Learning and Adaptivity - Project Report
Time Series Prediction (Stock Market Prediction)

Saugata Biswas, Ashok Meenakshi Sundaram

June 17, 2014

Contents

1 Introduction

The primary objective of this project is to learn to predict non linear time series. From the perspective of service or medical robotics which need to help people, the real life data like speech, activity, bio-informatics all require time series analysis to predict the future based on the past so that a control decision could be made. Also, in the other domains like meteorology, finance, marketing, web analysis etc. time series analysis is important. However there is no deterministic relation between the past and future. The variation of data are non linear. An effective way to learn a non linear series data is by computing moving average auto regressively. This project will focus on few machine learning approaches used for time series prediction and compare their results. The use case considered for this project is prediction of stock market based on its performance in the past.

2 Data Set and Features

Data set for the stock market prediction will be taken from Istanbul stock exchange which is available at `http://archive.ics.uci.edu/ml/datasets/ISTANBUL+STOCK+EXCHANGE` [1]. This data set contains indexes of Istanbul stock exchange in comparison with other international indexes e.g. SP, DAX, FTSE, NIKKEI etc. for a period of two years (2009-2011). This is a labeled data set with 538 entries. 70 percent of the data will be taken as training set, 15 percent for validation and the rest 15 percent will be used as test set.

Features i.e. other international indexes are already directly available in the data set but pre-processing of the data will be necessary. New additional features can be added by taking different statistical measures from the available measures. Python and Matlab has implementations of such statistical tools required to collect those features if required.

3 Methods

The following methods are being implemented for comparison and evaluation of time series prediction for Istanbul Stock Market data set.

3.1 Nonlinear Autoregressive Exogeneous Model (NARX)

In an autoregressive model, the output variable depends on its previous values. It falls under the category of Autoregressive Moving Average (ARMA) models. The model comprises of two parts namely the autoregressive part and the moving average part [2]. Nonlinear Autoregressive Exogeneous Model has exogeneous inputs. It relates the past values of the series and additionally the external series that influences it [3]. The implementation in matlab is given in App:.1. A neural network as shown in Fig:1 is implemented.

3.2 Linear Regression

Linear Regression fits a linear model i.e. linear predictor functions to the data set by adjusting a set of parameters in order to make the sum of the squared residuals of the model as small as

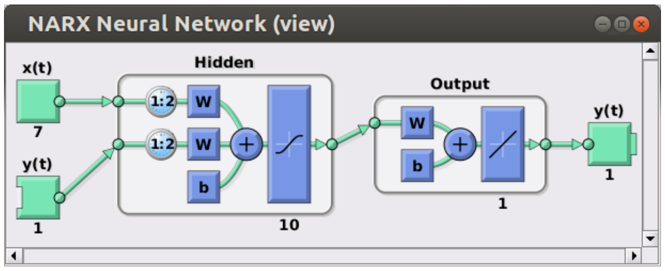

Figure 1: NARX Neural Network

possible [4]. The implementation in scikit is given in App:.2.

3.3 Support Vector Regression (SVR)

Support Vector Machines try to find a combination of samples to build a plane maximizing the margin between the two classes. They are primarily intended for classification. However, it can be extended to solve regression problems [5]. Nonlinear data are handled by kernels techniques by projecting the data in higher dimension. The model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction. The implementation in scikit is given in App:.2.

3.4 Ridge Regression

Ridge Regression is widely used for ill posed problems [6]. It overcomes some of the problems faced by ordinary least square minimization method i.e. linear regression for ill posed problems. A regularization term is included in the minimization process. The effect of regularization parameter varies with its scale. The implementation in scikit is given in App:.2.

4 Results

The following results were found after 10 runs of each model.

Nonlinear Autoregressive Exogeneous Model

The error historgram with 20 bins are show below in Fig:2 and Fig:3.

- Mean square error $3.22 * e^{-4}$
- R score $1.94 * e^{-1}$

Linear Regression

The prediction for test set is show below in Fig:4.

- Mean square error $4.01 * e^{-1}$
- R score $3.68 * e^{-1}$

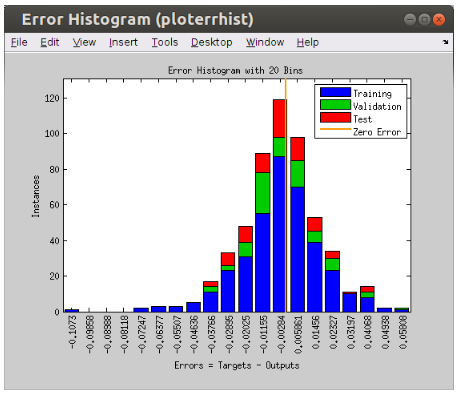

Figure 2: NARX Error Histogram

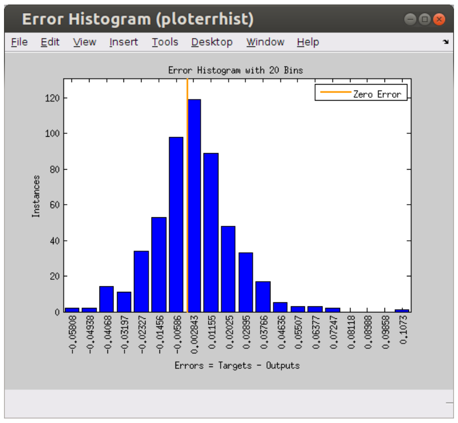

Figure 3: NARX Error Histogram - Test Set

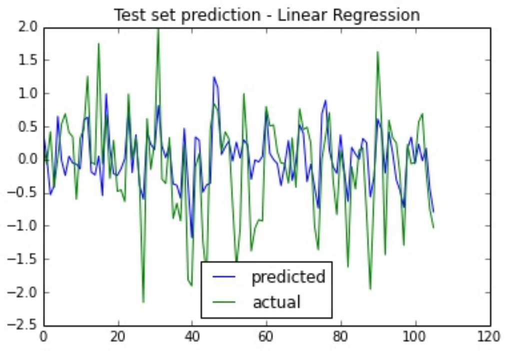

Figure 4: Linear Regression - Test Prediction

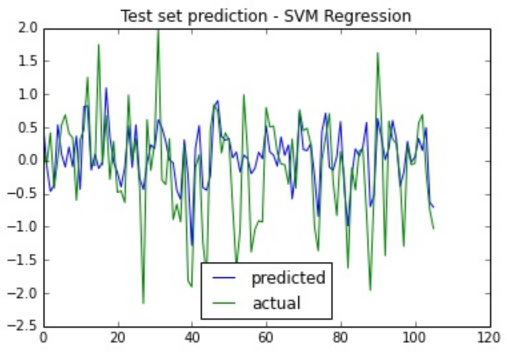

Figure 5: Support Vector Regression - Test Prediction

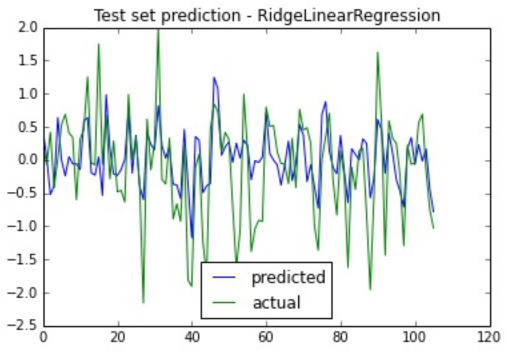

Figure 6: Ridge Regression - Test Prediction

Support Vector Regression

The prediction for test set is show below in Fig:5.

- Mean square error $4.42 * e^{-1}$
- R score $3.04 * e^{-1}$

Ridge Regression

The prediction for test set is show below in Fig:6.

- Mean square error $4.00 * e^{-1}$
- R score $3.70 * e^{-1}$

5 Conclusion

The performance of linear regression, support vector regression and ridge regression were are all found to be similar with almost similar R score and mean square error. However, the mean square error of nonlinear autoregressive exogeneous model was lower compared to the other methods. Therefore, we conclude that models belonging to the family of autoregressive moving average methods are well suited for nonlinear time series prediction.

Appendices

.1 Matlab Code - Nonlinear Autoregressive Moving Average Model

read_csv.m

```matlab
% Options
numCols = 10;
opts = {'Delimiter',','};

% Open file for reading
fid = fopen('../../data/istanbul_stock.csv');

% Read header line
headers = textscan(fid, repmat('%s',1,numCols), 1, opts{:});

% Read data
input = textscan(fid,'%s%f%f%f%f%f%f%f%f%f', opts{:});

% Close file
fclose(fid);

% Collect data

features ={};
for i=1:536
    temp = [];
    for j = 4:10
        temp = [temp ; input{1,j}(i,1)];
    end
    features = [features,temp];
end

ise ={};
for i=1:536
    temp = input{1,3}(i,1);
    ise = [ise,temp];
end

headers = [headers{:}];
%input = [input{:}];

clear temp fid opts numCols i j
```

narx.m

```matlab
% Solve an Autoregression Problem with External Input with a NARX Neural Network
% Script generated by NTSTOOL
% Created Sun Jun 15 21:11:07 CEST 2014
%
% This script assumes these variables are defined:
%
% features - input time series.
% ise - feedback time series.
```

```matlab

X = features;
T = ise;

% Create a Nonlinear Autoregressive Network with External Input
inputDelays = 1:2;
feedbackDelays = 1:2;
hiddenLayerSize = 10;
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.
[x,xi,ai,t] = preparets(net,X,{},T);

% 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,x,t,xi,ai);

% Test the Network
y = net(x,xi,ai);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotregression(t,y)
%figure, plotresponse(t,y)
%figure, ploterrcorr(e)
%figure, plotinerrcorr(x,e)

% Closed Loop Network
% Use this network to do multi-step prediction.
% The function CLOSELOOP replaces the feedback input with a direct
% connection from the outout layer.
netc = closeloop(net);
netc.name = [net.name '␣-␣Closed␣Loop'];
view(netc)
[xc,xic,aic,tc] = preparets(netc,X,{},T);
yc = netc(xc,xic,aic);
closedLoopPerformance = perform(netc,tc,yc)

% Step-Ahead Prediction Network
% For some applications it helps to get the prediction a timestep early.
% The original network returns predicted y(t+1) at the same time it is given y(t+1).
% For some applications such as decision making, it would help to have predicted
% y(t+1) once y(t) is available, but before the actual y(t+1) occurs.
% The network can be made to return its output a timestep early by removing one delay
% so that its minimal tap delay is now 0 instead of 1. The new network returns the
% same outputs as the original network, but outputs are shifted left one timestep.
nets = removedelay(net);
nets.name = [net.name '␣-␣Predict␣One␣Step␣Ahead'];
view(nets)
[xs,xis,ais,ts] = preparets(nets,X,{},T);
ys = nets(xs,xis,ais);
stepAheadPerformance = perform(nets,ts,ys)
```

.2 Scikit Code - Linear Regression, Support Vector Machine Regression, Ridge Regression

StockMarketPrediction.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 15 22:42:04 2014

@author: saugata, ashok
"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.svm import NuSVR
from sklearn.linear_model import Ridge


headers = []
ise_us = []
sp = []
dax = []
ftse = []
nikkei = []
bovespa = []
eu = []
em = []
features = []

###### reading data from file ##############
data_from_file = [i.strip().replace('␣', '').split(',') for i in open("../../data/istanbul_stock.
    csv").readlines()]

headers = data_from_file[0]

for i in range(1,537):
    ise_us = ise_us + [float(data_from_file[i][2])]
    sp = sp + [float(data_from_file[i][3])]
    dax = dax + [float(data_from_file[i][4])]
    ftse = ftse + [float(data_from_file[i][5])]
    nikkei = nikkei + [float(data_from_file[i][6])]
    bovespa = bovespa + [float(data_from_file[i][7])]
    eu = eu + [float(data_from_file[i][8])]
    em = em + [float(data_from_file[i][9])]

for i in range(0,536):
    features.append([sp[i],dax[i],ftse[i],nikkei[i],bovespa[i],eu[i],em[i]])

##################################################

features = np.array(features)
targets = np.array(ise_us).reshape(len(ise_us),1)

#########################################################
########## feedback of outputs ##########################
def incorporate_feedback(data,targets,fb=4):
    mod_targets = targets[fb:,:]
    prev_outputs = np.zeros((np.size(mod_targets,0), fb))

    for i in range(np.size(features, 0)-fb):
        feedback = targets[i:i+fb,]
        for j in range(fb):
            prev_outputs[i,j] = feedback[j,:]

    mod_features = np.hstack((data[fb:,:], prev_outputs))
    return mod_features, mod_targets
##########################################################

####### incorporating feedback ##########################
no_of_feedback = 10
print 'no␣of␣feedback␣=', no_of_feedback
features, targets = incorporate_feedback(features, targets, no_of_feedback)
###########################################################


########### normalizing ###################################
features = np.divide((features-np.mean(features,0)),np.std(features,0))
targets = np.divide((targets-np.mean(targets,0)),np.std(targets,0))

print features.shape, targets.shape

test_set_ratio = 0.8

train_features = features[:int(test_set_ratio*np.size(features,0)),:]
test_features = features[int(test_set_ratio*np.size(features,0)):,:]

train_targets = targets[:int(test_set_ratio*np.size(features,0)),:]
test_targets = targets[int(test_set_ratio*np.size(features,0)):,:]


########### training ################
clf = linear_model.LinearRegression()
clf.fit(train_features, train_targets)

print 'accuracy␣on␣training␣set', clf.score(train_features,train_targets)
print 'accuracy␣on␣test␣set', clf.score(test_features,test_targets)

print 'mse␣on␣training␣set␣=', np.sum((train_targets-clf.predict(train_features))**2)/np.size(
    train_features,0)
print 'mse␣on␣test␣set␣=', np.sum((test_targets-clf.predict(test_features))**2)/np.size(
    test_features,0)

```

```
plt.figure(0)
plt.plot(clf.predict(train_features), label = 'predicted')
plt.plot(train_targets, label = 'actual')
plt.legend(loc = 8)
plt.title('Train set prediction - Linear Regression')
plt.savefig('Train_set_prediction-LinearRegression.jpg', bbox_inches='tight')

plt.figure(1)
plt.plot(clf.predict(test_features), label = 'predicted')
plt.plot(test_targets, label = 'actual')
plt.legend(loc = 8)
plt.title('Test set prediction - Linear Regression')
plt.savefig('Test_set_prediction-LinearRegression.jpg', bbox_inches='tight')
################################################################
#### Support Vector Regression #################################
################################################################
print '\nSupport Vector Regression\n'

targets = targets[:,0]
test_set_ratio = 0.8

train_features = features[:int(test_set_ratio*np.size(features,0)),:]
test_features = features[int(test_set_ratio*np.size(features,0)):,:]

train_targets = targets[:int(test_set_ratio*np.size(features,0))]
test_targets = targets[int(test_set_ratio*np.size(features,0)):]
########### training ################
clf = NuSVR(C=1.0, nu=0.1)
clf.fit(train_features, train_targets)

print 'accuracy on training set', clf.score(train_features,train_targets)
print 'accuracy on test set', clf.score(test_features,test_targets)

print 'mse on training set =', np.sum((train_targets-clf.predict(train_features))**2)/np.size(
    train_features,0)
print 'mse on test set =', np.sum((test_targets-clf.predict(test_features))**2)/np.size(
    test_features,0)

plt.figure(2)
plt.plot(clf.predict(train_features), label = 'predicted')
plt.plot(train_targets, label = 'actual')
plt.legend(loc = 8)
plt.title('Train set prediction - SVM Regression')
plt.savefig('Train_set_prediction-SVMRegression.jpg', bbox_inches='tight')

plt.figure(3)
plt.plot(clf.predict(test_features), label = 'predicted')
plt.plot(test_targets, label = 'actual')
plt.legend(loc = 8)
plt.title('Test set prediction - SVM Regression')
plt.savefig('Test_set_prediction-SVMRegression.jpg', bbox_inches='tight')

################################################################
#### Support Vector Regression #################################
################################################################
print '\nRidge for Regression\n'

########### training ################
clf = Ridge(alpha=1.0)
```

```
clf.fit(train_features, train_targets)

print 'accuracy␣on␣training␣set', clf.score(train_features,train_targets)
print 'accuracy␣on␣test␣set', clf.score(test_features,test_targets)

print 'mse␣on␣training␣set␣=', np.sum((train_targets-clf.predict(train_features))**2)/np.size(
    train_features,0)
print 'mse␣on␣test␣set␣=', np.sum((test_targets-clf.predict(test_features))**2)/np.size(
    test_features,0)

plt.figure(4)
plt.plot(clf.predict(train_features), label = 'predicted')
plt.plot(train_targets, label = 'actual')
plt.legend(loc = 8)
plt.title('Train␣set␣prediction␣-␣RidgeLinearRegression')
plt.savefig('Train_set_prediction-RidgeLinearRegression.jpg', bbox_inches='tight')

plt.figure(5)
plt.plot(clf.predict(test_features), label = 'predicted')
plt.plot(test_targets, label = 'actual')
plt.legend(loc = 8)
plt.title('Test␣set␣prediction␣-␣RidgeLinearRegression')
plt.savefig('Test_set_prediction-RidgeLinearRegression.jpg', bbox_inches='tight')
plt.show()
```

References

[1] Akbilgic, Oguz, Hamparsum Bozdogan, and M. Erdal Balaban. ”A novel Hybrid RBF Neural Networks model as a forecaster.” Statistics and Computing (2013): 1-11.

[2] Whittle, P. (1951). Hypothesis Testing in Time Series Analysis. Almquist and Wicksell. Whittle, P. (1963). Prediction and Regulation. English Universities Press. ISBN 0-8166-1147-5.

[3] S. A. Billings. ”Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains, Wiley, ISBN 978-1-1199-4359-4, 2013.

[4] Arnold, Tom, and Jonathan Godbey. ”Introducing Linear Regression: An Example Using Basketball Statistics.” JOURNAL OF ECONOMICS AND FINANCE EDUCATION 11.2 (2012).

[5] Drucker, Harris, et al. ”Support vector regression machines.” Advances in neural information processing systems 9 (1997): 155-161.

[6] Tikhonov, A. N.; V. Y. Arsenin (1977). Solution of Ill-posed Problems. Washington: Winston & Sons. ISBN 0-470-99124-0.