UNIVERSITÀ DI PISA

Bike Rental Project

Course of Intelligent Systems

2014/15

Authors

Sara Egidi

Alessio Villardita

Bike Rental Project

Course of Intelligent Systems

Abstract

This report documents the in-depth study of a bike rental system located in Washington DC. The analysis is carried out with the use of neural and fuzzy systems and it aims at understanding the variables that affect the final number of bike rentals.

The first part is focussed on the use of neural networks to correctly fit the output of the systems. Two different approaches have been used: MLP and RBF.

After this first analysis the system could be modelled by means of a fuzzy system. Also for this case, two different approaches have been used and compared: Mamdani and ANFIS.

The second part of the project covers the realization of a system able to forecast the bike rentals, with both an open and a closed loop NARX system.

Finally, the list of the needed scripts is reported in the appendix, with a list of all the complementary files needed for the project.

About this document

.

The purpose of this document is to report all the design choices made while creating the different approximation systems.

A list of all the needed scripts is to be found in the appendix at the end of the document, along with some command line examples.

Some analysis were carried out separately, and reported in external pdf files. A list of those additional files is reported at the end of this document.

Contents

Introduction

This document reports the analysis and modelling of different intelligent systems to describe and predict the flux of bike rentals of an organization located in Washington D.C. The core data set is related to the two-year historical log corresponding to years 2011 and 2012 from Capital Bikeshare system, Washington D.C., USA which is publicly available at *http://capitalbikeshare.com/system-data.*

Before even starting, some data adjustments should be considered prior to the efficient use of the raw data from the given csv files (e.g. some columns may not be useful for the analysis and some data could be missing).

The dataset

Before starting the analysis, it is important to deeply inspect the dataset and determine how the available data should be interpreted.

At first, we started reasoning on the usable columns in the csv file, since some of them may be of no use for the purpose of the analysis. We decided to omit the following columns: instant, casual and registered. The first one is just an identifier, while casual and registered are strictly related to the overall count of the rented bikes, since the latter is the sum of those two terms.

Another consideration was done on the day column, since it was expressed in a date format. For this reason, it has been decided to split this column in order to obtain a "day" value with an integer format instead of a date one.

Since the data will be used to build Neural Networks and other systems, the data contained in the file should be normalized to properly infer on the data. As reported in the notes released within the dataset, some fields have already been normalized (temp, atemp, hum and windspeed), so there is no need to process their values. Other data normalization will be considered only whether the need to perform such computation will arise, i.e. the neural network performance will look biased or with high variance.

Part I

In the first part of the project, in order to estimate the number of rented bikes, different fittings have been carried out: using at first a multi-layer perceptron (MLP), then moving to the use of radial basis functions (RBFs) and concluding with a fuzzy model. In the following, all the steps, algorithms, considerations and results are reported and explained.

1.1 Neural fitting model

In this first part we were asked to develop a MLP and then a RBF network in order to select the best subset of features and then, based on this selection, to estimate the number of bike rentals.

1.1.1 MLP

Using the Matlab NN Fitting tool for each feature we are able to determine the optimal set of features by evaluating the MSE. To accomplish this task, an incremental analysis has been performed. First, we started evaluating the MSE for each single feature and then we moved up by adding to the working subset of features all the most important ones (one by one and according to a MSE based ranking).

1.1.1.1 Single feature selection

Here is reported the pseudo-code of the algorithm that has been used during the initial phases:

```
1. Load the datasets into the workspace
2. For each interesting single feature, do the following:
     i. Generate 5 samples of the MSE (with the selected feature) by
        calling the fitFeatureSize function
3. For each feature, compute the mean value of its array of MSE values
   and store it in the output array
4. Select the best performing feature
```

At first, all the data has to be loaded into the Matlab environment; to this aim the csv was saved as an *xlsx*, since the matlab function *csvread* considers only integers.

We decided to make a script just to load all the required data. The script's name is *init.m* (see reference "*Source Code: [1] - Init*" for the implementation).

Now that the workspace is ready, we can start using the NN fitting tool for each feature. The script file *fitFeatureSize.m ("Source Code"[2])* has been built, and it also has an input in order to have an adjustable number of the hidden layer size, so that later the analysis on the number of neurons can be conducted. The training ratios have been set to the default values (70% training, 15% test and 15% validation).

A second script has been developed to run these repeating actions several times to have as unbiased results as possible and assess the best ones according to their performances based on the MSE metric. The number of samples (i.e. iterations) has been set to 5 because of the complexity of every training execution, in other words to maintain the time required for each execution of the script low.

The final fixed layer size for the feature selection is 10, and the ranking of the most meaningful features is reported respectively for day and hour data in the tables below:

Hour data performances (values magnitude: 1.0e+03)

Feature	1st run	2nd run	3rd run	4th run	5th run	Average	Ranking
Season	3.073020	3.072964	3.073570	3.073004	3.073039	3.073144	1.Hour
Year	3.087187	3.087924	3.083720	3.083536	3.083549	3.085183	2.Atemp
Month	3.043909	3.043487	3.044393	3.043958	3.043507	3.043851	3.Temp
Hour	1.641693	1.641929	1.641349	1.652112	1.640787	1.643574	4.Hum
Holiday	3.286854	3.286970	3.286902	3.287028	3.286840	3.286919	5.Month
Weekday	3.286740	3.286415	3.286739	3.286994	3.286339	3.286645	6.Season
Workday	3.286961	3.286957	3.287075	3.287125	3.287012	3.287026	7.Year
Weather	3.219330	3.219350	3.219670	3.219700	3.220237	3.219657	8.Weather
Temp	2.725766	2.722592	2.719430	2.720154	2.724062	2.722401	9.Windspeed
Atemp	2.692575	2.696035	2.697648	2.684486	2.673570	2.688863	10.Weekday
Hum	2.919691	2.920702	2.923332	2.922717	2.925797	2.922448	11.Holiday
Windspeed	3.239237	3.239612	3.242666	3.238756	3.238749	3.239804	12.Workday

Day data performances (values magnitude: 1.0e+06)

Feature	1st run	2nd run	3rd run	4th run	5th run	Average	Ranking
Day	3.742980	3.786029	3.744801	3.755437	3.787748	3.763399	1.Atemp
Season	2.449701	2.564856	2.455175	2.450256	2.453277	2.474653	2.Temp
Year	2.544059	2.557654	2.548087	2.546009	2.545128	2.548187	3.Month
Month	2.290785	2.296869	2.304537	2.294913	2.311494	2.29972	4.Season
Holiday	3.732788	3.731317	3.735368	3.739153	3.732294	3.734184	5.Year
Weekday	3.738950	3.729965	3.757252	3.759154	3.733995	3.743863	6.Hum
Workday	3.742063	3.741446	3.735841	3.734157	3.736069	3.737915	7.Weather
Weather	3.379015	3.381255	3.376837	3.379157	3.376641	3.378581	8.Windspeed
Temp	2.137431	1.980439	2.023717	1.987187	1.996948	2.025144	9.Holiday
Atemp	1.987144	2.025613	1.972892	1.976033	2.031138	1.998564	10.Workday
Hum	1.064672	3.475557	3.431469	3.473722	3.944612	3.078006	11.Weekday
Windspeed	3.476741	3.455830	3.416739	3.510123	4.289328	3.629752	12.Day

As we can already see, Temp and Atemp seem to be the best performing ones in both of the analysis. Also hour performs well on the hour data, but that's justifiable by the fact that the count of the rented bikes is strictly related to the hours, as people rent more bikes during the day time.

An interesting result can also be seen in the humidity feature that in the hourly data set seems to be fairly more important than in the daily one. After some research, it emerged that the humidity raises a lot during the night, regardless of the weather, as it is strongly linked with the hours.

1.1.1.2 Features subset selection

By looking at the results, we could already make an assumption on the fact that we expect the first features of the ranking to perform good together. To test our assumptions, we started to see how the network performs for every combination of features, since it could be possible that even if two features are low in the ranking they still perform well while combined together. The script to test all the combinations of the day data is reported below:

```
for i = 2:12
    for j = i+1:13
        sum = zeros(1,1);
        for it_num = 1:5
            day_performance = fitfeatureSize(data_day(:,[i j]), cnt_day(), 10);
            sum = sum + day_performance;
        end;
        day_results(i-1, j-2) = sum/5;
    end;
end;
```

See "Source code" - [4] mlp2Features.m

In this case all the possible couples have been used in the training and their performance have been recorded over 5, and thus their average has been computed. The results are reported in the following table:

	season	year	month	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed
day	2.47E+006	2.56E+006	2.18E+006	3.72E+006	3.76E+006	3.72E+006	3.40E+006	2.01E+006	2.04E+006	3.40E+006	3.51E+006
season		1.23E+006	2.21E+006	2.45E+006	2.49E+006	2.44E+006	2.09E+006	1.89E+006	1.84E+006	2.15E+006	2.37E+006
year			1.02E+006	2.55E+006	2.51E+006	2.53E+006	2.23E+006	9.20E+005	9.25E+005	2.31E+006	2.34E+006
month				2.34E+006	2.34E+006	2.30E+006	1.93E+006	1.89E+006	1.91E+006	1.90E+006	2.14E+006
holiday					3.78E+006	3.74E+006	3.37E+006	1.99E+006	1.99E+006	3.40E+006	3.55E+006
weekday						3.77E+006	3.37E+006	1.97E+006	1.96E+006	3.45E+006	3.49E+006
workingday							3.37E+006	1.96E+006	1.97E+006	3.45E+006	3.46E+006
weathersit								1.69E+006	1.69E+006	3.36E+006	3.16E+006
temp									1.85E+006	1.73E+006	1.92E+006
atemp										1.61E+006	1.97E+006
hum											3.25E+006
Minimum	**2.47E+006**	**1.23E+006**	**1.02E+006**	**2.34E+006**	**2.34E+006**	**2.30E+006**	**1.93E+006**	**9.20E+005**	**9.25E+005**	**1.61E+006**	**1.92E+006**

By looking at the result table we start seeing some correlations, and after running the script several times we counted the total occurrences among the best performing couples.

	Feature 1	Feature 2
9.20E+005	temp	year
9.25E+005	atemp	year
1.02E+006	month	year
1.23E+006	season	year
1.61E+006	atemp	hum
1.92E+006	temp	windspeed
1.93E+006	month	weathersit
2.30E+006	month	workingday
2.34E+006	month	holiday
2.34E+006	month	weekday
2.47E+006	season	day

OCCURENCES	
month	5
year	4
season	2
temp	2
atemp	2
day	1
holiday	1
weekday	1
workingday	1
weathersit	1
hum	1
windspeed	1

We can see that year is performing well with most of the features, but we shouldn't consider it as a good feature for the fitting, since it's like considering a "transitory" value. As we see plotting the count or the temperature they both change in relation to year.

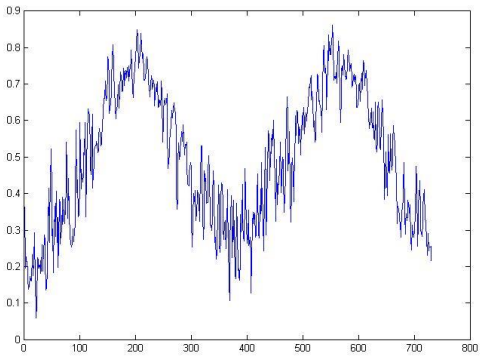

Figure 1- Temp

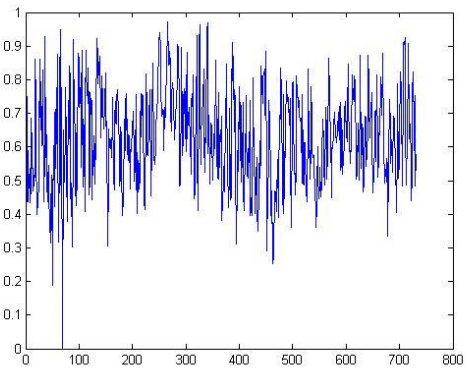

Figure 2 - Hum

By looking at the figures above we can see that there is always some kind of periodic trend for the best performing features, while the worst ones don't show this kind of behaviour (Figure 2).

To further improve the performances we decided to try out different fitting functions. The default one used by the tool is the `trainlm`, which is the `Levenberg-Marquardt`.

After some tries we discovered that the best one is `trainbr` that uses the `Bayesian regularization backpropagation` which, by using the weights and bias values obtained by Levenberg-Marquardt optimization, can further optimize the MSE.

So the fitting function was modified adding the following lines of code:

```
net = init(net);
net.trainFcn = 'trainbr';
```

With this method we observed an error with just a pair of features of the order of 1262 out of a range that spaces from 22 to 8714. This error is still too high, that's why we started adding more features to the network.

However, by doing this procedure we see that we shouldn't follow the first ranking as a guideline for the feature selection.

To obtain a final NN with the best features we tried to train the net with more and more features until we get no more tangible improvement, each time looking at all the possible combinations.

For the feature selection the approach was to first check which of the pairs performed better, and then to try that pair with all the remaining features. The best performing combination were then tested again and the process iterated until we saw no further improvement in the net.

Starting from the day, the feature selection process is summarized below:

3 features - 2nd run	
day	1.598E+006
season	1.388E+006
month	1.416E+006
holiday	1.597E+006
weekday	1.596E+006
workingday	1.606E+006
weathersit	1.602E+006
temp	-
atemp	1.586E+006
hum	-
windspeed	1.455E+006

4 features - 3rd run	
day	1.424E+006
season	-
month	1.345E+006
holiday	1.398E+006
weekday	1.421E+006
workingday	1.407E+006
weathersit	1.399E+006
temp	-
atemp	1.373E+006
hum	-
windspeed	1.319E+006

5 features - 1st run	
day	1.355E+006
season	-
month	1.315E+006
holiday	1.353E+006
weekday	1.335E+006
workingday	1.320E+006
weathersit	1.320E+006
temp	-
atemp	1.327E+006
hum	-
windspeed	-

6 Features - 3rd run	
day	1.307E+006
season	-
month	-
holiday	1.311E+006
weekday	1.277E+006
workingday	1.298E+006
weathersit	1.352E+006
temp	-
atemp	-
hum	-
windspeed	-

The first couple is composed by temp and hum, then this pair performed well together with season, then windspeed, month and weekday were added

For the hours we obtained the following trend:

3 Features – 1st run	
day	1.204E+004
season	1.215E+004
month	1.275E+004
hr	-
holiday	1.221E+004
weekday	1.034E+004
workingday	7.824E+003
weathersit	1.129E+004
temp	-
atemp	1.181E+004
hum	1.099E+004
windspeed	1.173E+004

4 Features – 4th run (10i)	
day	8.313E+003
season	8.573E+003
month	9.010E+003
hr	-
holiday	7.771E+003
weekday	7.794E+003
workingday	-
weathersit	7.485E+003
temp	-
atemp	7.908E+003
hum	6.770E+003
windspeed	8.169E+003

5 Features – 1st run	
day	8.837E+003
season	7.732E+003
month	8.248E+003
hr	-
holiday	7.150E+003
weekday	7.276E+003
workingday	-
weathersit	7.493E+003
temp	-
atemp	6.812E+003
hum	-
windspeed	6.962E+003

So the first pair is composed by hour and temp, then working, hum and atemp were added.

1.1.1.3 Final results and considerations

The final results from the feature selection are summarized in Figures 3 and 4 below, respectively for the day and the hour data.

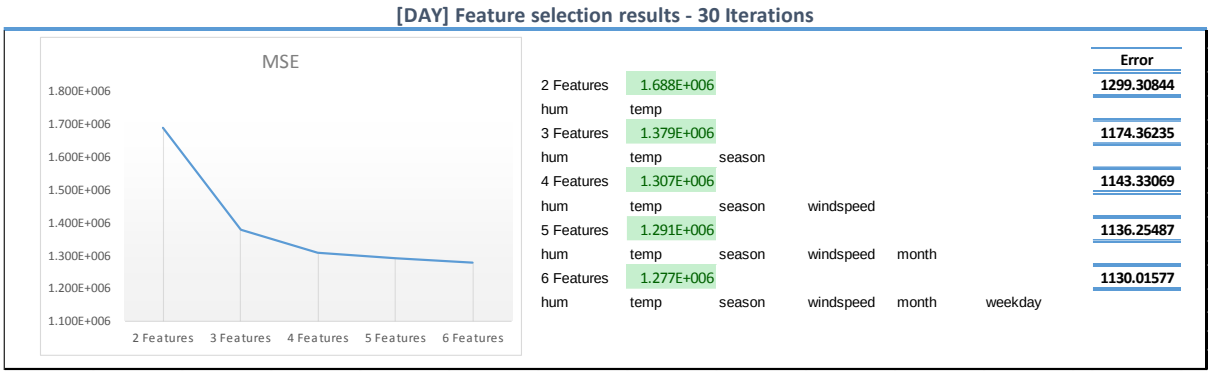

Figure 3 - Day results

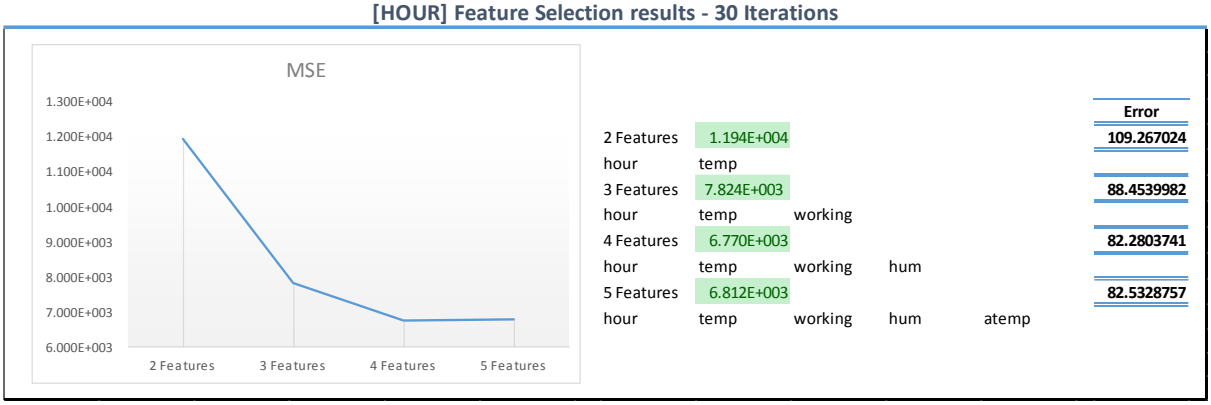

Figure 4 - Hour results

At first glance we notice the difference between the best performing features in the two cases; in the hour table we have a different feature with respect to the other common ones: hour.

Reasoning on the hourly results, we can conclude that in the hour analysis the humidity is much more meaningful w.r.t. the day analysis but this could have been predicted since the hour table contains all the data from the single hours. After some research we verified that the humidity raises during the night, and also it can be reasonable to say that bikes are rented more during the day than during the night.

Based on these results, we decided to select 4 features for each dataset, since even taking more features the net doesn't show any remarkable improvement.

The final selected features for the day dataset are **humidity, temperature**, **season** and **windspeed**; while for the hour dataset the selected features are **hour**, **temperature**, **working day** and **humidity**.

Now that the features have been selected, we may proceed to check whether or not the hidden layer size we used during the feature selection may be further adjusted to improve the performances even more.

To this aim we tested the selected features with different sizes for the hidden layer, and then proceeded plotting the average over 30 iterations. For the day dataset the results are shown in Figure 5. As we can see the performance improves as we increase the hidden layer size, but 10 could however be considered as a good choice for the final network.

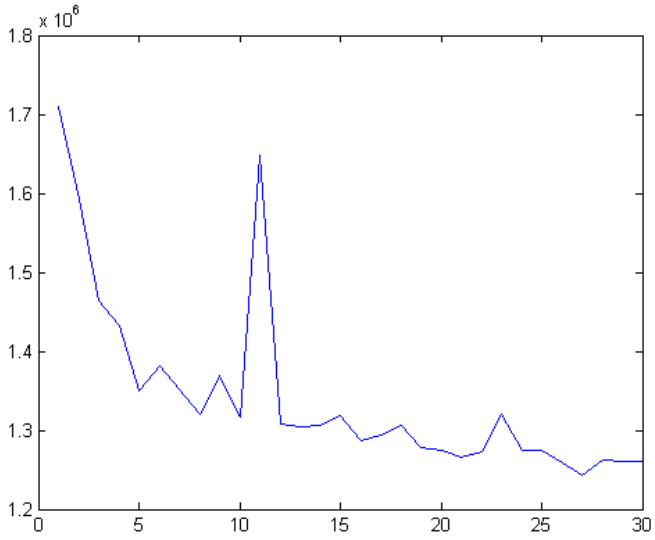

Figure 5 - Day NN with increasing layer size

The same reasoning was made for the hour dataset, and after 30 iterations with different hidden layer size the results are shown in Figure 6.

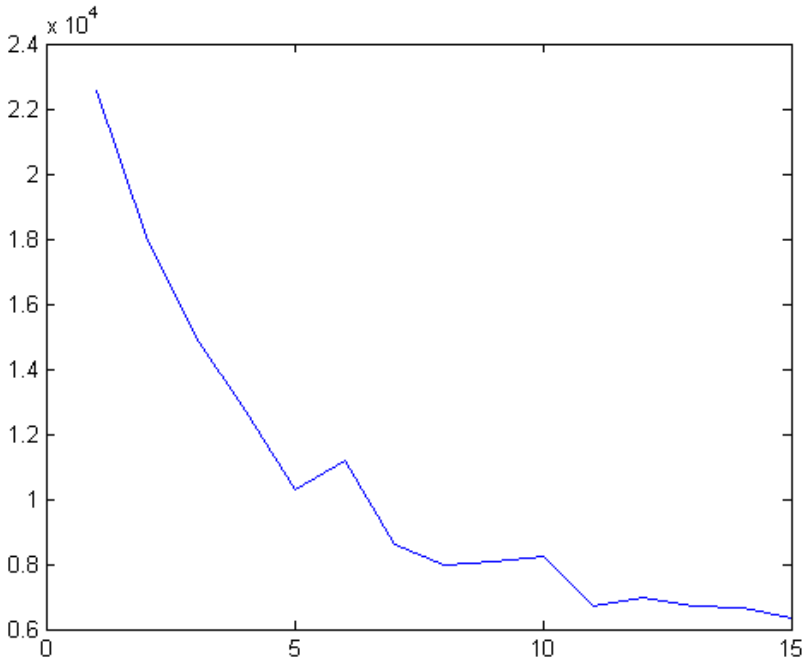

Figure 6 - Hour NN with increasing layer size

Based on these results we obtained the final error values for the two datasets

Day MSE	Hour MSE
1,307E+006	6,770E+003

1.1.2 RBF

The second part of the fitting assignment consisted in the developing of a RBF network for the same purpose of the previously documented network.

For this part of the project the dataset was normalized and not sampled purely randomly, but instead following a precise pattern. The dataset was in fact sampled taking the 70% of every single month as a training set and the remaining 30% as testing set, both in the hour and the day data.

For this purpose a script was developed and reported below (*"Source Code" [5] - dataSetMonthIntervals.m*)

```
function samples = DataSet_Samples( data_set )

%Sets dimension setting
data_set_size = numel(data_set(:,1));
%training_set_size = round(data_set_size * train_ratio);
%test_set_size = data_set_size - training_set_size;

start_idx = 1;
end_idx = 1;
month = data_set(1,5);

intervals = zeros(24, 2);
intervals_row_index = 1;

for row = 2:data_set_size
    row_month = data_set(row,5);
    if (row_month == month)
        end_idx = end_idx + 1;
        %        disp(end_idx);
    else
        if( row_month == 1)
            month = 1;
        else
            if ( row_month == (month + 1))
                month = month + 1;
            else
                fprintf('Unexpected error: end index = %0.0f\n', end_idx);
            end
        end
        % saving start and end indexes
        intervals(intervals_row_index, 1) = start_idx;
        intervals(intervals_row_index, 2) = end_idx;
        intervals_row_index = intervals_row_index + 1;
        % resetting indexes
        start_idx = end_idx + 1;
        end_idx = start_idx;

    end
end
intervals(intervals_row_index, 1) = start_idx;
intervals(intervals_row_index, 2) = end_idx;

day_samples = intervals;
end
```

The script returns the indices of the samples that are going to be used to actually train the network.

Once the data has been subsampled we can finally train and test the network. The implemented script trains a network and tests it with a variable spread, and returns the MSE based on the testing of all the different spread networks.

When using the `newrb` function the goal was set based on the MSE previously obtained with the MLP net. For what concerns the day dataset the goal was reached within the neurons limit, while for the hour dataset the net couldn't reach the goal even with 120 neurons. For this reason we analysed the MSE descent during the training of the network, in order to find the right trade-off between the MSE and the max number of neurons.

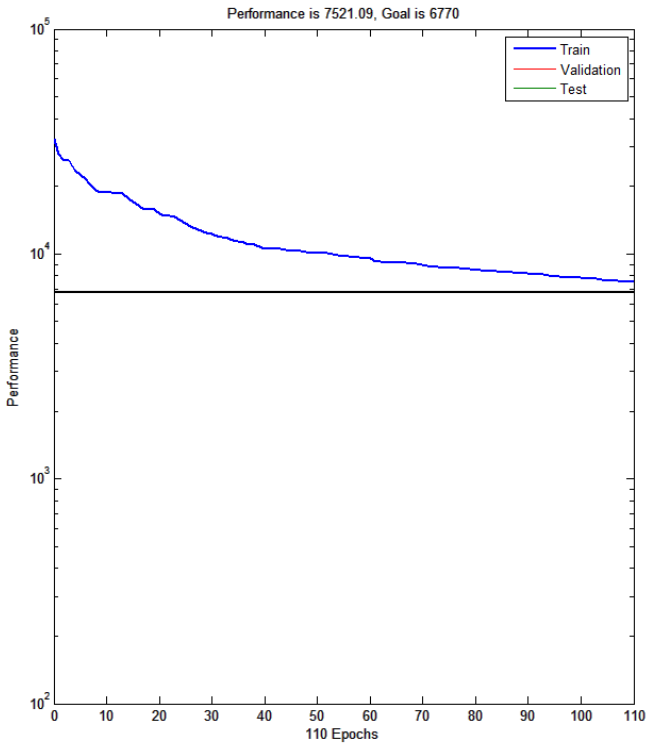

Figure 7 - RBF with 120 as neuron limit

Figure 7 shows the MSE slope for the hour RBF network on a logarithmic scale. The maximum number of neurons for this case was set to 60 for computational reasons.

The final spread analysis MSE are calculated based on a mean of multiple iteration for each single network (30 iterations for the day and 5 for the hour).

The script to test all the spreads is contained in the script *"Source Code" [7] - rbfFittingiter.m:*

```
function [ mse ] = rbfFittingIter( dataSet, features, samplesIntervals, targets,
goal, spreadValues, iterations, labels )
train_ratio = 70/100;
MAX_NEURONS = 60;

fprintf('Features selected = ');
for f = features
    fprintf('%s ', char(labels(f)));
end
fprintf('\n');
data_set_size = numel(dataSet(:,1));
training_set_size = round(data_set_size * train_ratio);

training_set_idx = zeros(training_set_size,1);

numel_samples_intervals = numel(samplesIntervals(:,1));

base_idx = 1;
%building the training set
for i = 1:numel_samples_intervals
    sample_size = samplesIntervals(i,2)-samplesIntervals(i,1)+1;
    training_sample_size = round(sample_size * train_ratio);
    sample_idx_interval = (samplesIntervals(i,1):samplesIntervals(i,2))';
    training_sample_idx = datasample(sample_idx_interval, training_sample_size, 1,
'Replace', false);
    training_set_idx(base_idx:base_idx+training_sample_size-1,1) =
training_sample_idx;
    base_idx = base_idx + training_sample_size;
end

idx = (1:data_set_size)';
test_set_idx = setdiff(idx, training_set_idx);

% Creating training and testing sets
training_set = dataSet(training_set_idx,features);
test_set = dataSet(test_set_idx,features);

fprintf('Training set size = %d \t Test set size = %d\n', numel(training_set_idx),
numel(test_set_idx));


% Computing MSE for each spread value
num_of_spreads = numel(spreadValues);
i = 1;
mse = zeros(num_of_spreads,1);
for s = spreadValues
    fprintf('%0.2f progress - Spread = %0.2f\n', (i*100/num_of_spreads),s);
    for j = 1:iterations
        net = newrb(training_set',targets(training_set_idx)', goal, s,
MAX_NEURONS);
        outputs = sim(net,test_set');
        mse(i,1) = mse(i,1) + perform(net, targets(test_set_idx)', outputs);
    end
    mse(i,1) =  mse(i,1)/iterations;
    i = i + 1;
end
plot(spreadValues,mse);
end
```

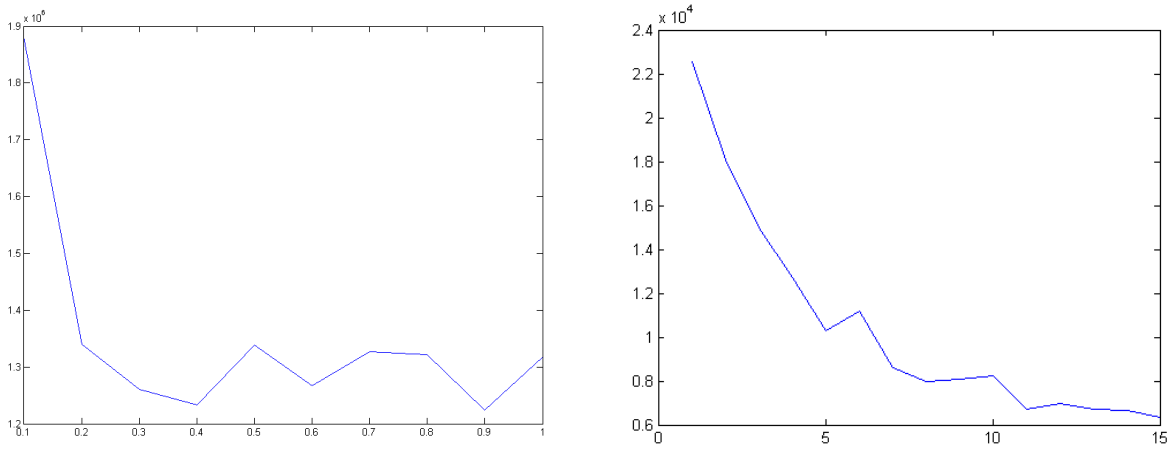

Figure 8 - Hidden layer analysis (day - hour)

For the day data two points of minimum where found, therefore an approach similar to the one of the binary research was used in order to find the actual minimum of the MSE. By doing so we also increased the granularity of the spread values, until we reached the results reported in the table below.

	Spread value
Day dataset	0.4
Hour dataset	0.2

The complete analysis is reported in the pdf file *'/Reports/SpreadAnalysisResults.pdf'*

1.2 Fuzzy model

This part of the project is aimed to develop two fuzzy systems in order to fit the number of bike rentals. In particular a Mamdani and an ANFIS were developed.

1.2.1 Mamdani Fuzzy System

In order to develop a Mamdani system, the rules must be chosen in a heuristic fashion, and for this case study the rules will be driven by bare intuition and inductive reasoning on the data (by means of graphical representations).

The first part of the project covered the understanding of the most meaningful features of the two datasets. Since those features seem to be very relevant, they will be used also in this part of the project, but in this case a further analysis is conducted. For each feature, a scatterplot and a histogram are plotted, in order to extract any possible intuition that could be helpful while defining membership functions and rules. The scatterplots have been made plotting the single features together with the total count of bikes, considering only the portion of the dataset regarding the second year in order to reduce the bias due to the transient data of the first year. The histograms have been generated to have a rough idea of the underlying distribution of the features, thus resulting helpful while designing the membership functions.

The most relevant insights observed are the following:

Day dataset

- Temperature: the scatterplot shows empty regions that let us conclude that a relationship between this feature and the total count of bikes exist and can be used to define a rule.
- Humidity: presents a major concentration around the middle of its support corresponding to mid-high values of the total count, while the other points are lying sparsely in the bottom part of the chart, suggesting that in order to make a rule to correctly identify these samples it is likely to consider this feature joint with at least another one.
- Season: clearly presents a behaviour different during spring and summer, so it is possible to define rules relying on this fact.

Hour dataset

- Humidity: as it could be predicted, humidity in this dataset proves to be strictly correlated to the hours, since on average it is more humid during the night. For this reason the humidity was not taken into consideration during the definition of the membership functions.
- Hour: from the scatterplot the correlation of count in relation to the time of the day is evident.
- WorkingDay: this important feature proves its weight in the scatterplots. Indeed the count of the bikes during the working days is on average higher than the non-working ones.
- Temperature: the scatterplot shows the relation of the bikes in regards to temperature. As the temperature raises also the count increases on average.

The number of bike rentals differs a lot between casual and registered users, so the first design choice was on whether to develop different fuzzy systems for each type of user (which is also strongly correlated with the day of the week) or on the total number of bike rentals. For the sake of simplicity the overall count of bikes was used to develop the systems.

The commented collection of plots is to be found in the report file *'fuzzyReasoning.pdf'*.

1.2.1.1 Membership Functions

To develop the Mamdani system the MATLAB Tool *fuzzy* was used. In particular the first stage was focused on the correct definition of the membership function for each input and output variable.

To this aim the annual behaviour of the system was studied with respect to the chosen features. This was done to better infer the data.

The chosen sets for the two datasets and the fuzzy models are reported in the figures below:

Day Fuzzy System

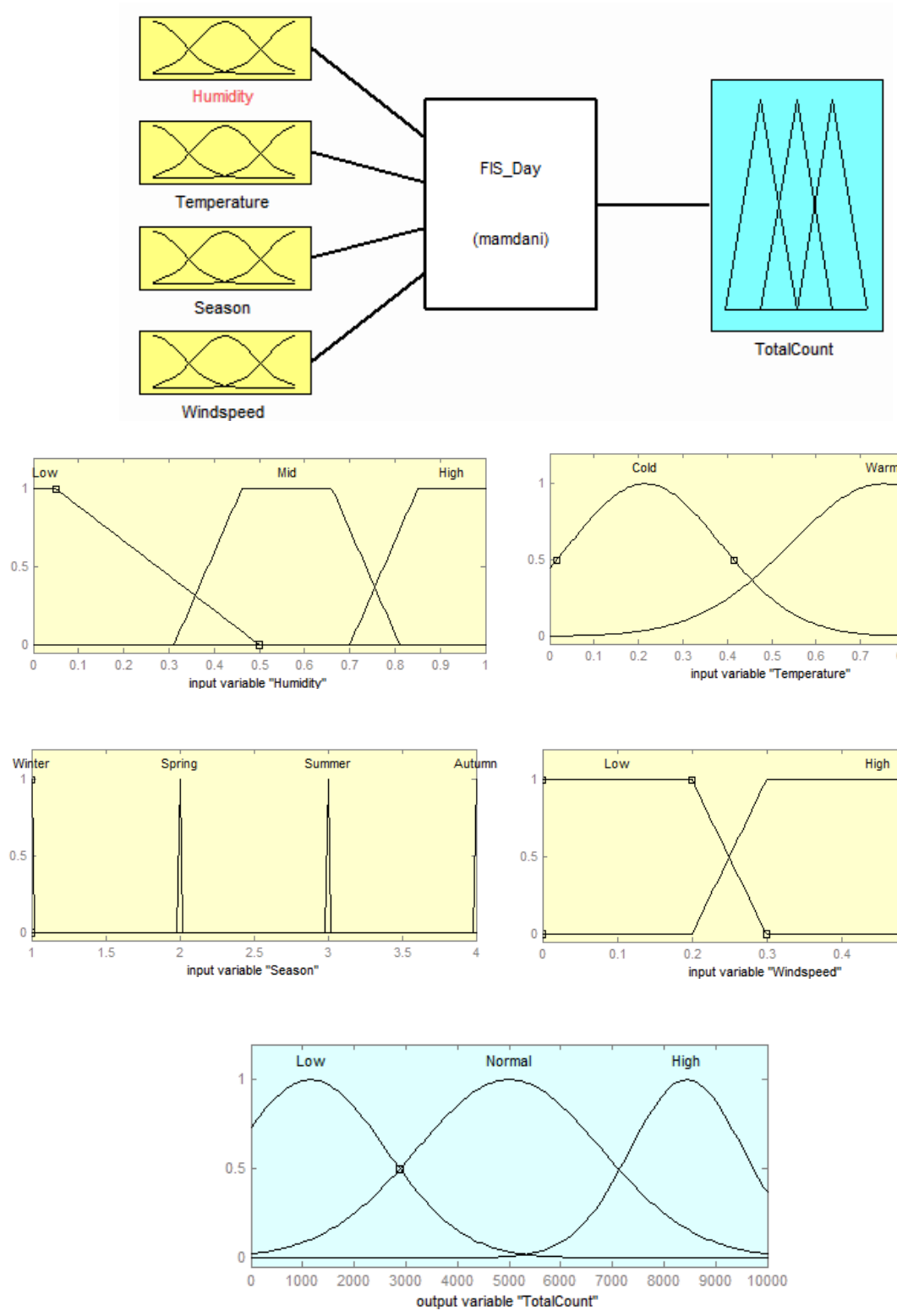

Figure 9 - day FIS

Hour Fuzzy System

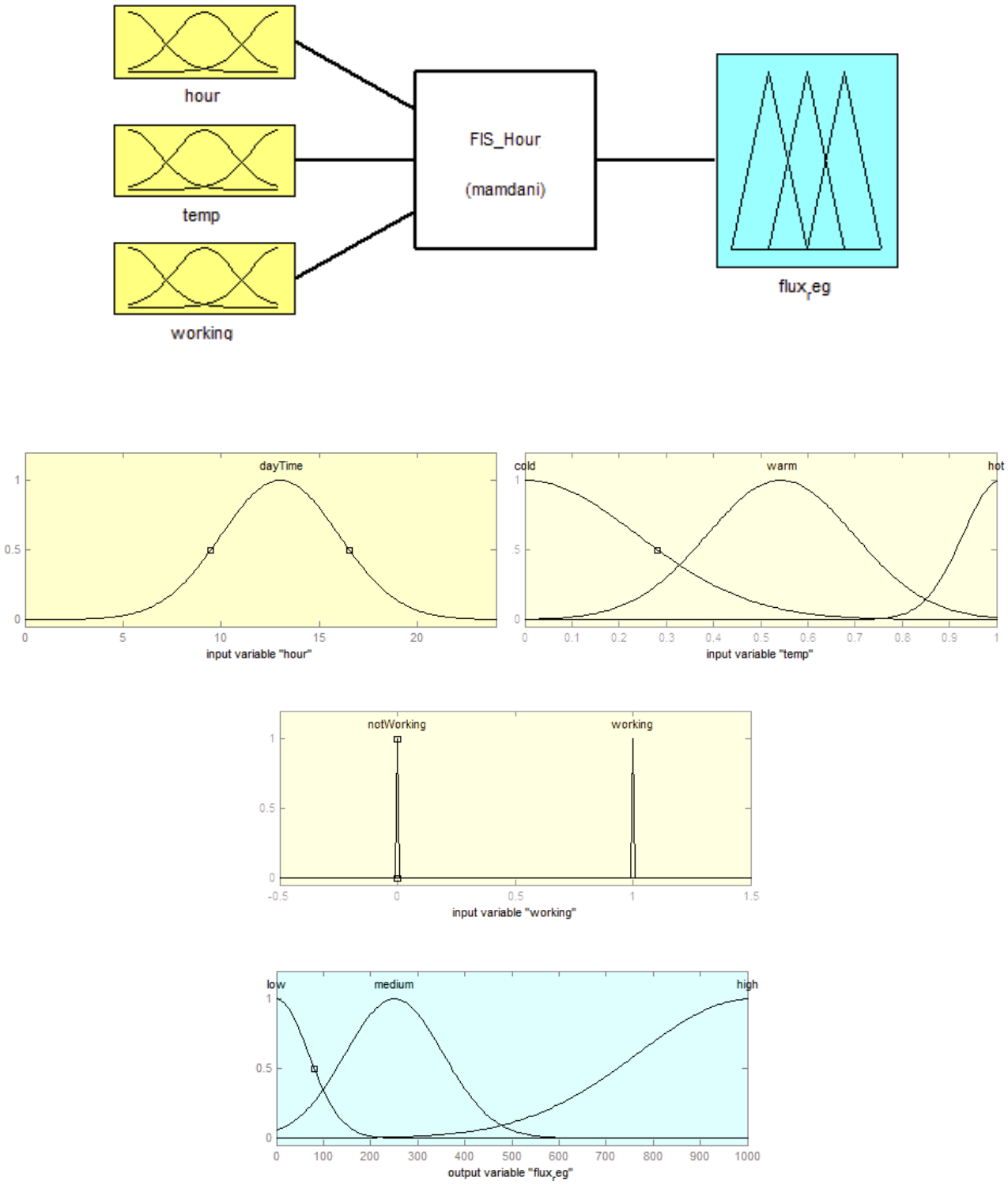

Figure 10 - Hour FIS

The values of the sets were determined by analysing histograms and scatterplots, and are meant to be meaningful in this specific situation, whereas in some other city the membership functions may be defined differently (i.e. in an arid city like Dubai the humidity ranges may be substantially different).

From the figures is evident that a feature common to both system (temperature) is defined in a different manner, indeed also the granularity of the measures was taken into account while resolving the membership function. In reference to the temperature feature: in the day dataset it is reported as a mean of the temperatures experienced during the day, whilst in the hour dataset it's reported on hourly basis; this changes a lot the vision that the system may have on that particular feature, and also influences the reasoning based on that component.

Hour considerations

Since one of the most important features is *workingDay* for the hour dataset, the fuzzy system was model to describe the behaviour of the dataset based on the influence of the work on the bike renting system. For this same reason the output variable chosen for this system was the count of registered users instead of the total one, because the model is striving to describe the overall work impact on the renting system.

Also, when defining the membership functions for the hours they first were defined to describe 3 distinct ranges: morning, noon and afternoon. After discerning the impact of the working factor, the hour membership functions were redefined to work according to the others and to bolster the main target of the model.

As a final consideration, we also deemed that taking workingDay as a feature would be more meaningful than defining a membership function for the weekdays (by taking from Monday to Friday as working ones) since in that case it wouldn’t include the festivities, while workingDay does.

1.2.1.2 Fuzzy Rules

The rules for the fuzzy systems had to be defined for both of the systems separately.

Day Rules

1. *If (Temperature is Cold) then (TotalCount is Low)*
2. *If (Humidity is High) and (Temperature is Cold) then (TotalCount is Low)*
3. *If (Humidity is High) and (Temperature is Warm) then (TotalCount is Normal)*
4. *If (Season is Spring) and (Windspeed is Low) then (TotalCount is High)*
5. *If (Season is Summer) and (Windspeed is Low) then (TotalCount is High)*
6. *If (Humidity is Mid) and (Temperature is Warm) then (TotalCount is Normal)*

These rules, as they show by themselves, clearly come from a purely heuristic approach. For example, it is easy to state that if there is lot of humidity on winter then it could be so cold that just few people would rent a bike instead of taking a bus. The same basic idea underlies the other rules. In some cases, such as for the 1st, 4th and 5th rules, some fair intuitions coming from a rough analysis of the scatterplots have also been used.

Hour Rules

The final rules for the hourly dataset are the following:

1. *If (temp is cold) then (flux_reg is low)*
2. *If (hour is dayTime) and (temp is warm) then (flux_reg is medium)*
3. *If (hour is dayTime) and (temp is hot) and (day is working) then (flux_reg is medium)*
4. *If (hour is dayTime) and (temp is hot) and (day is notWorking) then (flux_reg is high)*
5. *If (hour is not dayTime) or (temp is cold) then (flux_reg is low)*

While the first two rules may sound obvious, rules 3 and 4 verified our hypothesis since they resulted in a remarkable improvement in terms of MSE. Those two rules basically mean that the registered users, when the temperature is high, tend to make extensive use of the bike renting system, even more when they do not have to work (or equally attend university).

1.2.1.3 Results

With the fuzzy systems built as described so far, the final step is to evaluate their performance. The Mean Squared Error (MSE) has been adopted as metric and it has been computed as described in *"Source Code" [8] - evalMyFIS.m*. The FIS has been fed with all the rows in the data sets, but selecting only the features corresponding to the input variables defined in the two FISs. The final MSE values obtained are reported below:

Day	Hour
2.4372e+06	1.8097e+04

Comparing these results with the ones recorded with the MLP and RBF neural networks, it turns out that the fuzzy systems perform worse and this may be due to the simplicity of the rules defined and to the design of the membership functions. One way to improve such performances might be changing the rules and fine tuning the membership functions.

1.2.2 ANFIS

An *Adaptive Neuro-Fuzzy Inference System* makes it possible to build a fuzzy inference system where the membership function parameters are automatically tuned using an adaptive learning method. To accomplish the task of building such a system, the following choices have been made:

Data set partitioning: the data have been divided randomly into three subsets, the training, testing and checking respectively, and according to the proportions 70%, 15% and 15% of the data set size. To this aim, a function has been defined and it's described by the '*dataset_partitioning.m*' script. Some attention has been drawn to the actual meaning of our data set. Since its data covers two distinct years and the biasing effect of the first one has already been noticed, in this case the sampling has been carried out in such a way that the final subsets equally contain samples coming from both the two years. This led to a huge overall improvement of the ANFIS performance.

FIS generation: the *Grid partition* method has been used.

FIS training: it has been set an *error tolerance* of zero since the training error behaviour is not known at the beginning, and the number of *epochs* is equal to *40*. The *optimization method* has been chosen to be *hybrid*.

Membership functions: several trials varying the shape of the membership function have been made for the input variables; the output function was set to *constant*.

FIS testing: the performance has been evaluated with the testing set and the scores reported.

Hour	
Feature	**Number of M.F.s**
Hour	1
Workingday	2
Temp	3

Day	
Feature	**Number of M.F.s**
Humidity	3
Temp	2
Season	4
Windspeed	2

Daily data set results		
MF type	**Training Error (Check. error)**	**Testing Error**
Trimf	471.37 (528.984)	522.5639
Trapmf	499.138 (550.224)	527.576
Gbellmf	461.062 (507.772)	478.1122
Gaussmf	467.014 (498.597)	499.7374
Gauss2mf	**453.603 (479.622)**	**475.6232**
Pimf	511.231 (550.459)	518.7673
Dsigmf	479.22 (533.631)	540.9319
Psigmf	479.22 (533.632)	540.9321

Hourly data set results		
MF	**Training Error (Check. error)**	**Testing Error**
Trapmf	117.486 (117.743)	117.6158
Gaussmf	117.7 (118.167)	118.0606
Gauss2mf	117.609 (117.958)	117.85

When dealing with the hourly data set, a configuration with 3 M.F.s for the '*hour*' feature has been made; all the M.F.s were *trapmf*. The results are the following:

trapmf : train error = 92.7725 (93.4153) test error = 92.3996.

gaussianmf: train error = 93.7088 (95.038) test error = 93.4211.

1.2.2.1 Results analysis

The results reported bring us an evidence: the ANFIS built on the hourly data set performs better than the one developed on the daily data set. Moreover, the two FISs structures are quite different: the former has a leaner network than the latter, meaning that it is simpler in its architecture. These two results may be stating that ANFIS systems perform better as the sample size increases.

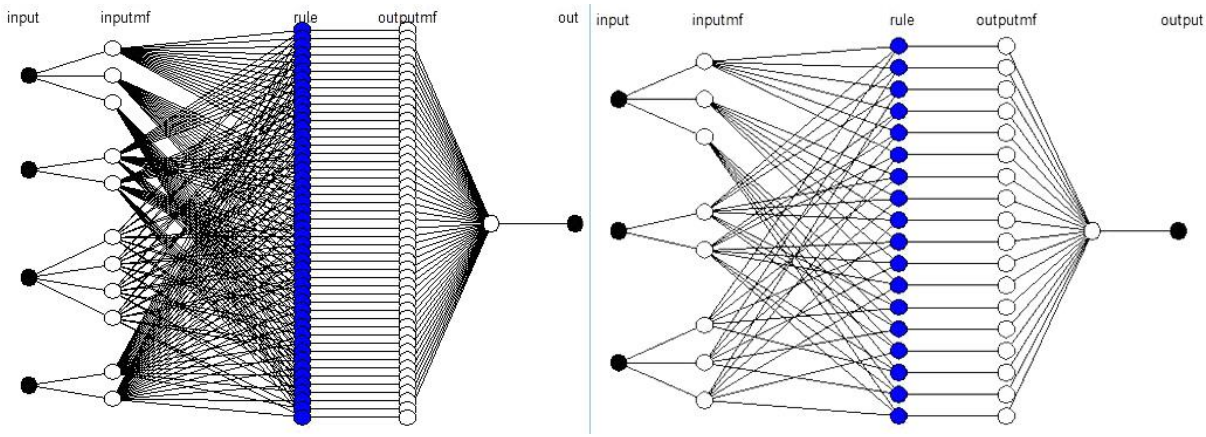

Figure 11 - Day and Hour network structure

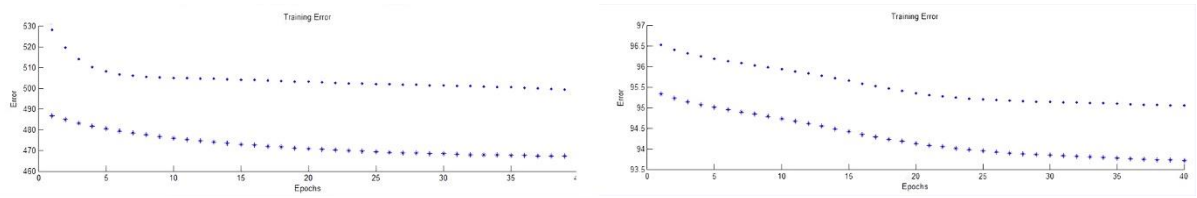

Figure 12 - Day and Hour Gaussian training

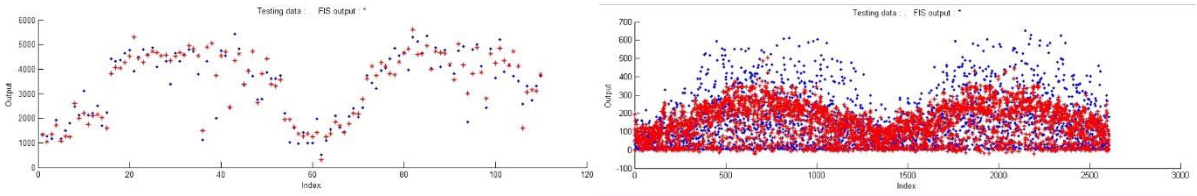

Figure 13 - Day and Hour testing

Part II

Forecasting Model

This part of the project is aimed at actually predicting the total number of the bike rentals based on seasonal and weather related features.

We were asked to first develop the forecasting model, and subsequently to evaluate its forecasting performances.

For what concerns the dataset, also in this case some consideration were made. The time series tool will automatically sample the input stream to extract the training, validation and testing data. As for the previous parts of this project the sample should describe the dataset entirely, and the reasoning made for the RBF may be applied. Albeit sampling 70% of each month for the testing set may look appealing, we finally decided to let the tool handle it since we will compute the average over many independent iterations. By doing so, the sample is guaranteed to be equally representative of the data.

2.1 Forecast based on open loop

The data has been divided in two years, and the first one has been used to develop a forecasting model able to predict the number of bikes based on a subset of the available features.

The first step consisted on determining which features would actually help the prediction, and after that determine the number of delays and hidden neurons for the network.

Since the delay choice depends on the chosen features, the feature selection was the first analysis that has been carried out. This is because giving a delay of 2 to a feature like season may not be useful at all.

Even if those choices should be heuristic, some plots helped the decision process.

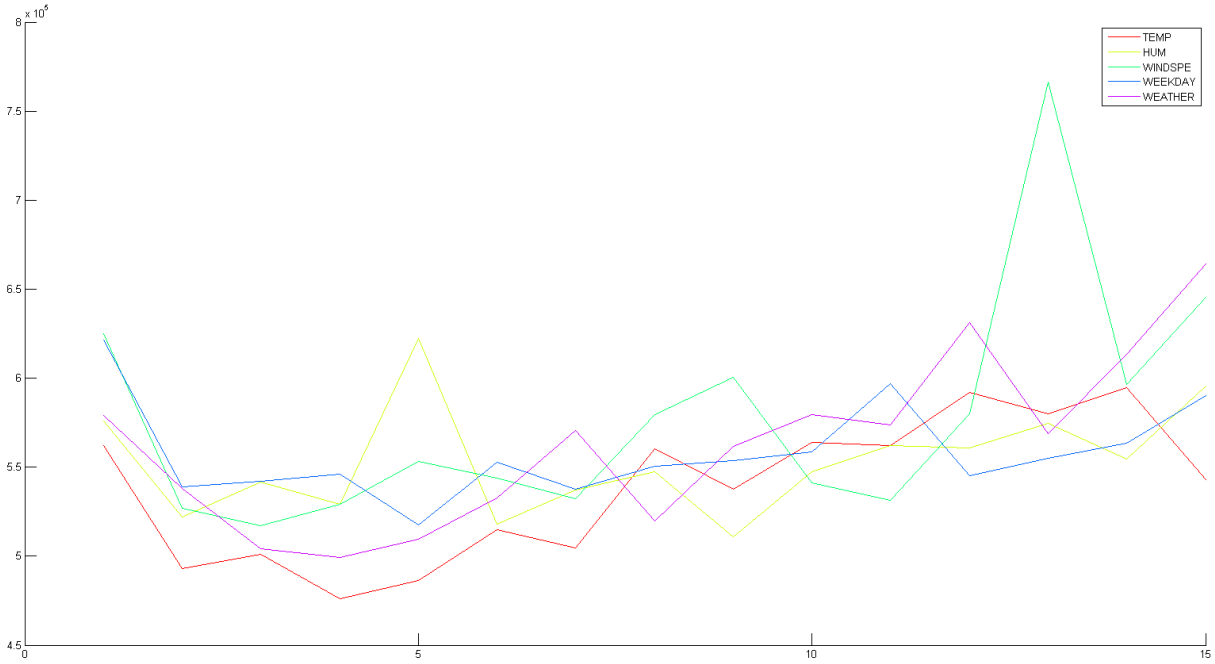

Figure 14 - Single feature performance

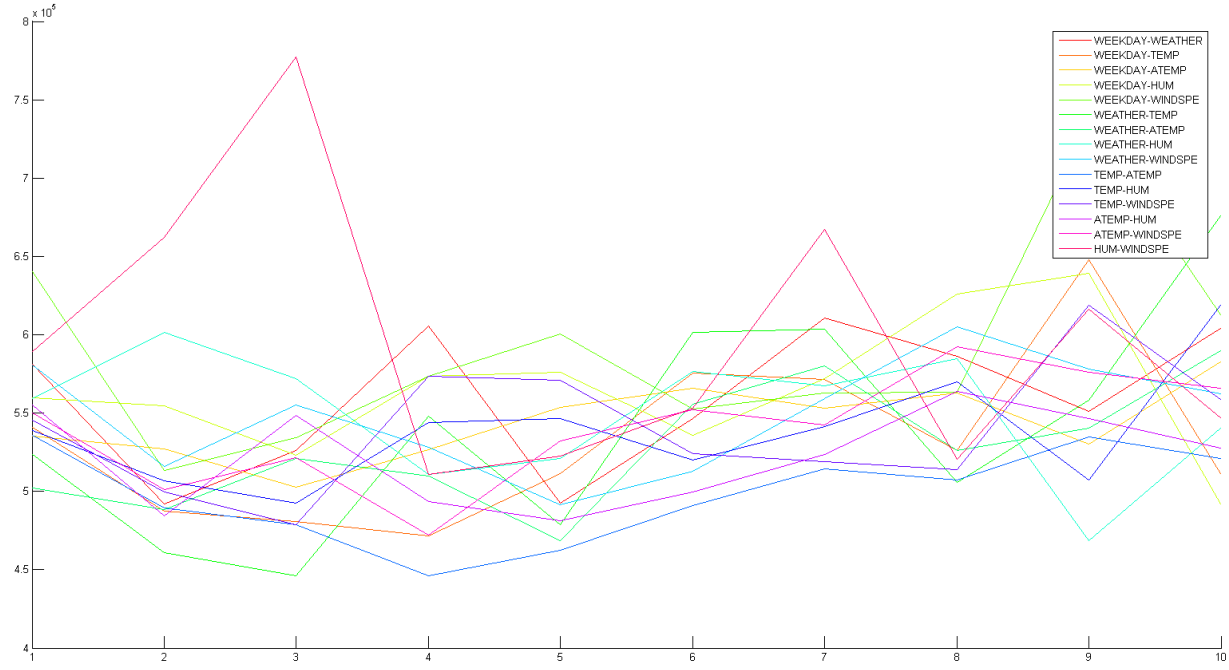

Figure 15 - Pairs of features performance

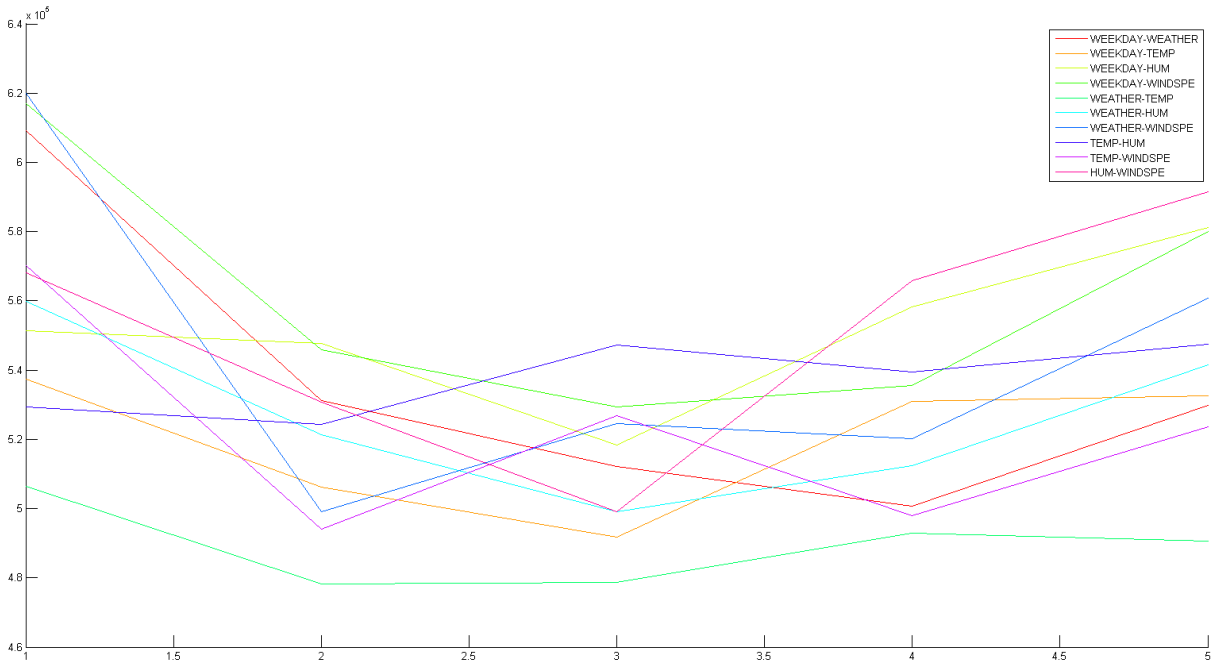

Figure 16 - Pair of features (zoom)

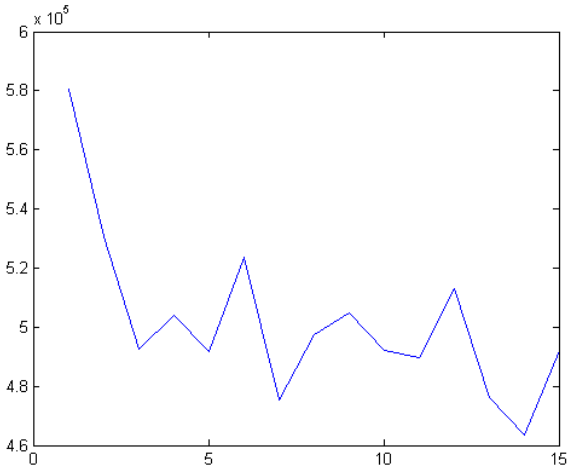

Figure 17 - Hidden Layer choice

Based on these plots, we defined the final feature set as: weather, temp, hum and weekday.

We've also chosen 2 as a value for the delay and 14 (figure 17) for the hidden neurons.

The final MSE obtained with these values is 4.6117e+05, which accounts to an error of around 670 in the overall rented bikes number

The following plot in figure 18 displays the error autocorrelation function. It describes how the prediction errors are related in time. Ideally, there should only be only one nonzero value of the autocorrelation function in zero lag representing the actual MSE, and it would mean that the prediction errors were acting as white noise since they are completely uncorrelated with each other.

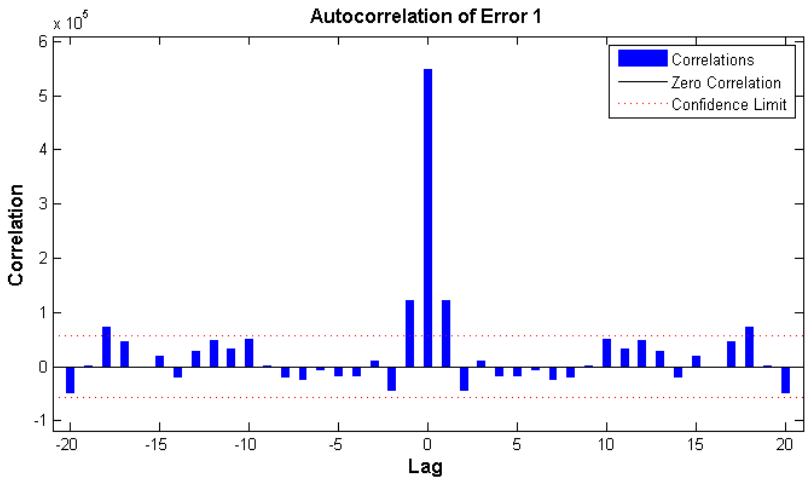

Figure 18 - Error autocorrelation

In our case, except for the zero and one lags, the correlation almost always falls within the 95% confidence limits around zero represented by the red lines ($\pm 2\sqrt{n}$), so the model seems to be adequate.

2.2 Forecast based on closed loop

Now that the network is optimized, we can proceed to actually use it to forecast the number of bike rentals. To do so, after training the net with the first year, we proceeded by closing the loop of the NARX network to evaluate the MSE of the prediction based on the second year data. The resulting network is illustrated in the figure below:

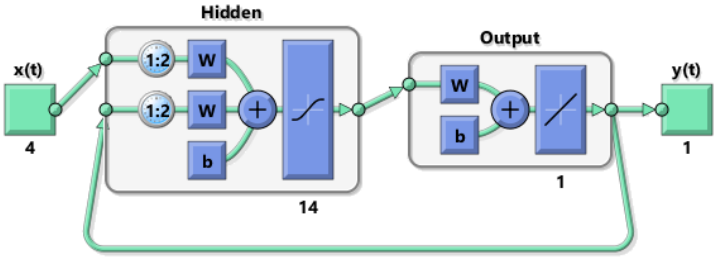

Figure 19 - Closed network

After preparing the closed network, it could be tested by feeding it with the second year data. Figure 20 shows a particularly lucky case, where the blue line represents the actual number of bikes rented, and the red line represents the forecast. We refer to this figure as a lucky one because the prediction is very variable, since it depends on how "good" is the random sampling in the training process.

The performance of the forecasting system could be further improved by training the net several times before closing the loop.

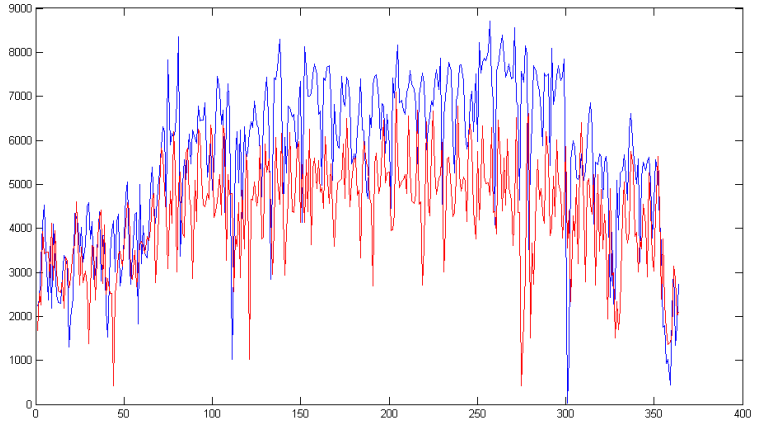

Figure 20 - One day ahead prediction over a year

To evaluate the average forecasting performance for a specific set of days ahead the closed network was tested with 30 different uniformly distributed dates taken at random, checking the performance respectively at 1, 2 , 7, 10 and 15 days ahead. When testing the net with the data, the delay was taken into account, shifting the testing data 2 days before the given starting date.

```
targetSeriesClosed = targetSeriesTotal((dates(i-delay):(dates(i)+daysAhead(j)-1));
```

By doing these steps it was possible to compare the two different approaches in terms of MSE. Results are shown in Figure 21.

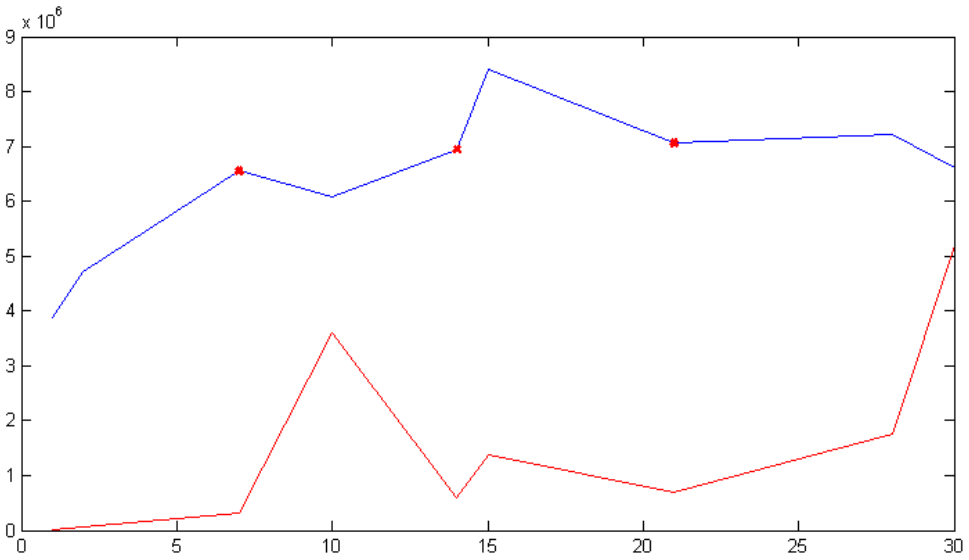

Figure 2 - Two approaches to forecasting

By analysing the above figure, it can be noticed some kind of periodicity in the MSEs. This could derive by the fact that, even if we take the average over 30 iterations, the feature

weekday is one on which the forecast is based. By taking that feature, independently from the starting date (be it Monday or Friday) a similar MSE will occur again after one week (while there is no significant change in temperature and humidity in the course of a week). This can be reiterated to justify the similar MSEs after 7, 14, 21 or 28 days, even if the similarities fade with time. Inspecting the same figure is evident that the forecast one day ahead tends to be more precise in comparison with the 15 days ahead, but this is reasonable as we lose confidence in the forecast when increasing the forecasted period.

It is also possible to see that at the beginning, in the early days of January 2012, the forecasts result closer to the actual values registered. This is because those days reported similar characteristics (in terms of the selected features) to the days of 2011, used during the training. While moving away from those days, the trend of the number of bike rentals grows, keeping the same shape as the first year but with an offset, since the service became more popular in 2012. Thus, the forecasting system can follow the trend of bike rentals as learnt from the first year, whilst it encounters some difficulties while predicting such a variation due to a factor not represented by any of the given features.

Conclusions

This survey has documented the analysis and the study of a bike sharing system dataset in order to find insights on the data and predict the possible number of bike rentals.

Different intelligent systems have been designed and developed: Multi-Layer Perceptron, Radial Basis Neural Network, Fuzzy Inference System, ANFIS and dynamic neural network for Time Series Prediction. Despite their intrinsic different nature, all the approaches achieved reasonable performance with just few features.

This remarkable result is due to an adequate features selection process and to the tuning of the systems parameters, such as the number of neurons, spread and membership functions. Of course finer results could be achieved: spending more time to finely tune the systems' parameters or performing a more in-depth data analysis would result in more accurate results. It is easy to understand that more accuracy does not come for free: it would be case that more complexity would arise in the systems, in terms of required data, number of neurons or fuzzy rules. Thus, our design results to be quite light and simple, from both the computational and structural sides, producing readable and consistent results. We decided to keep the number of required features low, just three or four, because most of the improvement of the system is already achievable with this choice.

In a comparison among the different developed systems, the fuzzy inference systems would be the unquestionable winner. FIS and ANFIS have reported much better performance than the other networks, and this could be explained by the following: neural networks learn to generalize from data, so they try to exploit input punctuality to get to the underlying meaning of the whole data set. Fuzzy inference systems apply a different approach: define a ("rough") general behaviour of the phenomenon and use this definition to infer from the given data set. Therefore FIS has greater flexibility than neural networks, and thus makes possible to smooth out all the localities coming from the data and directly intercept the real nature of the modelling phenomenon.

Finally, the dataset object of the study represents just the first two year of the service. In this case we are striving to find some meaning at what seems to be a transitory state of the system, which will become more and more stable as the years pass by.

Appendix

This last section contains a list of all the scripts needed for the project, followed by the required command lines needed to run the scripts. More examples to be found in the `readme` file.

Part I

Files needed

init.m

MLP

fitFeatureSize.m
mlp2Features.m
mlpNFeatures.m

RBF

dataSetMonthIntervals.m
rbfFitting.m

Fuzzy

evalMyFIS.m
datasetPartitioning.m

Commands examples

Init

MLP

mlp2Features(data_day,cnt_day,[10 12 3],10,10,data_day_labels)
mlpNFeatures(data_day,cnt_day,[10 12],[3 11 13],10,10,data_day_labels)

RBF

hour_months_intervals = dataSetMonthIntervals(data_day)
rbfFitting(data_hour,[10 12],hour_months_intervals,cnt_hour,1.0e+06,0.1:1)

Fuzzy

evalMyFIS(FIS_Day, data_day(:,[12 10 3 13]), cnt_day)
datasetPartitioning(data_day, 0.7, 0, 0.3, [10, 12, 13])

Part II

Files needed	*Commands*
Open Loop	**Open Loop**
NTS_Day.m *Day_Forecast.m* *Day_Forecast2Features.m* *Day_ForecastNFeatures.m* *Day_ForecastHiddenLayer.m*	*DayForecastFeatures (data_day, cnt_day,[7 9 10 11 12 13],15,10,data_day_labels);* *DayForecast2Features(data_day, cnt_day,[7 9 10 11 12 13],5,10,data_day_labels);* *DayForecastNFeatures(data_day, cnt_day,[7 13 11],5,10,data_day_labels);* *DayForecastHiddenLayer(data_day, cnt_day, 2,15);*
Closed Loop	**Closed Loop**
ntsDayClosed.m *dayForecastingClosed.m*	*dayForecastingClosed(data_day, cnt_day,[7 9 10 12], 2, 14, 1, [1 2 7 10 15]);*

External Files

Source Code.pdf

Spread Analysis Results.pdf

Fuzzy Reasoning.pdf