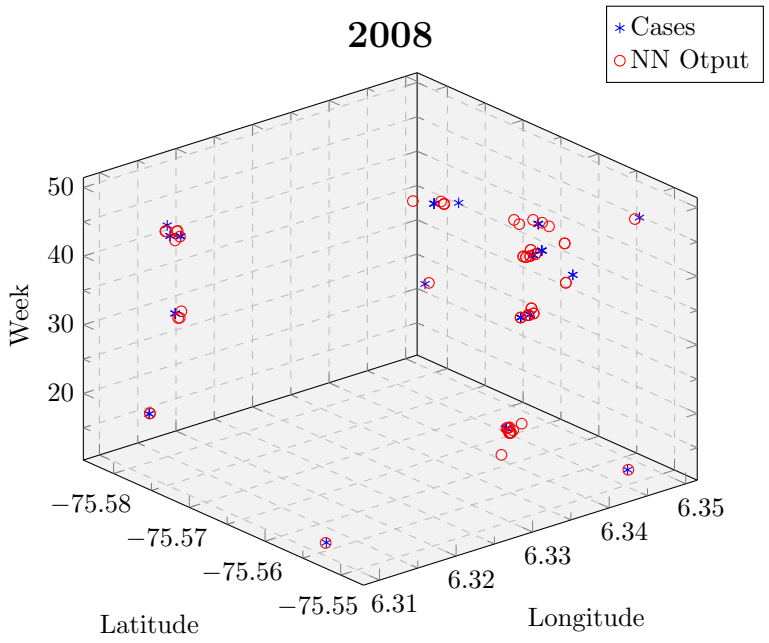

Figure 8: Location of the real and the predicted Cases in 2008

In Figure 8, the location of the original Cases and the predicted by the neural network are shown. Note that the blue asterisks are relatively close to the red circles (the main idea is to have the asterisks and the circles as close as possible). To quantify how close are they, for each variable (Longitude, Latitude and Week of the Cases) we subtract the predicted data to the original ones, this works as an error measure.

The results given in Figure 9 show that the neural network for 2008 has a good approximation predicting the Longitude and the Latitude. Although the Week variable has a bigger scale, it means that the neural network has a lag of an a half of a week predicting the Week variable which is an acceptable interval. Then, in general terms, and confirming the analysis of the Figure 8, the neural network for 2008 has a good behavior.

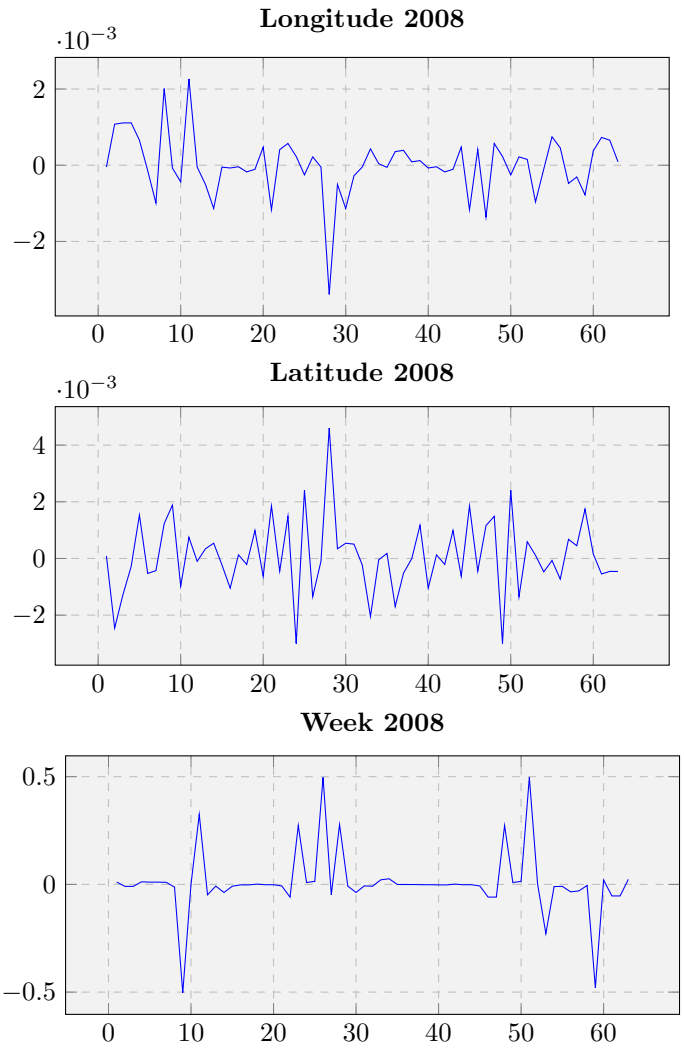

Figure 9: Difference between real and predicted data variables 2008

For 2010 we have 429 observations that are divided, 50% to training, 25% to testing and 25% to validation. The neural network has 50 neurons in the hidden layer.

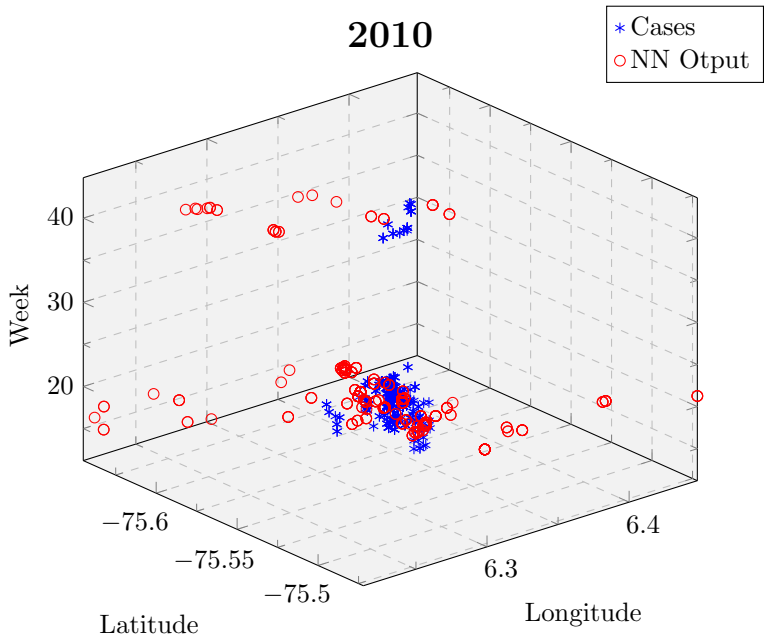

Figure 10: Location of the real and the predicted Cases in 2010

Figure 10 shows that the neural network for 2010 could not learn the behavior of the data set, the errors (see Figure 11) predicting the variables are not permissible, specially in the Week variable (it would have a lag from half to one month) and follow by the Latitude one, this, considering that a variation of a decimal in a Latitude is a significant change in the point location.

Despite the fact, the bad results were expected. In 2010 there was an DF epidemic in the municipality of Bello, as a consequence the data set of this year has an abnormally pattern, then the cases location becomes unpredictable.

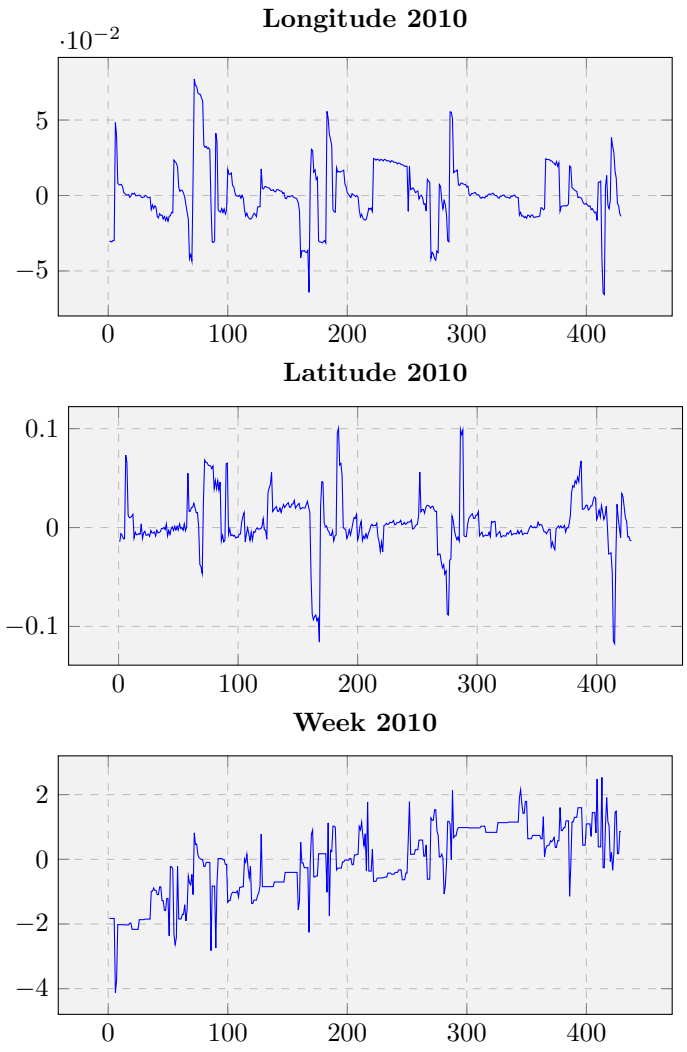

Figure 11: Difference between real and predicted data variables 2010

For 2011 we have 38 observations that are divided, 70% to training, 15% to testing and 15% to validation. The neural network has 5 neurons in the hidden layer.

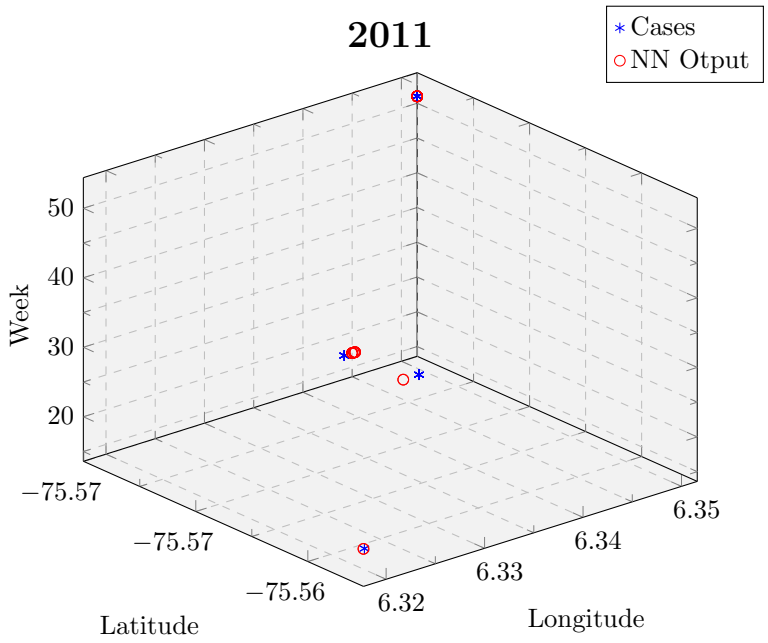

Figure 12: Location of the real and the predicted Cases in 2011

Figure 12 evidence that the neural network for 2011 predicts correctly (with a small error) the location of the cases, this is confirmed with the difference between the variables (see Figure 13) where, based on the scale of the graphics, we conclude that the neural network has an excellent approximation.

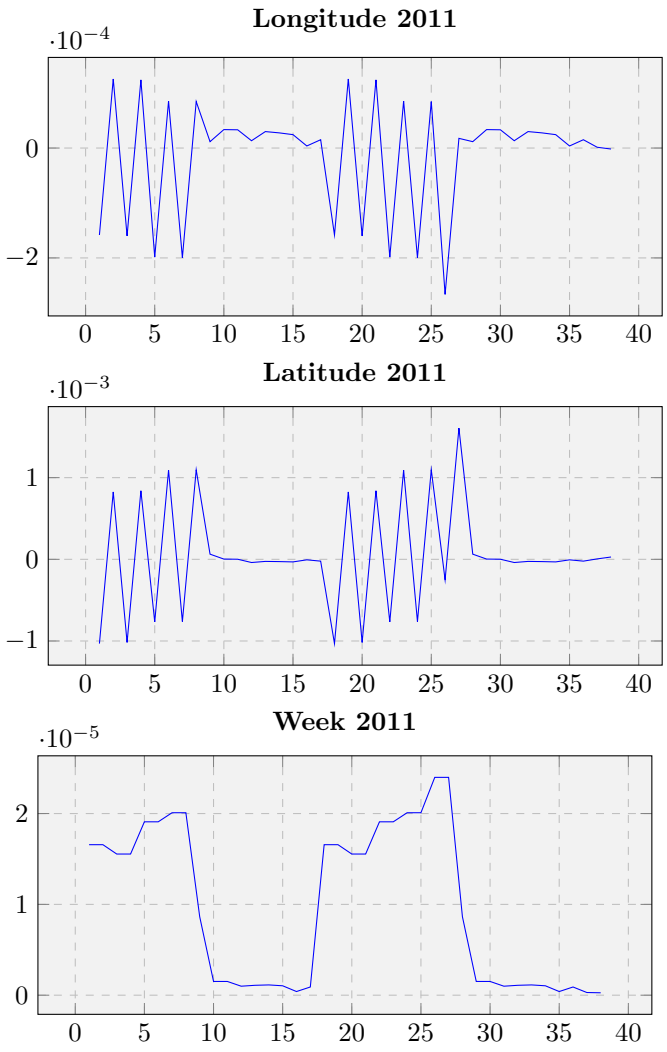

Figure 13: Difference between real and predicted data variables 2011

For 2012 we have 53 observations that are divided, 70% to training, 15% to testing and 15% to validation. The neural network has 10 neurons in the hidden layer.

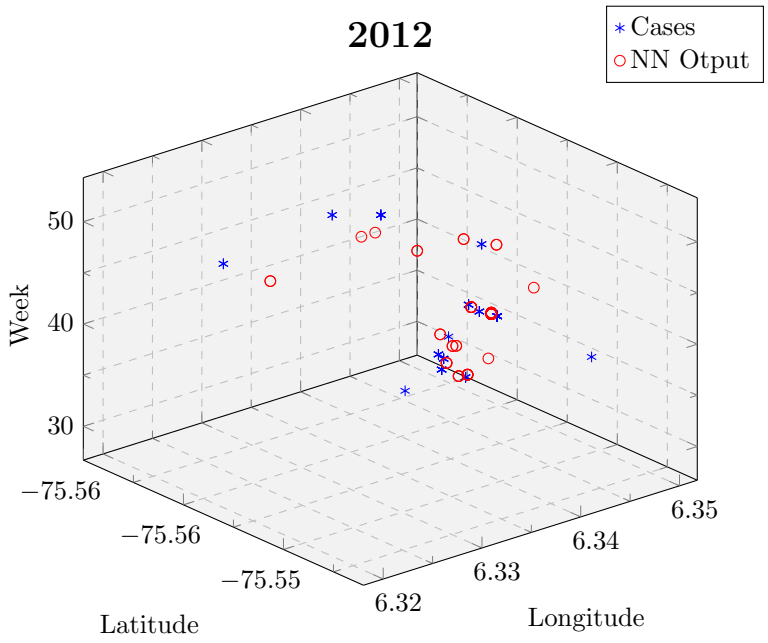

Figure 14: Location of the real and the predicted Cases in 2012

In Figure 14 we observed that the neural network for 2012 estimates correctly (with a small error) the Longitude and Latitude variables, however the estimation of the Week variable has some looseness, this is reflected in the Figure 15, where the third graph has a considerable error in the 45^{th} observation approximately.

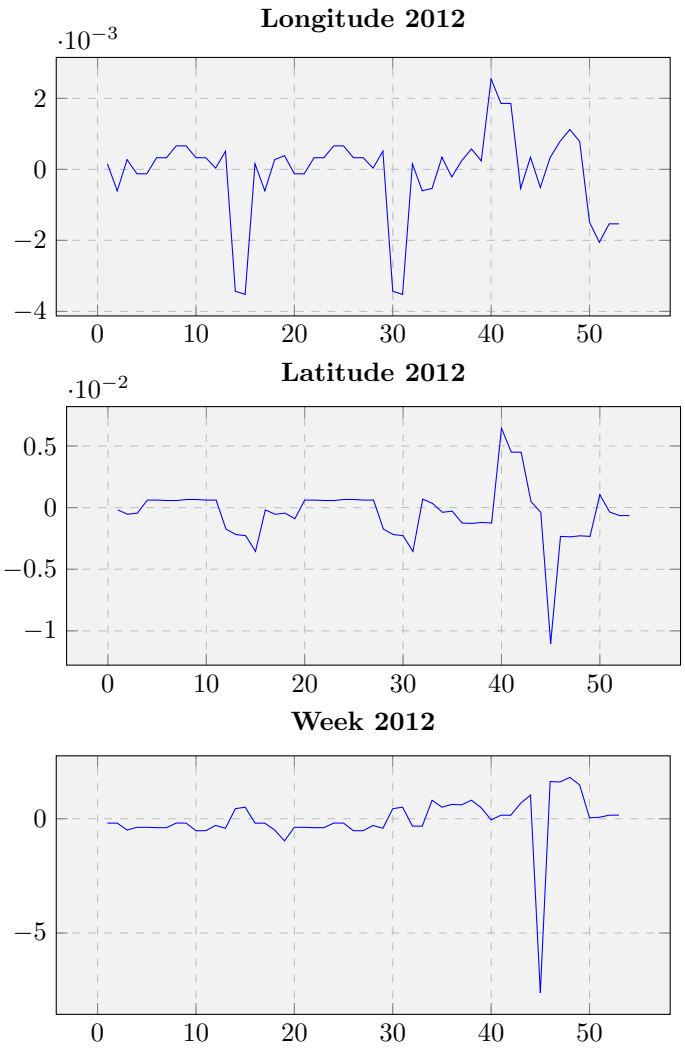

Figure 15: Difference between real and predicted data variables 2012

Finally, we created a general neural network that uses the data of the 2008, 2011 and 2012, for this new set of data we have 197 observations that are divided, 60% to training, 20% to testing and 20% to validation. The neural network has 10 neurons in the hidden layer.

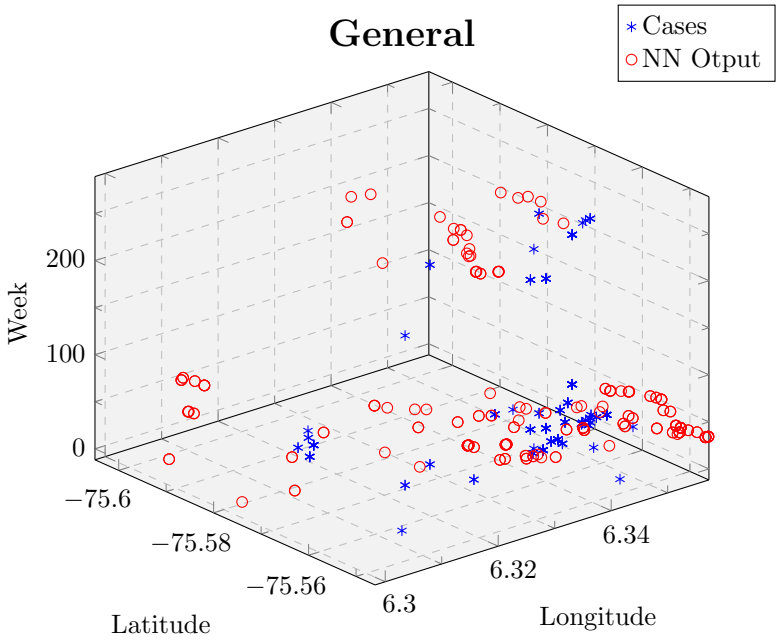

Figure 16: Location of the real and the predicted Cases in NN without 2010 data

In Figure 16 we can not conclude rightly if the neural network has good or a bad behavior, but with the help of Figure 17, it is clear that the predicted variables are approximate to the real ones. Note that the scale of the difference between the variables is small except by some outliers in the Week variable.

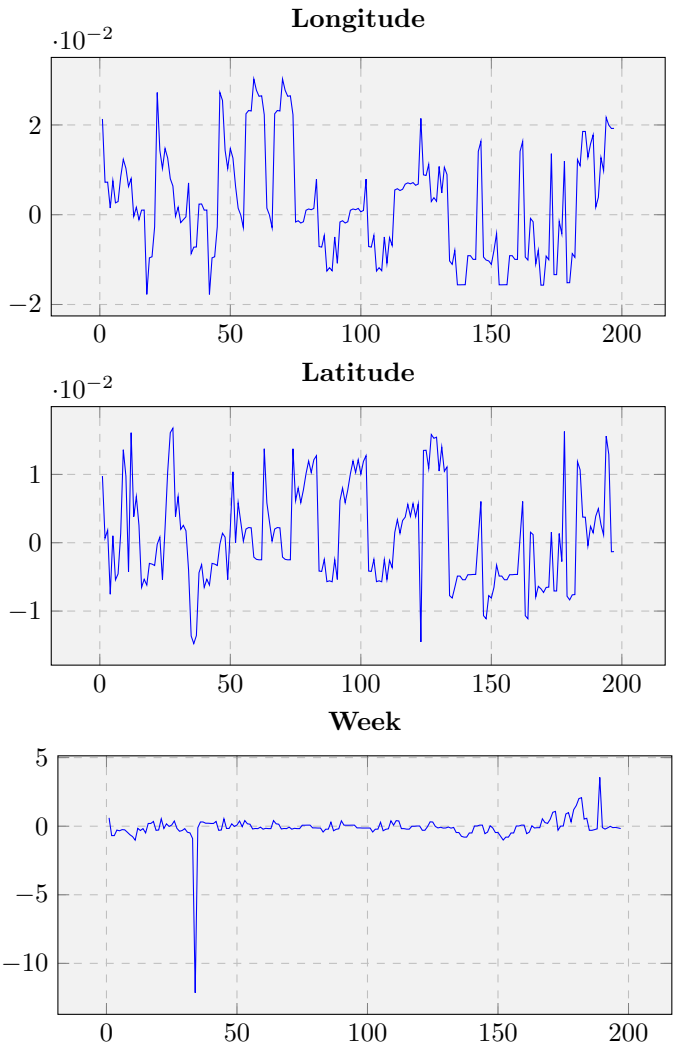

Figure 17: Difference between real and predicted data variables