Bike renting project

Commented MATLAB scripts

This document contains the main scripts needed for the project.
Each script comes with a brief description.

1- Init

This script loads the data from the hour and day dataset files and formats the labels for further use during the project.

```
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

data_day = xlsread('C:\Users\Alessio\Google Drive\Bike Sharing Project\Egidi_Villardita - Bike Sharing [IS Project]\data\dayData.xlsx','','','basic');
data_hour = xlsread('C:\Users\Alessio\Google Drive\Bike Sharing Project\Egidi_Villardita - Bike Sharing [IS Project]\data\hourData.xlsx','','','basic');

data_day_labels = cellstr(['instant'; 'day    '; 'season '; 'yr     ';
    'mnth   '; 'holiday';   'weekday'; 'working'; 'weather'; 'temp   ';
    'atemp  '; 'hum    '; 'windspe'; 'casual '; 'registe'; 'cnt    ']);

data_hour_labels = cellstr(['instant'; 'day    '; 'season '; 'yr     ';
    'mnth   '; 'hour   '; 'holiday';    'weekday'; 'working'; 'weather';
    'temp   '; 'atemp  '; 'hum    '; 'windspe'; 'casual '; 'registe';
    'cnt    ']);

data_day_labels = upper(data_day_labels); %just output styling
data_hour_labels = upper(data_hour_labels);

%Initialize target vectors
cnt_day = zeros(731,1);
cnt_hour = zeros(17379,1);

%Fill target vectors
cnt_day = data_day(1:731,16);
cnt_hour = data_hour(1:17379,17);
```

2 - fitFeatureSize.m

This script creates a neural network for the given input and target with the number of hidden neurons passed ad a parameter. It returns the mean square error.

```
function performance = fitFeatureSize(input, target, hiddenLayerSize)
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

inputs = input';
targets = target';

% Create a Fitting Network
net = fitnet(hiddenLayerSize);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs);
end
```

3 - mlpNFeatures.m

Computes the performance MSE for each subset of features on the given data set and target, with the specified set of fixed features, number of neurons and by adding the given features in 'featuresIndexes' one by one to the base features set. 'it' specifies the iterations, i.e. many times the same subset of features must be evaluated. 'featuresLabels' contains the labels created in the 'init.m' script.

Returns all the computed MSEs and prints the best subset found.

```
function [ day_results_nf ] = mlpNFeatures( dataSet, target, fixedFeatures, featuresIndexes,
neurons, it, featuresLabels )
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

fprintf('Training with %d features FIXED:', numel(fixedFeatures));
for i = fixedFeatures
    fprintf('%s\t', char(featuresLabels(i)));
end
fprintf('\n');

numFeat = numel(featuresIndexes);

day_performance = 0;
day_results_nf = zeros(numFeat,1);

j = 1;
for i = featuresIndexes
        sum = 0;
        for it_num = 1:it
            day_performance = fitFeatureSize(dataSet(:,[fixedFeatures i]), target, neurons);
            sum = sum + day_performance;
        end;
        day_results_nf(j) = sum/it;
        fprintf('%0.2f%% Feature: %s with error = %0.3e\n',(j * 100/numFeat),
char(featuresLabels(i)), day_results_nf(j));
        j = j + 1;
end;

bestFeat = featuresIndexes(1);
bestPerf = day_results_nf(1);

for j = 2:numFeat
    if(day_results_nf(j) < bestPerf)
        bestPerf = day_results_nf(j);
        bestFeat = featuresIndexes(j); %retrieving best feature index
    end
end;

fprintf('\nBest feature: %s with error = %0.3e\n', char(featuresLabels(bestFeat)), bestPerf);
```

4 - mlp2Features.m

Computes the performance MSE for each possible couple of features of the given data set and target, with the specified set of features and number of neurons. The field 'it' specifies the iterations, i.e. how many times the same couple of features will be evaluated.

The field 'featuresLabels' contains the labels created in the 'init.m' script.

Returns all the computed MSEs and prints out the best couple found.

```
function [ day_results_2f ] = mlp2Features( dataSet, target, featuresIndexes, neurons, it,
featuresLabels )
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

fprintf('Training with all the possible couples of features\n');

data_day = dataSet;
cnt_day = target;
indexes = featuresIndexes;
numIt = it;
data_day_labels = featuresLabels;

numFeat = numel(indexes);


% upper triangular matrix expected, fix 1 feature and couple with the remaining 11; 11*12 / 2
= 66 couples
day_results_2f = zeros(numFeat-1,numFeat-1);

progress = 0;

k = 1; %used to properly set j's values
for i = indexes
    l = k;
    jIndexes = indexes;
    jIndexes(1:k) = []; % removing first k elements
    for j = jIndexes
        sum = 0;
        for it_num = 1:numIt
            day_performance = fitFeatureSize(data_day(:,[i j]), cnt_day, neurons);
            sum = sum + day_performance;
        end;
        day_results_2f(k, l) = sum/numIt;
        progress= progress + 1;
        %printing like "10% Feature (YR,MONTH) with error = 1.0001e5"
        fprintf('%0.2f%% Features: (%s,%s)\n',(progress * 100)/((numFeat-1)*numFeat/2),
char(data_day_labels(i)), char(data_day_labels(j)));
        l = l + 1;
    end;
    k = k + 1;
end;
```

```
bestFeatA = indexes(1);
bestFeatB = indexes(2);
bestPerf = day_results_2f(1,1);

for i = 1:numFeat-1
    for j = i:numFeat-1
        if(day_results_2f(i,j) < bestPerf)
            bestPerf = day_results_2f(i,j);
            bestFeatA = indexes(i); %retrieving best feature index
            bestFeatB = indexes(j+1); %retrieving best feature index
        end
    end;
end;
fprintf('\nBest couple of features: (%s,%s) with error = %0.3e\n',
char(data_day_labels(bestFeatA)), char(data_day_labels(bestFeatB)), bestPerf);
end
```

5 - dataSetMonthIntervals.m

Explores the given dataset and for each month records its starting and ending index (i.e. the indexes corresponding to the recording order of the dataset). As result, all the indexes found will be used to apply sub-sampling over all the dataset, but on a monthly base.

```
function month_intervals = dataSetMonthIntervals( dataSet )
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

%Sets dimension setting
data_set_size = numel(dataSet(:,1));

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

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

for row = 2:data_set_size
    row_month = dataSet(row,5);
    if (row_month == month)
        end_idx = end_idx + 1;
    else
        if( row_month == 1 || row_month == (month + 1))
            month = row_month;
        else
            fprintf('Unexpected error: end index = %0.0f\n', end_idx);
        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;

month_intervals = intervals;
end
```

6 - rbfFitting.m

Starting from a data set and its set of features, for each spread value defined in 'spreadValues', creates a RBF NN, tests it and returns a column array of MSEs.

Data set division: here the given data set is divided into two subsets: the training and the testing sets. More in detail, the two are generated starting from a set of partitions on the data set. That is: the training set is generated by sampling 70% of elements for each partition defined in 'samplesIntervals'. It has been used to perform a monthly based sampling, so to achieve lower MSEs.

```
function [ mse ] = rbfFitting( dataSet, features, samplesIntervals, targets, goal,
spreadValues )
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]


train_ratio = 70/100;
%spread_values = 0.1:0.1:1;
MAX_NEURONS = 120;

%Setting sets dimensions
% Compared with dividerand, the resulting data sets differ only when
% working with the hour data set: in this case, the training set has 2 more
% samples (12167) than the one resulting from dividerand (12165).

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);
    net = newrb(training_set',targets(training_set_idx)', goal, s, MAX_NEURONS);
    outputs = sim(net,test_set');
    mse(i,1) = perform(net, targets(test_set_idx)', outputs);
    i = i + 1;
end
plot(spreadValues,mse);
end
```

7 - rbfFittingIter.m

Same as rbfFitting.m but here repeating the training as many times as specified by the parameter 'iterations', so to generate #'iterations' samples.

Data set division: here the given data set is divided into two subsets: the training and the testing sets. More in detail, the two are generated starting from a set of partitions on the data set. That is: the training set is generated by sampling 70% of elements for each partition defined in 'samplesIntervals'. It has been used to perform a monthly based sampling, so to achieve lower MSEs.

```
function [ mse ] = rbfFittingIter( dataSet, features, samplesIntervals, targets, goal,
spreadValues, iterations, labels )
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]


train_ratio = 70/100;
MAX_NEURONS = 60;

fprintf('Features selected = ');
for f = features
    fprintf('%s ', char(labels(f)));
end
fprintf('\n');

%Setting sets dimensions
% Compared with dividerand, the resulting data sets differ only when
% working with the hour data set: in this case, the training set has 2 more
% samples (12167) than the one resulting from dividerand (12165).

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
```

8 - evalMyFIS.m

Computes the MSE given FIS, inputs and targets. A FIS must be created prior to call evalMyFIS using the "fuzzy" tool.

```
function [ mse ] = evalMyFIS( fis, input, target )
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

fis_output = evalfis(input, fis);
se = (fis_output - target).^2;

mse = sum(se) / numel(se);

end
```

9 - dataSetPartitioning.m

Divides the given data set into three subsets each one with a size proportional to its respective percentage. The 'features' input refers to the list of indexes that have to be used to select the columns for the final subsets.
In order to have equal proportion of samples coming from the two years in the three data sets, the partitioning process is done twice, one for each year.

```
function [ anfis_day_tr, anfis_day_tst, anfis_day_chk ] =
datasetPartitioning(dataset, p_tr, p_tst, p_chk, features)
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

[ idx_tr_1y, idx_tst_1y, idx_chk_1y ] = dividerand(numel(dataset(1:365,1)), p_tr,
p_tst, p_chk);
[ idx_tr_2y, idx_tst_2y, idx_chk_2y ] = dividerand(numel(dataset(366:731,1)), p_tr,
p_tst, p_chk);
anfis_day_tr = dataset([idx_tr_1y idx_tr_2y], features);
anfis_day_tst = dataset([idx_tst_1y idx_tst_2y], features);
anfis_day_chk = dataset([idx_chk_1y, idx_chk_2y], features);

end
```

10 – NTSDay.m

Solves an Autoregression Problem with External Input with a NARX Neural Network. This is the script generated by the ntstool slightly edited to meet some customizations. The delay and the hidden layer size are passed as parameters. Returns the mse of the network.

```
function [ mse ] = NTSDay (inputSeries, targetSeries, inputDelays, hiddenLayerSize)

% Fix data format
inputSeries = tonndata(inputSeries,false,false);
targetSeries = tonndata(targetSeries,false,false);

% Create a Nonlinear Autoregressive Network with External Input
feedbackDelays = inputDelays;
net = narxnet(inputDelays,feedbackDelays,hiddenLayerSize);

% Prepare the Data for Training and Simulation
% The function PREPARETS prepares timeseries data for a particular network,
% shifting time by the minimum amount to fill input states and layer states.
% Using PREPARETS allows you to keep your original time series data unchanged, while
% easily customizing it for networks with differing numbers of delays, with
% open loop or closed loop feedback modes.
[inputs,inputStates,layerStates,targets] = preparets(net,inputSeries,{},targetSeries);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,inputs,targets,inputStates,layerStates);

% Test the Network
outputs = net(inputs,inputStates,layerStates);
%errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs);

mse = performance;
% View the Network
% view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotregression(targets,outputs)
%figure, plotresponse(targets,outputs)
%figure, ploterrcorr(errors)
%figure, plotinerrcorr(inputs,errors)

end
```

11 – DayForecastFeature.m

Computes the MSE values of different nets, each with a different feature as input. Each of those networks is evaluated with different delay, and their MSE is stored in a vector. Plots the averaged (over a number of iterations defined as a constant inside the script) MSE values of each feature for each delay within the limit.

```
function [ mse_values ] = DayForecastFeature (dataset, target, features, delays,
hiddenLayerSize, data_day_labels)
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

NUM_IT = 30;
%Data setup
inputSeries = dataset(1:365,:);
targetSeries = target(1:365);

num_features = numel(features);

fprintf('Training with %d features FIXED.\n', num_features);

mse_values = zeros(num_features,delays);

% iterate for each feature
for j = 1:num_features
    fprintf('Training with %s.\n', char(data_day_labels(features(j))));
    for i = 1:delays
        temp = 0;
        for it = 1:NUM_IT
            temp = temp + NTSDay(inputSeries(:,features(j)), targetSeries, 1:i,
hiddenLayerSize);
        end
        mse_values(j, i) = temp / NUM_IT;
    end
end

% plot with different colours
plot_legend = cell(num_features, 1);
color_map = hsv(num_features);
clf;
hold on;
for i = 1:num_features
    plot(1:delays,mse_values(i,:), 'color', color_map(i, :));
    plot_legend{i} = char(data_day_labels(features(i)));
end
legend(plot_legend);
hold off;

end
```

12 – DayForecast2Features.m

Computes and plots the MSE of a forecast system in relation to different delays and to all the possible pairs of features. The number of neurons is fixed. The number of iterations is defined in the script and it stands for the number of times the MSE will be calculated for the same settings. The overall MSE for each layer size is then calculated taking the averages of the multiple runs. The script finally returns a plot of all the features pairs in relation to the different delay values

```
function [ mse_values ] = DayForecast2Features (dataset, target, features, delays,
hiddenLayerSize, data_day_labels)
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]
 NUM_IT = 30;
%data setup
inputSeries = dataset(1:365,:);
targetSeries = target(1:365);
num_features = numel(features);
num_couples = (num_features-1)*num_features/2;
fprintf('Training with %d features FIXED.\n', num_features);
mse_values = zeros(num_couples,delays);
plot_legend = cell(num_couples, 1);

k = 1;
for i = 1:num_features
    for j = i+1:num_features
        fprintf('Training with %s and %s.\n', char(data_day_labels(features(i))),
char(data_day_labels(features(j))));
        for d = 1:delays
            temp = 0;
            for it_num = 1:NUM_IT
                temp = temp + NTSDay(inputSeries(:,[features(i) features(j)]), targetSeries,
1:d, hiddenLayerSize);
            end;
            mse_values(k, d) = temp / NUM_IT;
        end
        fprintf('%0.2f%% Features: (%s,%s)\n',(k * 100)/num_couples,
char(data_day_labels(features(i))), char(data_day_labels(features(j))));
        plot_legend{k} = strcat(char(data_day_labels(features(i))),'-
',char(data_day_labels(features(j))));
        k = k + 1;
    end;
end;

%plot with different colours
color_map = hsv(num_couples);

hold on;
for i = 1:num_couples
    plot(1:delays,mse_values(i,:), 'color', color_map(i, :));
end
legend(plot_legend);
hold off;

end
```

13 – DayForecastNFeatures.m

Computes and plots the MSE of a forecast system in relation to different delays and to each subset of features on the given data set and target, with the specified set of fixed features and number of neurons. The script finally returns a plot of all the features subsets in relation to the different delay values

```
function [ mse_values ] = DayForecastNFeatures (dataset, target, features, delays, 
hiddenLayerSize, data_day_labels)
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

NUM_IT = 30;

%data partitioning
inputSeries = dataset(1:365,:);
targetSeries = target(1:365);

num_features = numel(features);
fprintf('Training with %d features FIXED.\n', num_features);
mse_values = zeros(num_features,delays);
plot_legend = cell(num_features, 1);

selectedFeatures = [9 10 12];

k = 1;
for i = 1:num_features
    fprintf('Training WEATHERSIT and TEMP with %s \n', char(data_day_labels(features(i))));
    for d = 1:delays
        temp = 0;
        for it_num = 1:NUM_IT
            temp = temp + NTSDay(inputSeries(:,[features(i) selectedFeatures]), targetSeries, 
1:d, hiddenLayerSize);
        end;
        mse_values(k, d) = temp / NUM_IT;
    end
    plot_legend{k} = strcat('weather-temp + ',char(data_day_labels(features(i))));
    k = k + 1;
end;


color_map = hsv(num_features);

hold on;
for i = 1:num_features
    plot(1:delays,mse_values(i,:), 'color', color_map(i, :));
end
legend(plot_legend);
hold off;

end
```

14 – DayForecastHiddenLayer.m

Computes and plots the MSE of a forecast system in relation to different neuron numbers. The delays are fixed as well as the features are.

The number of iterations is defined in the script and it stands for the number of times the MSE will be calculated for the same neuron number.

The overall MSE for each layer size is then calculated taking the averages of the multiple runs.

```
function [ mse_values ] = DayForecastHiddenLayer (dataset, target, delays,
hiddenLayerSizeLimit)
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

NUM_IT = 30;

%data partitioning
inputSeries = dataset(1:365,:);
targetSeries = target(1:365);

mse_values = zeros(hiddenLayerSizeLimit,1);

selectedFeatures = [7 9 10 12];

for i = 1:hiddenLayerSizeLimit
    temp = 0;
    for it_num = 1:NUM_IT
        temp = temp + NTSDay(inputSeries(:, selectedFeatures), targetSeries, 1:delays, i);
    end;
    mse_values(i) = temp / NUM_IT;
end

plot(mse_values);

end
```

15 – ntsDayClosed.m

Computes and plots the MSE of a forecasting system in relation to the given setup (delays, hidden layer size, features), data set and target.
The script finally returns a plot of the forecasted values for the 2nd year and the real measured values of the same year, all the MSEs in relation to the different forecasts computed in two ways: predictionErrors contains the forecasting errors computed on the first days of January 2012, results contains the forecasting MSEs computed on 30 dates sampled at random.

• • •

```
function [ predictionErrors, results, net ] = ntsDayClosed (inputSeriesTotal,
targetSeriesTotal, inputDelays, hiddenLayerSize, daysAhead)
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

inputSeries = tonndata(inputSeriesTotal(1:365,:),false,false);
targetSeries = tonndata(targetSeriesTotal(1:365),false,false);

inputSeriesClosed = tonndata(inputSeriesTotal(366:731,:),false,false);
targetSeriesClosed = tonndata(targetSeriesTotal(366:731),false,false);

feedbackDelays = inputDelays;
net = narxnet(inputDelays,feedbackDelays,hiddenLayerSize);

[inputs,inputStates,layerStates,targets] = preparets(net,inputSeries,{},targetSeries);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,inputs,targets,inputStates,layerStates);

% Closed Loop Network
% Use this network to do multi-step prediction.

netc = closeloop(net);
netc.name = [net.name ' - Closed Loop'];
%view(netc);
[xc,xic,aic,tc] = preparets(netc,inputSeriesClosed,{},targetSeriesClosed);
yc = netc(xc,xic,aic);

TS = size(tc,2);
outputClosed = cell2mat(yc);
outputTarget = cell2mat(tc);
plot(1:TS,outputTarget,'b',1:TS,outputClosed,'r')

% computing prediction squared errors for the first days of January 2012
 predictionErrors = zeros(numel(daysAhead),1);
j = 1;
for i = daysAhead
```

```
    predictionErrors(j,1) = (outputClosed(i) - outputTarget(i))^2;
    j = j + 1;
end

% Computing forecasts and errors by sampling 30 dates at random
delay = max(inputDelays);
maxForecast = max(daysAhead);
dates = datasample((366:(731-maxForecast))', 30, 1, 'Replace', false); % indeces

% Column vector with MSEs, one for each of the forecasts requested
 results = zeros(numel(daysAhead),1);
for j = 1:numel(daysAhead)
    sum = 0;
    for i = 1:30
        inputSeriesClosed = tonndata(inputSeriesTotal((dates(i)-
delay):(dates(i)+daysAhead(j)-1),:),false,false);
        targetSeriesClosed = tonndata(targetSeriesTotal((dates(i)-
delay):(dates(i)+daysAhead(j)-1)),false,false);
        [xc,xic,aic,tc] = preparets(netc,inputSeriesClosed,{},targetSeriesClosed);
        yc = netc(xc,xic,aic);
        outputClosed = cell2mat(yc);
        outputTarget = cell2mat(tc);

        % interested in the last element, the only interesting forecast
        sum = sum + (outputClosed(daysAhead(j)) - outputTarget(daysAhead(j)))^2;
    end
    results(j,1) = sum/30;
end
end
```

16 – dayForecastingClosed.m

Computes and plots the MSE of a forecasting system in relation to the given setup (delays, hidden layer size, features), data set and target.
The script finally returns a plot of all the MSEs in relation to the different forecasts.

```
function [ net, mse ] = dayForecastingClosed (dataset, target, selectedFeatures, delays,
hiddenLayerSize, numIt, daysAhead)
% @Authors: Alessio Villardita [villardita.alessio@gmail.com]
%           Sara Egidi [egidi.sara@gmail.com]

%data setting
inputSeries = dataset(:,selectedFeatures);
targetSeries = target(:);

numDaysAhead = numel(daysAhead);

% Here, when numIt > 1, a new NTS system is created and evaluated, so that
% a MSE can be computed in order to have a rough idea of the prediction
% errors coming from a specific setup of the system.
errors = zeros(numDaysAhead,numIt);
for i = 1:numIt
    [errors(:,i), results, net] = ntsDayClosed(inputSeries, targetSeries, 1:delays,
hiddenLayerSize, daysAhead);
end

mse = zeros(numDaysAhead,1);

for i = 1:numDaysAhead
    mse(i,1) = mean(errors(i, :));
end

figure;
plot(daysAhead, mse, 'r', daysAhead, results,'y');
end
```