Weekly Report of Research Work

WR-ABS-TEMP-2015A-No.009

汤吉(Ji TANG)

Number: WR-ABS-TEMP-2015A, E-mail: tangji08@hotmail.com

Date: 18/1/2016 - 24/1/2016

January 24, 2016

Contents

1 Work

1. Improving all my codes.
2. Programming the programs to calculate the moving average lines for different periods.
3. Writing the functions to calculate the Bollinger Bands line and plot it on the Price chart.
4. Designing a simple GUI calling all the functions I have written.
5. Choosing a paper and translated a little bit of it.
6. Learning the Neural Network.
7. Designing a complete program including setting the parameters, training the data, ploting and GUI for the neural network example.

2 Normalize all my codes

This week, the first thing I have done is reconstructing all my function in order to have better use of them.

After this work, the codes are like this:

```
# -*- coding:utf-8 -*-
__author__ = 'Jojo'

import ...
version = sys.version.find("Red Hat")
if version == -1:
    import matplotlib.pyplot as plt

datapath = os.getcwd() + "Data/"

# This function will read the file and translate it to data
# *** the lists of Date, Time, Price ***
def file2data(filename):...


# To calculate the numbers of days between two dates
# *** an int ***
```

```
def datediff(beginDate, endDate):...


#This function will set the first data time as the "0" time and format the
    following data based on this time
# *** a float array ***
def time2number(Date, Time):...


if version == -1:
    # Plot the Data
    # *** No return ***
    def plotforexdata(Date, Time, Price, name):...


    # Plot the txt by calling the plotforexdata function
    # *** No return ***
    def plotforextxt(filename):...


# To transform the UTC txt to data
# *** the lists of Time, UTC, Price ***
def UTCfile2data(filename):...


# To transform the UTC txt to data in the giving periode
# *** the lists of Time, UTC, Price ***
def input2data(forextype, startdate, enddate):...


if version == -1:
    # To plot the UTC data with a name
    # *** No return ***
    def plotUTCforexdata(Time, UTC, Price, name):...


    # To plot the chart in the giving time for the giving type
    # *** No return ***
    def plotpricechart(type, starttime, endtime):...
```

3 MVA (Moving Average)

The moving average line is the average value of the giving period line.

```python
# code: UTF-8
__author__ = 'Jojo'
from FkNN import *
from changetime import *


# Calculate the moving average line of each period
# *** a matrix of Price and a list of Period ***
def MVA(UTC, Price, Period):
  ...
    return MVAPrice, period

if version == -1:
    # Plot the MVA line
    # *** No return ***
    def plotMVAdata(Time, UTC, Price, MVAPrice, period, name):
        ...
        plt.show()


    # To plot the MVA chart in the giving time for the giving type
    # *** No return ***
    def plotMVA(type, startime, endtime, Period):
        ...
        plotMVAdata(Time, UTC, Price, MVAPrice, period, type[-6:])
```

4 Bollinger Bands

4.1 Introduction

Bollinger Bands is a technical analysis tool invented by John Bollinger in the 1980s as well as a term trademarked by him in 2011.[1] Having evolved from the concept of trading bands, Bollinger Bands and the related indicators

Bollinger Bands consist of: an N-period moving average (MA) an upper band at K times an N-period standard deviation above the moving average (MA + Kσ) a lower band at K times an N-period standard deviation below the moving average ($MA - K\sigma$)

Typical values for N and K are 20 and 2, respectively. The default choice for the average is a simple moving average, but other types of averages can be employed as needed. Exponential moving averages is a common second choice. Usually the same period is used for both the middle band and the calculation of standard deviation.

4.2 Interpretation

The use of Bollinger Bands varies widely among traders. Some traders buy when price touches the lower Bollinger Band and exit when price touches the moving average in the center of the bands. Other traders buy when price breaks above the upper Bollinger Band or sell when price falls below the lower Bollinger Band. Moreover, the use of Bollinger Bands is not confined to stock traders; options traders, most notably implied volatility traders, often sell options when Bollinger Bands are historically far apart or buy options when the Bollinger Bands are historically close together, in both instances, expecting volatility to revert towards the average historical volatility level for the stock.

When the bands lie close together, a period of low volatility is indicated. Conversely, as the bands expand, an increase in price action/market volatility is indicated. When the bands have only a slight slope and track approximately parallel for an extended time, the price will generally be found to oscillate between the bands as though in a channel.

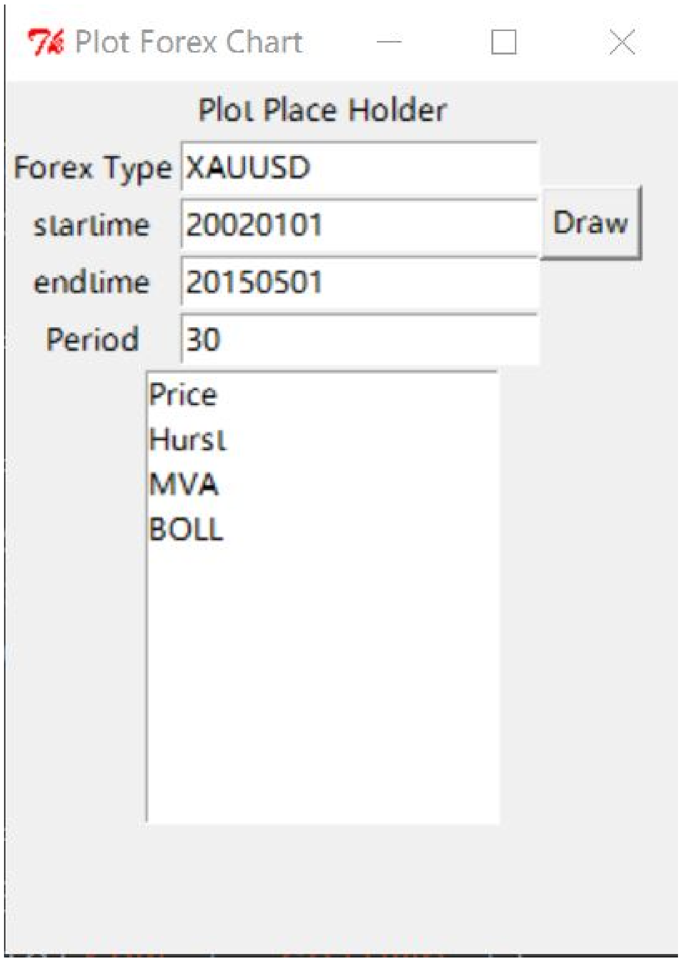
Traders are often inclined to use Bollinger Bands with other indicators to confirm price action. In particular, the use of oscillator-like Bollinger Bands will often be coupled with a non-oscillator indicator-like chart patterns or a trendline. If these indicators confirm the recommendation of the Bollinger Bands, the trader will have greater conviction that the bands are predicting correct price action in relation to market volatility.[?]

4.3 Code

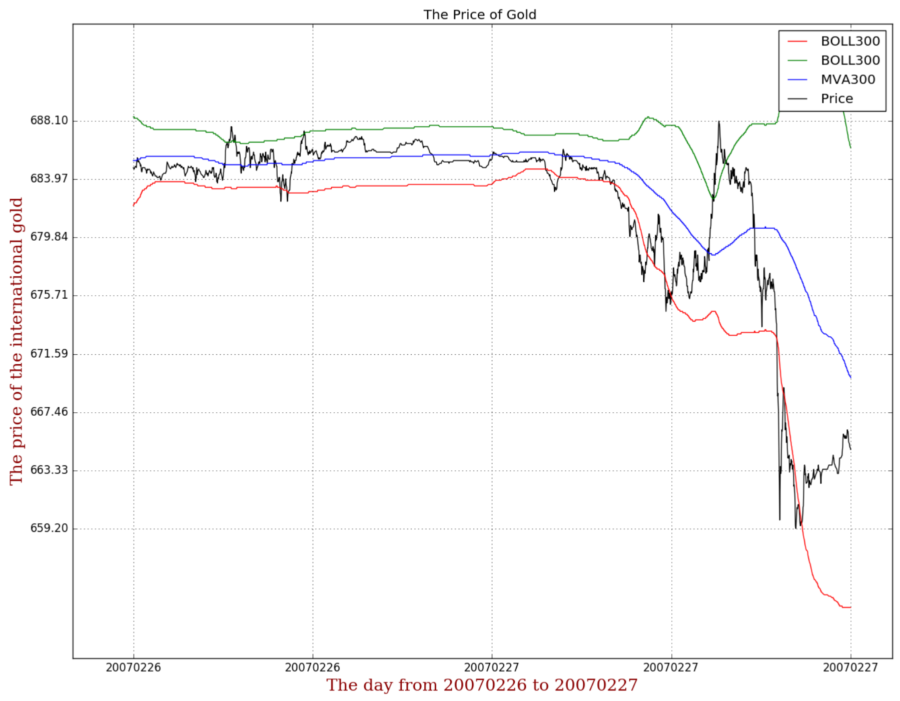
```
# Calculate the moving average line of each period
# *** a matrix of MVA, a 3D-matrix of BOLL and a list of Period ***
def BOLL(UTC, Price, Period):...


# To calculate the moving average value and BOLL values and store it to the "
    datapath"
# *** No return ***
def saveBOLL(type, startime, endtime, Period):...
```

```


    # Plot the BOLL line
    # *** No return ***
    def plotBOLLdata(Time, UTC, Price, MVAPrice, BOLLline, period, name):...


    # To plot the BOLL chart in the giving time for the giving type
    # *** No return ***
    def plotBOLL(type, starttime, endtime, Period):...
```

5 GUI for ploting the chart



Forex Type XAUUSD
startime 20020101
endtime 20150501
Period 30



6 The paper I select

I have select a paper named ” Hurst exponent and Financial Market Predictability” , and begin to translate it.

HURST EXPONENT AND FINANCIAL MARKET PREDICTABILITY

Bo Qian Khaled Rasheed
Department of Computer Science
University of Georgia
Athens, GA 30601
USA
[qian, khaled]@cs.uga.edu

ABSTRACT
The Hurst exponent (H) is a statistical measure used to classify time series. H=0.5 indicates a random series while H>0.5 indicates a trend reinforcing series. The larger the H value is, the stronger trend. In this paper we investigate the use of the Hurst exponent to classify series of financial data representing different periods of time. Experiments with backpropagation Neural Networks show that series with large Hurst exponent can be predicted more accurately than those series with H value close to 0.50. Thus Hurst exponent provides a measure for predictability.

KEY WORDS
Hurst exponent, time series analysis, neural networks, Monte Carlo simulation, forecasting

1. Introduction

The Hurst exponent, proposed by H. E. Hurst [1] for use in fractal analysis [2],[3], has been applied to many research fields. It has recently become popular in the finance community [4],[5],[6] largely due to Peters' work [7],[8]. The Hurst exponent provides a measure for long-term memory and fractality of a time series. Since it is robust with few assumptions about underlying system, it has broad applicability for time series analysis. The values of the Hurst exponent range between 0 and 1. Based on the Hurst exponent value H, a time series can be classified into three categories. (1) H=0.5 indicates a random series. (2) 0<H<0.5 indicates an anti-persistent series. (3) 0.5<H<1 indicates a persistent series. An anti-persistent series has a characteristic of "mean-reverting", which means an up value is more likely followed by a down value, and vice versa. The strength of "mean-reverting" increases as H approaches 0.0. A persistent series is trend reinforcing, which means the direction (up or down compared to the last value) of the next value is more likely the same as current value. The strength of trend increases as H approaches 1.0. Most economic and financial time series are persistent with H>0.5.

In time series forecasting, the first question we want to answer is whether the time series under study is predictable. If the time series is random, all methods are expected to fail. We want to identify and study those time series having at least some degree of predictability. We know that a time series with a large Hurst exponent has strong trend, thus it's natural to believe that such time series are more predictable than those having a Hurst exponent close to 0.5. In this paper we use neural networks to test this hypothesis.

Neural networks are nonparametric universal function approximators [9] that can learn from data without assumptions. Neural network forecasting models have been widely used in financial time series analysis during the last decade [10],[11],[12]. As universal function approximators, neural networks can be used for surrogate predictability. Under the same conditions, a time series with a smaller forecasting error than another is said to be more predictable. We study the Dow-Jones index daily return from Jan. 2, 1930 to May. 14, 2004 and calculate the Hurst exponent of each period of 1024 trading days. We select 30 periods with large Hurst exponents and 30 periods with Hurst exponents close to random series, and then we use these data to train neural networks. We compare forecasting errors for these two groups and find that the errors are significantly different. This research is done using Matlab. All Matlab programs generating result for this paper can be downloaded from www.arches.uga.edu/~qianbo/research.

The remainder of the paper is organised as follows: Section 2 describes the Hurst exponent in detail. Section 3 then describes the monte carlo simulation process we used to generate data with similar structure to the financial series of interest to us. Section 4 describes a scramble test that we conducted to help verify that there is structure in the series due to the order of samples. Section 5 describes neural networks and their use to verify that sequences with larger values of the Hurst exponent can be more accurately learned and predicted than those with lower Hurst exponent values. Finally, the paper is concluded in section 6.

7 Artificial neural network

7.1 Background

Examinations of humans' central nervous systems inspired the concept of artificial neural networks. In an artificial neural network, simple artificial nodes, known as "neurons", "neurodes", "processing elements" or "units", are connected together to form a network which mimics a biological neural network.

There is no single formal definition of what an artificial neural network is. However, a class of statistical models may commonly be called "neural" if it possesses the following characteristics: 1.contains sets of adaptive weights, i.e. numerical parameters that are tuned by a learning algorithm, and 2.capability of approximating non-linear functions of their inputs.

The adaptive weights can be thought of as connection strengths between neurons, which are activated during training and prediction.

Neural networks are similar to biological neural networks in the performing of functions collectively and in parallel by the units, rather than there being a clear delineation of subtasks to which individual units are assigned. The term "neural network" usually refers to models employed in statistics, cognitive psychology and artificial intelligence. Neural network models which command the central nervous system and the rest of the brain are part of theoretical neuroscience and computational neuroscience.[?]

In modern software implementations of artificial neural networks, the approach inspired by biology has been largely abandoned for a more practical approach based on statistics and signal processing. In some of these systems, neural networks or parts of neural networks (like artificial neurons) form components in larger systems that combine both adaptive and non-adaptive elements. While the more general approach of such systems is more suitable for real-world problem solving, it has little to do with the traditional, artificial intelligence connectionist models. What they do have in common, however, is the principle of non-linear, distributed, parallel and local processing and adaptation. Historically, the use of neural network models marked a directional shift in the late eighties from high-level (symbolic) artificial intelligence, characterized by expert systems with knowledge embodied in if-then rules, to low-level (sub-symbolic) machine learning, characterized by knowledge embodied in the parameters of a dynamical system.

7.2 Improvement

Computational devices have been created in CMOS, for both biophysical simulation and neuromorphic computing. More recent efforts show promise for creating nanodevices for very large scale principal components analyses and convolution. If successful, these efforts could usher in a new era of neural computing that is a step beyond digital computing, because it depends on learning rather than programming and because it is fundamentally analog rather than digital even though the first instantiations may in fact be with CMOS digital devices.

Between 2009 and 2012, the recurrent neural networks and deep feedforward neural networks developed in the research group of Jürgen Schmidhuber at the Swiss AI Lab IDSIA have won eight international competitions in pattern recognition and machine learning. For example, the bi-directional and multi-dimensional long short term memory (LSTM) of Alex Graves et al. won three competitions in connected handwriting recognition at the 2009 International Conference on Document Analysis and Recognition (ICDAR), without any prior knowledge about the three different languages to be learned.

Fast GPU-based implementations of this approach by Dan Ciresan and colleagues at IDSIA have won several pattern recognition contests, including the IJCNN 2011 Traffic Sign Recognition Competition, the ISBI 2012 Segmentation of Neuronal Structures in Electron Microscopy Stacks challenge, and others. Their neural networks also were the first artificial pattern recognizers to achieve human-competitive or even superhuman performance on important benchmarks such as traffic sign recognition (IJCNN 2012), or the MNIST handwritten digits problem of Yann LeCun at NYU.

Deep, highly nonlinear neural architectures similar to the 1980 neocognitron by Kunihiko Fukushima and the ” standard architecture of vision” , inspired by the simple and complex cells identified by David H. Hubel and Torsten Wiesel in the primary visual cortex, can also be pre-trained by unsupervised methods of Geoff Hinton’ s lab at University of Toronto. A team from this lab won a 2012 contest sponsored by Merck to design software to help find molecules that might lead to new drugs.[?]

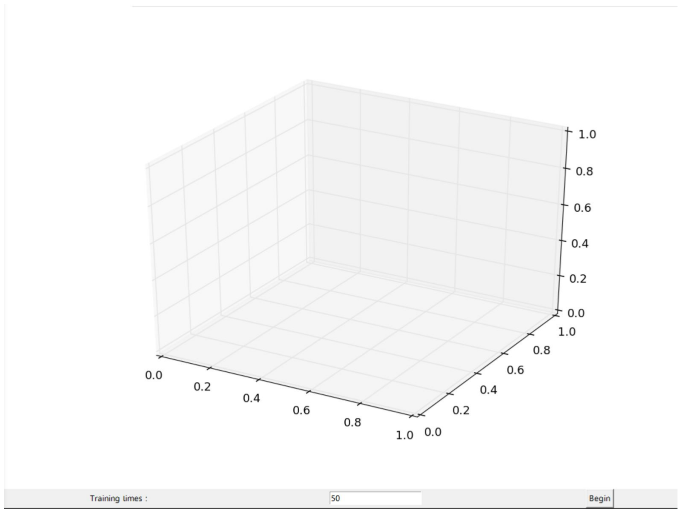
7.3 Full Code

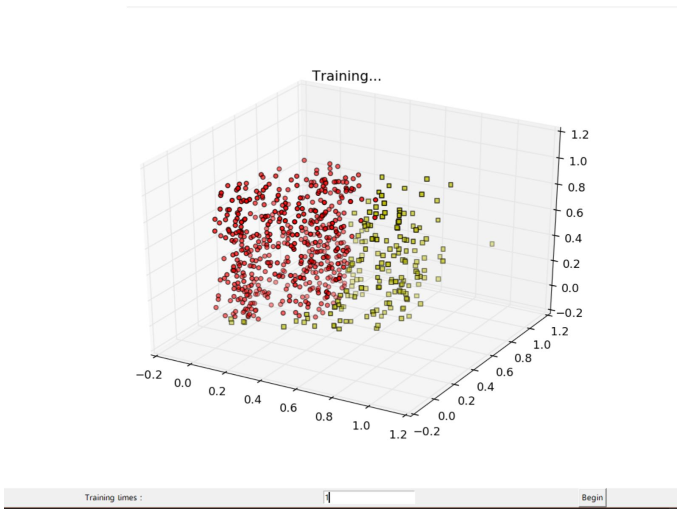
I have selected a python tool named ” pybrain” to realize the artificial neural network.

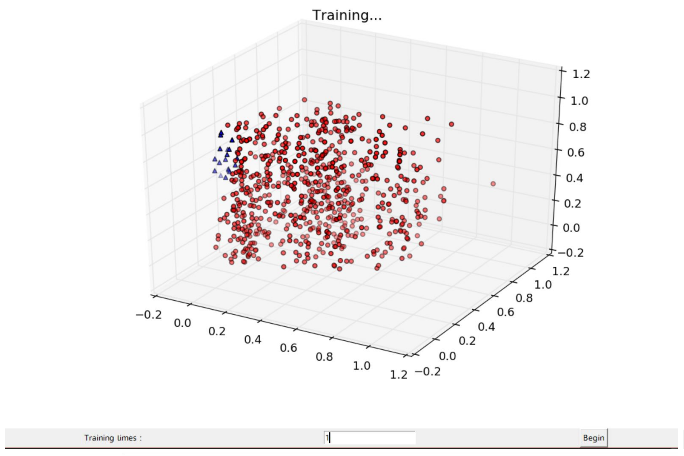
```
#!/usr/bin/env python
#coding:utf-8
from pybrain.datasets import ClassificationDataSet
from pybrain.utilities import percentError
from pybrain.datasets import SupervisedDataSet
from pybrain.tools.shortcuts import buildNetwork
from pybrain.supervised.trainers import BackpropTrainer
from pybrain.structure.modules import SoftmaxLayer
from numpy import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import os
```

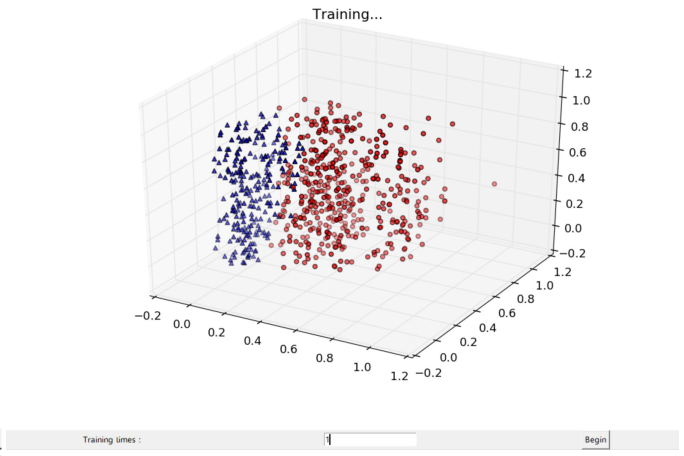
```
from Tkinter import *
import matplotlib
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

datapath = os.getcwd()[0:-6] + "/kNN/"
#-----------------------------------------------------------------------
def file2matrix(filename):
    fr = open(filename) #Load the data
    NumbersOfLines = len(fr.readlines()) #Get the number of the lines
    returnMatrix = zeros((NumbersOfLines, 3)) #Create Numpy matrix to return
    classLabelVector = []
    fr = open(filename) #Reload the data
    index = 0
    for line in fr.readlines():
        lines = line.strip() #Strip the beginning and the end blank
        listFromLine = lines.split() #Separate each line by '\t'
        returnMatrix[index, :] = listFromLine[0:3] #Get the 3 values of each line
        classLabelVector.append(float(listFromLine[-1])) #Get the evaluate of each
            person
        index += 1
    return returnMatrix, classLabelVector
def autoNorm(dataSet):
    minValues = dataSet.min(0) #The min value
    maxValues = dataSet.max(0) #The max value
    ranges = maxValues - minValues
   # normDataset = zeros(shape(dataSet))
   m = dataSet.shape[0] #The number of lines
    normDataset = dataSet - tile(minValues, (m, 1))
    normDataset = normDataset/tile(ranges, (m, 1)) #Calculate the values
        normalized
    return normDataset, ranges, minValues

def drawPic():
     try:
          sampleCount = int(inputEntry.get())
     except:
          sampleCount = 50
          print 'Enter an integer.'
          inputEntry.delete(0, END)
          inputEntry.insert(0, '50')
     # Need column vectors in dataset, not arrays
    for i in range(sampleCount):
        trainer.trainEpochs(1)
        trnresult = percentError(trainer.testOnClassData(), trndata['class'])
        tstresult = percentError(trainer.testOnClassData(dataset=tstdata),
            tstdata['class'])
        if i % 20 == 0:
            t.delete(1.0, END)
        t.insert(END, "epoch:" + str(trainer.totalepochs) + "  train error:" +
            str(round(trnresult, 2)) \
                 + "%  test error:" + str(round(tstresult, 2)) + "%\n")
        # Clear the Figure
        drawPic.f.clf()
        drawPic.a = drawPic.f.add_subplot(111, projection='3d')
        drawPic.a.set_title('Training...')
        for a, c, m in [(0, 'r', 'o'), (1, 'b', '^'), (2, 'y', 's')]:
            out = fnn.activateOnDataset(alldata)
            out = out.argmax(axis=1)
            here = (out == a)
            drawPic.a.scatter(alldata['input'][here, 0], alldata['input'][here,
                1], alldata['input'][here, 2], c=c, marker=m)
        drawPic.canvas.show()

if __name__ == '__main__':
    matplotlib.use('TkAgg')
    root = Tk()
    root.title('Neural network training')
    new = Tk()
    new.title('Note the errors')
    new.geometry('350x800')
    t = Text(new)
    t.grid(row=1, column=1, rowspan=50, columnspan=50)
    t.pack()
    # Putting a figure on GUI
    drawPic.f = Figure(figsize=(10, 7), dpi=108, facecolor="white")
    ax = drawPic.f.add_subplot(111, projection='3d')
    drawPic.canvas = FigureCanvasTkAgg(drawPic.f, master=root)
    drawPic.canvas.show()
    drawPic.canvas.get_tk_widget().grid(row=0, columnspan=3)
    # Setting Labels and Text
    Label(root, text='Training : times').grid(row=1, column=0)
    inputEntry = Entry(root)
    inputEntry.grid(row=1, column=1)
    inputEntry.insert(0, '50')
    Button(root, text='Begin', command=drawPic).grid(row=1, column=2, columnspan
        =3)

    alldata = ClassificationDataSet(3, 1, nb_classes=3)
    datingDataMat, datingLabels = file2matrix(datapath +'datingTestSet.txt')
    normMat, ranges, minValues = autoNorm(datingDataMat)
    for i in range(len(normMat)):
        alldata.addSample(normMat[i], [datingLabels[i] - 1])
    tstdata, trndata = alldata.splitWithProportion(0.25)
    trndata._convertToOneOfMany( )
    tstdata._convertToOneOfMany( )
    alldata._convertToOneOfMany( )
    t.insert(END, "Number of training patterns: " + str(len(trndata)) + "\n")
    t.insert(END, "Input and output dimensions: " + str(trndata.indim) + str(
        trndata.outdim) + "\n")
    fnn = buildNetwork(trndata.indim, 100, 5, trndata.outdim, outclass=
        SoftmaxLayer)
    fnn.activate([3, 100, 1])
    trainer = BackpropTrainer(fnn, dataset=trndata, momentum=0.01, verbose=True,
        weightdecay=0.0001)
    # Begin the loop
    root.mainloop()
```

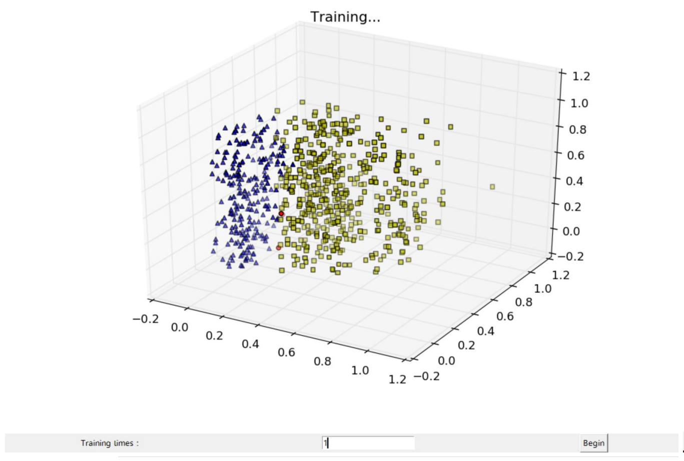
7.4 Test

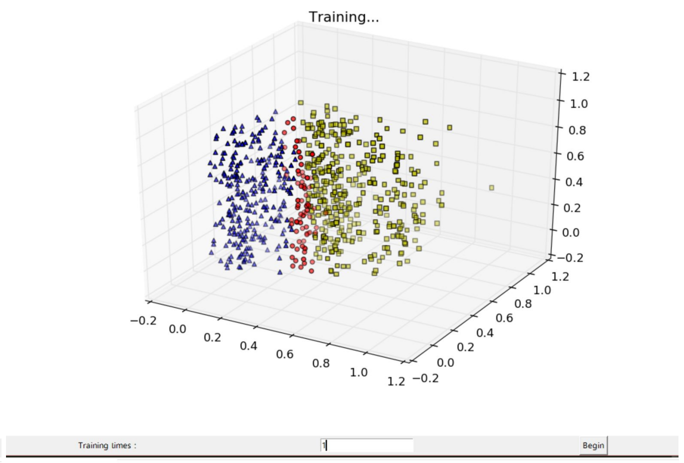


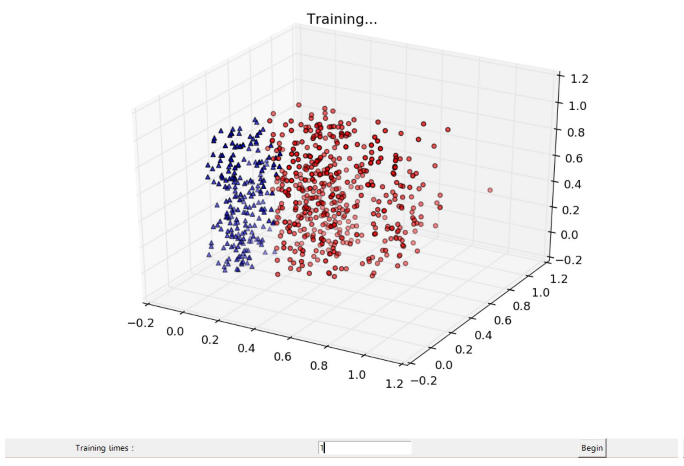


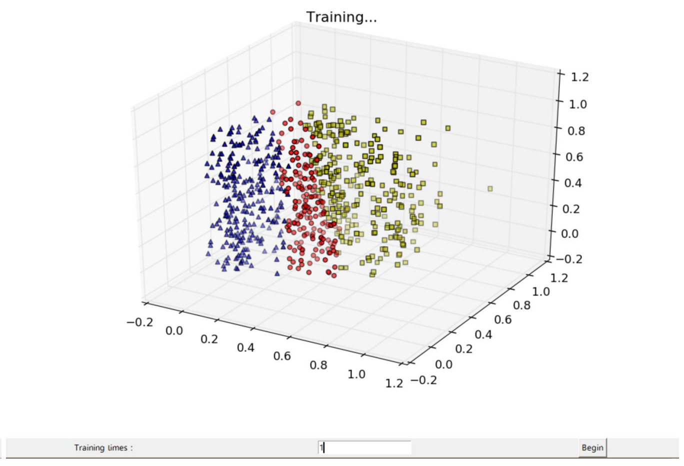

Training...
Training times :
1
Begin

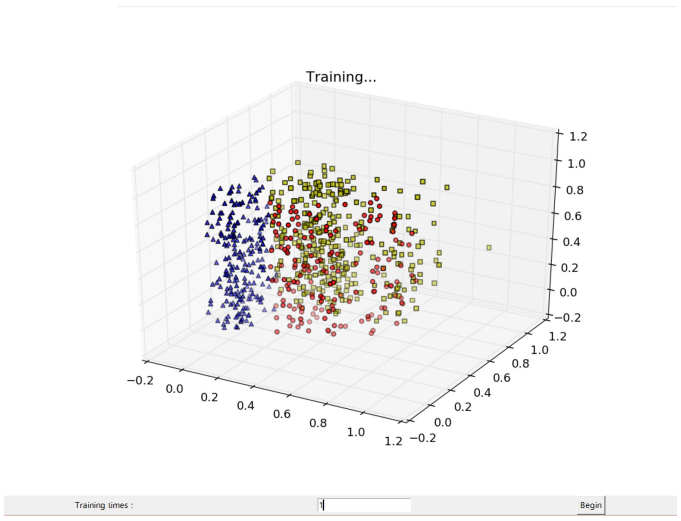

Training...
Training times :
1
Begin

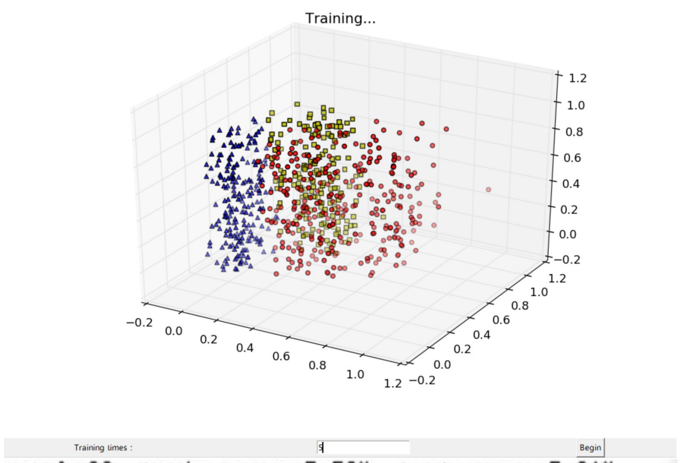

Training...
Training times :
1
Begin

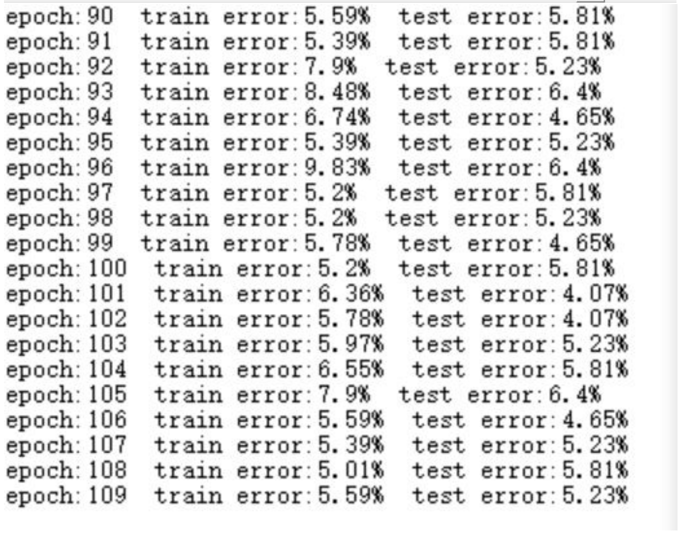

Training...
Training times :
1
Begin


Training...
Training times :
1
Begin

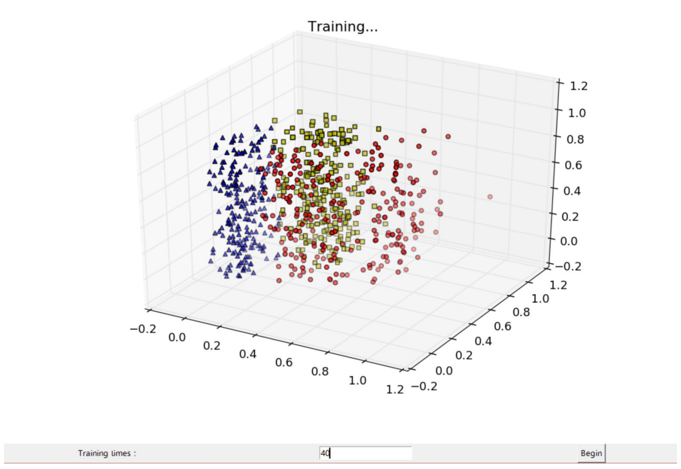

Training...
Training times :
1
Begin







```
epoch: 90  train error: 5.59%  test error: 5.81%
epoch: 91  train error: 5.39%  test error: 5.81%
epoch: 92  train error: 7.9%  test error: 5.23%
epoch: 93  train error: 8.48%  test error: 6.4%
epoch: 94  train error: 6.74%  test error: 4.65%
epoch: 95  train error: 5.39%  test error: 5.23%
epoch: 96  train error: 9.83%  test error: 6.4%
epoch: 97  train error: 5.2%  test error: 5.81%
epoch: 98  train error: 5.2%  test error: 5.23%
epoch: 99  train error: 5.78%  test error: 4.65%
epoch: 100  train error: 5.2%  test error: 5.81%
epoch: 101  train error: 6.36%  test error: 4.07%
epoch: 102  train error: 5.78%  test error: 4.07%
epoch: 103  train error: 5.97%  test error: 5.23%
epoch: 104  train error: 6.55%  test error: 5.81%
epoch: 105  train error: 7.9%  test error: 6.4%
epoch: 106  train error: 5.59%  test error: 4.65%
epoch: 107  train error: 5.39%  test error: 5.23%
epoch: 108  train error: 5.01%  test error: 5.81%
epoch: 109  train error: 5.59%  test error: 5.23%
```



It works really good. The dataset is the same with kNN method, and now we can see the procession of classifying directly.