Softmax regresija

Tomaž Tomažič (63100281)

19. april 2015

1 Uvod

Cilj naloge je bil napovedati eno izmed devetih skupin, kateri pripada nek izdelek.

2 Podatki

Podatki so imeli 93 značilk katere so bile anonime. Vsi podatki so zasegali diskretne vrednosti in so bili nenegativni. Na voljo je bilo 50000 učnih primerov, v katerih sta močno prevladala drugi in šesti razred.

3 Metode

Ker rešujemo večrazredni problem sem uporabil logistično regresijo v kombinaciji one-vs-all in softmax regresijo. Za obe metodi sem uporabil prečno preverjanje za pridobitev najboljšega regularizacijskega paramtera lambda. Za oceno napake sem uporabil funkcijo log_loss() iz knjižnjice scipy. Metoda one-vs-all je neodvisna od implementacije, saj sprejme cenilno funkcijo in njen odvod kot argument. Deluje tako da zgradim 9 modelov, za vsak razred posebej. Podatke sem poizkušal normalizirati z razredoma Binarizer in Normalizer. Slednji je bil po prečnem preverjanju mnogo boljši in sem ga zato tudi uporail. Prečno preverjanje sem izvajal na 4 foldih.

Za obe metodi sem tudi numerično preveril pravilnost gradienta.

4 Rezultati

Rezultati so pokazali da sta si metodi softmax in one-vs-all zelo podobni, vendar je softmax vseeno nekoliko boljši.

Ker so razredi neeakomerno porazdeljeni s stevilom ucnih primerov, bi bilo tudi smiselno pogledati F1 score. Tega zal nisem uspel narediti.

5 Izjava o izdelavi domače naloge

Domačo nalogo in pripadajoče programe sem izdelal sam.

Tabela 1: Rezultati na strežniku

model	rezultat (9b7)
one-vs-all	0.69286
softmax	0.61184

Priloge

A Programska koda

```
import csv
import Orange
import numpy as np
import scipy as sp
import scipy.optimize
from sklearn import preprocessing
from sklearn import metrics
from copy import deepcopy
from sklearn import cross_validation

epsilon = 0.00000001

def readFile(path="data4_reduced.csv"):
    with open(path, newline='') as csvfile:
        data = csv.reader(csvfile, delimiter=',')
        data = [i[1:-1] + [i[-1][-1]] for i in data]

    data.pop(0)
    data = np.array(data).astype(int)
    dataX = data[:, :-1]
    dataX = stackOnes(dataX)
    dataY = data[:, -1]
    return dataX, dataY

def readTestFile(path="test.csv"):
    with open(path, newline='') as csvfile:
        data = csv.reader(csvfile, delimiter=',')
        data = [i[1:] for i in data]

    data.pop(0)
    data = np.array(data).astype(int)
    data = stackOnes(data)
    return data

def numerical_grad(f, params, epsilon):
    num_grad = np.zeros(len(theta))
    perturb = np.zeros(len(params))
    for i in range(params.size):
        perturb[i] = epsilon
        j1 = f(params + perturb)
        j2 = f(params - perturb)
        num_grad[i] = (j1 - j2) / (2. * epsilon)
        perturb[i] = 0
    return num_grad
```

```
def cost_softmax(th, X, Y, _lambda=0.001):
    theta = th.reshape(Y.shape[1], X.shape[1])

    M = X.dot(theta.T)
    P = np.exp(M - np.max(M, axis=1)[:, None])
    P /= np.sum(P, axis=1)[:, None]

    cost = -np.sum(np.log(P+epsilon) * Y)
    cost += _lambda * th.dot(th) / 2.0
    cost /= X.shape[0]

    return cost

def grad_softmax(th, X, Y, _lambda):
    theta = th.reshape(Y.shape[1], X.shape[1])

    M = X.dot(theta.T)
    P = np.exp(M - np.max(M, axis=1)[:, None])
    P /= np.sum(P, axis=1)[:, None]

    grad = X.T.dot(P - Y).T
    grad += _lambda * theta
    grad /= X.shape[0]
    return grad.ravel()

def train_softmax(X, Y, alpha):
    '''train theta '''
    theta = np.zeros(numClasses * numFeatures)
    theta, j, d = scipy.optimize.fmin_l_bfgs_b(cost_softmax, theta, fprime=grad_softmax, args
    if d['warnflag'] != 0:
        print(d['warnflag'])
    theta = theta.reshape((numClasses, numFeatures))
    return theta

def CV_softmax(X, Y, alpha):
    score = 0.0
    K = 4
    for i in range(K): #K random splits
        trainX, testX, trainY, testY = cross_validation.train_test_split(X, Y, test_size=0.2,
        trainY = np.eye(numClasses)[trainY-1]
        #predict
        theta = train_softmax(trainX, trainY, alpha)
        prediction = predict_class(theta, testX)
        score += metrics.log_loss(testY-1, prediction)
    return score/K

def CV_one_VS_all(X, Y, alpha):
    score = 0.0
    K = 4
    for i in range(K): #K random splits
        trainX, testX, trainY, testY = cross_validation.train_test_split(X, Y, test_size=0.2,
        #predict
        theta, prediction = one_VS_all(trainX, trainY, testX, cost_log_reg, grad_log_reg, alp
        score += metrics.log_loss(testY-1, prediction)
    return score/K

def getAlpha(cvfun, X, Y):
    alphas = np.hstack((np.linspace(0.01, 1, 20), np.linspace(1, 100, 20)))
    # alphas = np.hstack((np.linspace(20, 200, 20)))
    best_alpha = (alphas[0], 100)
```

```
    for i in alphas:
        score = cvfun(X, Y, i)
        if (score < best_alpha[1]):
            best_alpha = (i, score)
        print(i, score)
    print(best_alpha)
    return best_alpha[0]

def predict_class(theta, X):
    M = X.dot(theta.T)
    P = np.exp(M - np.max(M, axis=1)[:, None])
    P /= np.sum(P, axis=1)[:, None]
    return P

def sigmoid(z):
    r = 1. / (1 + np.power(np.e, -z))
    return r

def cost_log_reg(theta, X, Y, _lambda=0.001):
    m = Y.shape[0]
    h0 = sigmoid(X.dot(theta))
    regularized_theta = np.hstack((0, theta[1:]))
    cost = 1/m * (-Y.T.dot(np.log(h0+epsilon)) - ((1-Y).T.dot(np.log(1-h0+epsilon))))
    cost += (_lambda/(2*m)) * (regularized_theta.dot(regularized_theta))
    return cost

def grad_log_reg(theta, X, Y, _lambda=0.001):
    m = Y.shape[0]
    h0 = sigmoid(X.dot(theta))
    regularized_theta = np.hstack((0, theta[1:]))
    grad = 1/m * ((h0 - Y).T.dot(X)).T
    grad += _lambda/m * regularized_theta;
    return grad

def stackOnes(data):
    return np.column_stack((np.ones(data.shape[0]), data))


def savePrediction(p):
    f = open("result.csv", 'w')
    f.write("id,Class_1,Class_2,Class_3,Class_4,Class_5,Class_6,Class_7,Class_8,Class_9\n")
    np.set_printoptions(suppress=True)
    np.set_printoptions(precision=7)
    for i, j in enumerate(p):
        f.write(str(i+1) + "," + ",".join(j.astype(str)) + "\n")
    f.close()

def one_VS_all(X, Y, testX, cost=cost_log_reg, grad=grad_log_reg, alpha = 0.01):
    prediction = np.zeros((testX.shape[0], 9))
    theta = np.zeros((numFeatures, 9))
    theta_1_vs_all = []
    for i in range(9):
        newY = deepcopy(Y)
        newY[Y < i+1] = 0
        newY[Y == i+1] = 1
        newY[Y > i+1] = 0  # vse kar je 1 ostane 1, vecje od ena se spremeni v 0
        # print(newY.any())

        (theta[:, i], f, d) = scipy.optimize.fmin_l_bfgs_b(cost, theta[:, i], fprime=grad, ar
        if d['warnflag'] != 0:
```

```
            print(d['warnflag'])

    prediction = predict_class(theta.T, testX)
    return (theta, prediction)


##########################################
#read files
##########################################
# (X, Y) = readFile()
# (trainX, trainY) = readFile('predict.csv')
(X, Y) = readFile("train.csv")
trainX = readTestFile('test.csv')

numFeatures = X.shape[1]

#normalize
p = preprocessing.Normalizer().fit(X)
X = p.transform(X)
trainX = p.transform(trainX)


##########################################
# 1 vs all
##########################################
alpha = getAlpha(CV_one_VS_all, X, Y)
theta, prediction = one_VS_all(X, Y, trainX, cost_log_reg, grad_log_reg, alpha)
savePrediction(prediction)

# m = metrics.log_loss(trainY-1, prediction)

##########################################
# numerical gradient for logistic regression
##########################################
# theta = np.zeros(numFeatures)
# costng = lambda x: cost_log_reg(x, X, Y,0)
# gradng = lambda x: grad_log_reg(x, X, Y,0)
# ng = scipy.optimize.check_grad(costng, gradng, theta)
# print("ng", ng)# error == 1.02399876562e-06
##########################################


##########################################
# softmax
##########################################
prediction = np.zeros((trainX.shape[0], 9))
numClasses = 9
alpha = 0.16
alpha = getAlpha(CV_softmax, X, Y)
print(alpha)

#predict
newY = np.eye(numClasses)[Y-1]
theta = train_softmax(X, newY, alpha)
prediction = predict_class(theta, trainX)
savePrediction(prediction)

# m = metrics.log_loss(trainY-1, prediction)

##########################################
```

```
# softmax numerical gradient
##########################################
# theta = np.ones((numFeatures * numClasses))
# ng = numerical_grad(lambda params: cost_softmax(params, X, newY, 0), theta, 1e-5)
# ag = grad_softmax(theta, X, newY, 0)
# print(np.sum((ag - ng)**2))  # 8.40724893311e-14
# print(ag[:5])
# print(ng[:5])
##########################################
```