# Milestone 2 Report - Supervised Learning

Machine Learning

Agustín Mora Acosta
Agustin.Mora1@alu.uclm.es
UCLM
Ciudad Real, Spain

Andrés González Díaz
andres.gonzalez8@alu.uclm.es
UCLM
Ciudad Real, Spain

## KEYWORDS

supervised, machine learning, kneighbors, gridsearch, dengue

**ACM Reference Format:**
Agustín Mora Acosta and Andrés González Díaz. 2020. Milestone 2 Report - Supervised Learning: Machine Learning. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

The purpose of this work is to use machine learning techniques to make prediction of dengue cases over the environmental data collected by various U.S. Federal Government Agencies from two different cities (San Juan in Puerto Rico and Iquitos in Peru).

This data comes from a competition of the site DrivenData. The information about this competition can be found in the following link: https://www.drivendata.org/competitions/44/dengai-predicting-disease-spread/

Our main objective, as we said previously, is to apply supervised learning techniques in order to predict the total cases of Dengue for each city, year and week of the test data. In the notebooks, a submission has been built containing the predictions for both cities, which can be uploaded to DrivenData to evaluate the error made on the test set when predicting.

You can find the repository that contains the notebooks used to apply these techniques in the following link: https://github.com/Agustin6199/Milestone-2-Supervised-Learning

## 2 BASE LINE

First of all, we established a base line where we use a basic regression technique: kNN. K-Nearest Neighbors (kNN) is an algorithm based on learning by analogy, commonly known as instance-based algorithm, which does not attempt to construct a general internal model, but simply stores instances of training data. When a new case arrives to be classified, the model look for the most similar cases and then, classification is built according to the class which those cases belong.

### 2.1 Preprocessing

In this base line, we defined some preprocessing steps we'll use in further lines. This steps are the followings:

- **Handling empty values**: We can find some missing values filled as Nan's on our data. To fill those values, we used the K-Nearest Neighbors imputer, which utilizes the kNN method to replace the missing values in data. We tried out different values for the number of neighbors, but the value that gave the best results was 4.
- **Normalization**: For normalize data, we tested several scalers, concluding that the MinMaxScaler was the one that gave us better results. This are the scalers we tested: StandardScaler, RobustScaler, MaxAbsScaler, MinMaxScaler.
- **Other preprocessing steps**: In addition to the above, we also removed indexes and columns that were not useful for the algorithm, like the week_start_date.

### 2.2 Model

After this preprocessing steps, we used 2 nested loops to iterate over different combinations of parameters, fitting a kNN model with each combination, and validating the model with cross-validation. We only tested combinations of 2 parameters, number of neighbors and weights. For the weights we tested 'distance' and 'uniform' values, giving both similar results. For the number of neighbors we tested integer numbers from 1 to 50, concluding that the best results on validation were given from 30 neighbors onwards.

When we tested every combination of parameters, we built a model with the parameters of the model with best results on validation:

- Number of neighbors(n_neighbors): 45
- weights: 'distance'

Test data has empty values as well, so we had to fill such values like we did previously with the training data. For this, we used the imputer fitted previosly over training data. Then, we also normalize data using the same scaler we previously fitted on training data.

Lastly, in this baseline we have defined how to predict over test data, obtaining the total cases of dengue for this data. After predicting, we round the result, and we cast it to integer. Finally, we


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

generate an output file, which contains such total cases in the appropriate format to make a submission to DrivenData.

The best result achieved from this baseline was a MAE of 26.6298. Although perhaps we could have done more testing on this baseline to improve this result, we decided to move to a middle line where we used a slightly more complex model.

## 3 MIDDLE LINE

After the base line commented, we established a middle line. In this middle line we added some new features and a feature selection. We also changed the model to a Random Forest Regressor, which parameters were optimized using a Randomized Search with cross validation.

Random Forest is an ensamble of parallel randomized decision trees, each of which over-fits the data, and averages the results to find the final result.

### 3.1 Feature addition

To improve the results, we analyzed the data in order to find new features to add that could provide information. In this analysis, we took out the charts that we can see in Figure 1 and Figure 2. In the charts, we can see that the temporary factor affects the number of total cases of dengue that there are in a certain week, since there are stages in which there are a greater number of cases, and there are other stages in which the number of cases is smaller. Even, if we look more in detail (especially in the Iquitos graph), we can see that a certain seasonal factor exists, since those times with greater number of cases matches with the end/beginning of the year.

**Figure 1: Time representation of data (San Juan)**

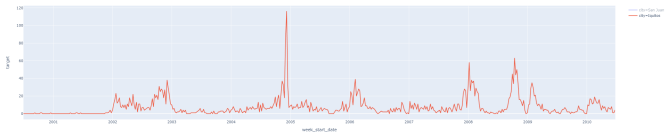

**Figure 2: Time representation of data (Iquitos)**

In order for the model to take this temporal factor into consideration, we extracted various temporal component like weekday, month or season. Then we applied some transformations to add numeric periodic features using this temporal components. This way, we got some new features that represent the temporal component between 0 and 1, where December value is closer from January than from October.

### 3.2 Feature selection

We also add a feature selection step before building our model, in order to keep those features which provides more information to our objective. For this feature selection, first we computed the correlation matrix that between the different features we can see in Figure 3, in order to identify those features that are redundant and therefore do not contribute to the model.

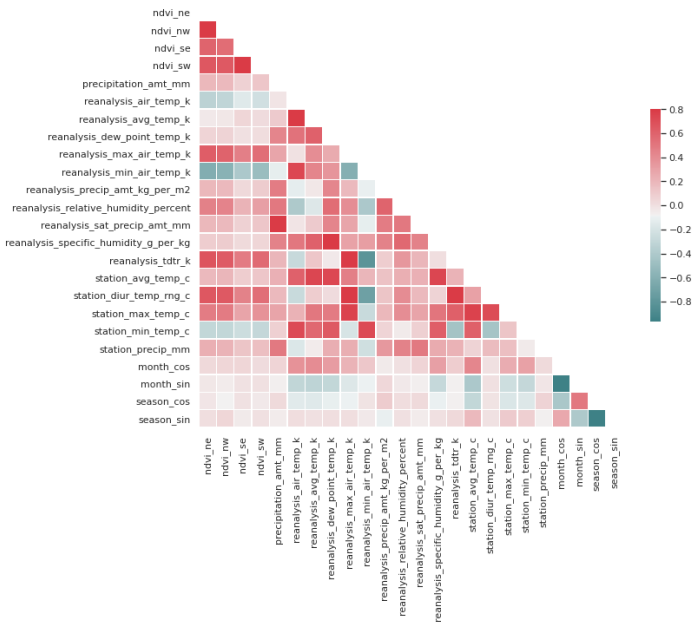


**Figure 3: Correlation matrix between features**

As we can see in the correlation matrix, some features are strongly correlated between them (nvdi_xx features for example). We carried on with this feature selection, keeping in mind this correlations. After this, we used a model called 'SelectPercentile' from feature_selection package of Scikit-Learn to select those definitive features. This model, select features according to a percentile of the highest score, using a certain score function. For this purpose, we established 'f_regression' as our score function, since this function provides us the F-value between label/feature for regression tasks. As a percentile we used the value 60, with which this model selected 14 characteristics from the 24 original ones. These features were the following:

'ndvi_ne', 'ndvi_nw', 'ndvi_se', 'ndvi_sw', 'reanalysis_air_temp_k', 'reanalysis_avg_temp_k', 'reanalysis_dew_point_temp_k', 'reanalysis_max_air_temp_k', 'reanalysis_min_air_temp_k', 'reanalysis_relative_humidity', 'reanalysis_tdtr_k', 'station_diur_temp_rng_c', 'station_min_temp_c', 'season_sin'

As we can see, in this new features we have every ndvi_xx features, which had a strong correlation between them and therefore are redundant, so we dropped 3 of this features in order to reduce even more the final features set. Finally, we reduced our features to 11, we used this definitive features for the following steps.

### 3.3 Model

As we said before, we have used Random Forest Regressor as estimator. For the parameter optimization of this estimator, we first used a Grid Search approach which allowed us to methodically construct and evaluate a model for each combination of specified algorithm parameters in a grid. This approach was discarded due the execution time was so high, preventing us from executing it completely. The solution to this problem was to use a Randomized Search instead of this Grid Search, which only evaluates a few combinations of parameters, but was much better in terms of execution time. In this Randomized Search we built and evaluated models with 250 random combinations of parameters.

For validation, we defined a cross validator using TimeSeriesSplit from Scikit Learn, which is a specifically time series validator that provides train/test indices to split time series data samples that are observed at fixed time intervals, in train/test sets.

This are the results of the best models tested: Table 1

As we can see the top 2 models share features, that's why teir results are close. The different between them is that in the first one we used MinMaxScaler and in the second we used RobustScaler.

## 4 HIGH LINE

Finally we will comment on our last line of improvement, the high line. In this high line, we have used the same tools that we use in the middle line, but with another approach, creating a different model for each city. In addition to the model, we have also declared an imputer and a scaler for each city, as well as having to add some extra code cells to manipulate the test data.

This way we made some improvements in the results, but in order to achieve better results we tried to change the model to a more complex one. For this we decided to use other ensamble models, the boosting models. This algorithms sequentially builds a set of decision trees that involve weak learners. In each iteration of the algorithm, each individual model tries to correct the errors made in the previous iteration by optimizing a loss function.

For this, we tried two libraries that contain boosting algorithms based on gradient descent, XGBoost and LightGBM. Unfortunately, none of the tests we did using these libraries gave us better results than those provided by Random Forests. Below we can see the set of parameters we used in the parameter search of one of these two models:

*"boosting_type":['gbdt', 'dart', 'goss', 'rf'],*
*"learning_rate":[0.5, 0.1, 0.01, 0.001],*
*"n_estimators":[16, 64, 256, 512, 1024, 2048],*
*"num_leaves": [16, 32, 64, 128],*
*"max_depth": [2, 4, 8, 16, -1],*
*"reg_alpha":[0., 0.1, 0.25, 0.5],*
*"reg_lambda":[0., 0.1, 0.25, 0.5]*

Finally, this are the results of the best models tested: Table 2

After choosing the best model, we got a score of 25.0216, achieve by applying Random Forest Regresor that is the same model that in middle line. The reason as we said before is that others model didnt give us better results. Some features of the model is that we used MinMaxScaler for both, if we look at it we see that Iquitos takes higher values in some parameters like 'n_estimators' or that 'max_depth' is not limited, this may be because there are fewer Iquito's records than San Juan.

## 5 APPENDIX

The DrivenData users used to make the submissions are the following:

- **User1**: Ofeucor
- **User1**: AgustinMora6199
- **User3**: Ofeucor2

In addition to our GitHub repository mentioned in the introduction, you can view the notebooks we have used for this work online at the following links:

- Base Line
- Medium Line
- High Line

| Score | criterio | max_features | n_estimators | max_depth | min_samples_split | min_samples_leaf |
|---|---|---|---|---|---|---|
| 25.8317 | 'mse' | sqrt | 16 | 12 | 28 | 8 |
| 25.9663 | 'mse' | sqrt | 16 | 12 | 28 | 28 |
| 26.2837 | 'mae' | log | 32 | 8 | 12 | 8 |

**Table 1: Medium Line**

| | Score | criterio | max_features | n_estimators | max_depth | min_samples_split | min_samples_leaf | Scaler |
|---|---|---|---|---|---|---|---|---|
| sj | 25.0216 | 'mse' | log2 | 32 | 12 | 31 | 48 | MinMaxScaler |
| iq | 25.0216 | 'mae' | sqrt | 128 | None | 44 | 48 | MinMaxScaler |
| sj | 25.0865 | 'mse' | auto | 16 | 4 | 14 | 28 | RobustScaler |
| iq | 25.0865 | 'mse' | sqrt | 128 | 2 | 11 | 19 | RobustScaler |
| sj | 25.9663 | 'mse' | sqrt | 16 | 2 | 32 | 29 | MinMaxScaler |
| iq | 25.9663 | 'mse' | sqrt | 16 | 16 | 32 | 32 | StandardScaler |

**Table 2: High Line**