Machine Learning
Exercise 2

Marc Toussaint
Machine Learning & Robotics lab, U Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany

April 14, 2016

1 Getting Started with Ridge Regression

In the appendix you find starting-point implementations of basic linear regression for Python, C++, and Matlab. These include also the plotting of the data and model. Have a look at them, choose a language and understand the code in detail.

On the course webpage there are two simple data sets `dataLinReg2D.txt` and `dataQuadReg2D.txt`. Each line contains a data entry (x, y) with $x \in \mathbb{R}^2$ and $y \in \mathbb{R}$; the last entry in a line refers to y.

a) The examples demonstrate plain linear regression for `dataLinReg2D.txt`. Extend them to include a regularization parameter λ. Report the squared error on the full data set when trained on the full data set.

b) Do the same for `dataQuadReg2D.txt` while first computing quadratic features.

c) Implement cross-validation (slide 02:18) to evaluate the *prediction error* of the quadratic model for a third, noisy data set `dataQuadReg2D_noisy.txt`. Report 1) the squared error when training on all data (=*training error*), and 2) the mean squared error $\widehat{\ell}$ from cross-validation.

Repeat this for different Ridge regularization parameters λ. (Ideally, generate a nice bar plot of the generalization error, including deviation, for various λ.)

Python (by Stefan Otte)

```
#!/usr/bin/env python
# encoding: utf-8
"""
This is a mini demo of how to use numpy arrays and plot data.
NOTE: the operators + - * / are element wise operation. If you want
matrix multiplication use ‘‘dot‘‘ or ‘‘mdot‘‘!
"""
import numpy as np
from numpy import dot
from numpy.linalg import inv

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D  # 3D plotting


###############################################################################
# Helper functions
def mdot(*args):
    """Multi argument dot function. http://wiki.scipy.org/Cookbook/MultiDot"""
    return reduce(np.dot, args)
```

```
def prepend_one(X):
    """prepend a one vector to X."""
    return np.column_stack([np.ones(X.shape[0]), X])


def grid2d(start, end, num=50):
    """Create an 2D array where each row is a 2D coordinate.

    np.meshgrid is pretty annoying!
    """
    dom = np.linspace(start, end, num)
    X0, X1 = np.meshgrid(dom, dom)
    return np.column_stack([X0.flatten(), X1.flatten()])


###############################################################################
# load the data
data = np.loadtxt("dataLinReg2D.txt")
print "data.shape:", data.shape
np.savetxt("tmp.txt", data)  # save data if you want to
# split into features and labels
X, y = data[:, :2], data[:, 2]
print "X.shape:", X.shape
print "y.shape:", y.shape

# 3D plotting
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')  # the projection arg is important!
ax.scatter(X[:, 0], X[:, 1], y, color="red")
ax.set_title("raw data")

plt.draw()  # show, use plt.show() for blocking

# prep for linear reg.
X = prepend_one(X)
print "X.shape:", X.shape

# Fit model/compute optimal parameters beta
beta_ = mdot(inv(dot(X.T, X)), X.T, y)
print "Optimal beta:", beta_

# prep for prediction
X_grid = prepend_one(grid2d(-3, 3, num=30))
print "X_grid.shape:", X_grid.shape
# Predict with trained model
y_grid = mdot(X_grid, beta_)
print "Y_grid.shape", y_grid.shape

# vis the result
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')  # the projection part is important
ax.scatter(X_grid[:, 1], X_grid[:, 2], y_grid)  # don't use the 1 infront
ax.scatter(X[:, 1], X[:, 2], y, color="red")  # also show the real data
ax.set_title("predicted data")
plt.show()
```

C++

(by Marc Toussaint)

```
//g++ -I../../../src -L../../../lib -fPIC -std=c++0x main.cpp -lCore

#include <Core/array.h>

//===========================================================================

void gettingStarted() {
  //load the data
  arr D = FILE("../01-linearModels/dataLinReg2D.txt");

  //plot it
  FILE("z.1") <<D;
  gnuplot("splot 'z.1' us 1:2:3 w p", true);

  //decompose in input and output
  uint n = D.d0; //number of data points
  arr Y = D.sub(0,-1,-1,-1).reshape(n);        //pick last column
  arr X = catCol(ones(n,1), D.sub(0,-1,0,-2)); //prepend 1s to inputs
  cout <<"X dim = " <<X.dim() <<endl;
  cout <<"Y dim = " <<Y.dim() <<endl;

  //compute optimal beta
  arr beta = inverse(~X*X)*~X*Y;
  cout <<"optimal beta=" <<beta <<endl;

  //display the function
  arr X_grid = grid(2, -3, 3, 30);
  X_grid = catCol(ones(X_grid.d0,1), X_grid);
  cout <<"X_grid dim = " <<X_grid.dim() <<endl;

  arr Y_grid = X_grid * beta;
  cout <<"Y_grid dim = " <<Y_grid.dim() <<endl;
  FILE("z.2") <<Y_grid.reshape(31,31);
  gnuplot("splot 'z.1' us 1:2:3 w p, 'z.2' matrix us ($1/5-3):($2/5-3):3 w l", true);

  cout <<"CLICK ON THE PLOT!" <<endl;
}

//===========================================================================

int main(int argc, char *argv[]) {
  MT::initCmdLine(argc,argv);

  gettingStarted();

  return 0;
}
```

Matlab

(by Peter Englert)

```
clear;

% load the date
load('dataLinReg2D.txt');

% plot it
figure(1);clf;hold on;
plot3(dataLinReg2D(:,1),dataLinReg2D(:,2),dataLinReg2D(:,3),'r.');

% decompose in input X and output Y
n = size(dataLinReg2D,1);
X = dataLinReg2D(:,1:2);
Y = dataLinReg2D(:,3);

% prepend 1s to inputs
X = [ones(n,1),X];

% compute optimal beta
beta = inv(X'*X)*X'*Y;

% display the function
[a b] = meshgrid(-2:.1:2,-2:.1:2);
Xgrid = [ones(length(a(:)),1),a(:),b(:)];
Ygrid = Xgrid*beta;
Ygrid = reshape(Ygrid,size(a));
h = surface(a,b,Ygrid);
view(3);
grid on;
```