Kombiniranje algoritmov strojnega učenja

Tomaž Tomažič (63100281)

7. april 2015

1 Uvod

Cilj naloge je zgraditi modele, ki napovejo vonj vsake kemikalje. Potrebno je napovedati intenziteto, všečnost vonja in 19 opisov kemikalij oziramo stopnjo podobnosti vonja.

2 Podatki

V datoteki *molecular_descriptors_data* so bili podani podatki značilnosti kemikalij. Za vsako kemikalijo je bilo podanih 4871 atributov. Podatki niso bili normalizirani. Približno 1800 atributov je bilo takšnih, da so imeli vse vrednosti enake in sem jih zato lahko enostavno izbrisal. V drugi datoteki $TrainSet - hw2$ so bili podatki o ocenah vonjav, katere je podalo 49 ocenjevalcev za vsako spojino. Vsaka spojina ima ocene za dve različni stopnji razredčenosti na ocenjevalca. Podatki ki jih napovedujemo zavzemajo vrednosti od 0 do 100. V slednji datoteki najdemo veliko manjkajočih ocen za določeno razredčenost.

3 Metode

Iz podatkov sem odstranil vse vrstice v katerih so manjkajoči podatki. odstranil sem tudi vse atribute ki so imeli enake vrednosti na vseh podatkih. Opise kemikalij sem združil s podatki o ocenah kemikalij.

Izbral sem 6 razlicnih metod napovedovanja: naključni gozdovi z vrečenjem, linearna regresija (Ridge/LASSO) s predhodno transformacijo podatkov z metodo glavnih komponent, linearna regresija (Ridge/LASSO) v prostoru originalnih značilk, Elastic net, KNN (k-nearest neighbors) in regresijsko drevo.

Iz vsake metode sem za vsak primer tako dobil 21 napovedi. Z 4 delnim prečnim preverjanjem sem zgradil matrike dimenzij mx21 katere sem uporabil za učenje modela agregacije z L2 regresijo. V končni fazi sem tako imel 21 modelov agregacije. Testne primere sem napovedal z modeli naučenimi nad celotnimi podatki.

Parametre modelov sem fixno določil in izbral glede na prečno preverjanje iz prejšnje naloge.

Podobno sem naredil tudi z povprečenjem, kjer sem namesto L2 regresije uporabil kar povprečja.

4 Rezultati

Zanimivo in nepričakovano je da sem dobil boljše rezultate z povprečenjem. Prav tako sem dobil veliko slabše rezultate z vključitvijo transformiranih podatkov z metodo PCA

Tabela 1: Rezultati na strežniku

model	rezultat (9b7)
povprečenje	3.73
ansambli	3.43

5 Izjava o izdelavi domače naloge

Domačo nalogo in pripadajoče programe sem izdelal sam.

Priloge

A Programska koda

```
import numpy as np
import csv
import Orange
import math
import scipy.stats
from collections import defaultdict
from scipy import optimize
from sklearn.cross_validation import train_test_split
from sklearn.cross_validation import KFold
from sklearn.decomposition import PCA

names = ["INTENSITY/STRENGTH", "VALENCE/PLEASANTNESS", "BAKERY", "SWEET", "FRUIT", "FISH", "C

class BaggedLearner(Orange.classification.Learner):
    """Bootstrap aggregating learner."""
    def __init__(self, learner, k=3):
        self.k = k # number of bootstrap samples and corresponding models
        self.learner = learner # base learner
        self.name = "bagged␣" + self.learner.name

    def __call__(self, data):
        """Returns a bagged model inferred from the training set."""
        models = []
        for epoch in range(self.k):
            indices = np.random.randint(len(data), size=len(data)) # sample indices
            models.append(self.learner(data[indices])) # model inference from a sample
        model = BaggedModel(models) # return a predictor that remembers a list of models
        model.name = self.name
        return model

class BaggedModel(Orange.classification.Model):
```

```
    """Bootstrap aggregating classifier."""
    def __init__(self, models):
        self.models = models # a list of predictors

    def __call__(self, data, ret=Orange.classification.Model.Value):
        """Given a data instance or table of data instances returns predicted class."""
        return np.mean([m(data) for m in self.models], axis=0)


def filterSameColumns(data):
    return data[:, np.min(data, axis=0) != np.max(data, axis=0)]

def convertDillusion(string):
    newStr = string[2:].replace(",", ".")
    newStr = newStr[:4] + newStr[4:].replace(".", "")
    return float(newStr)

def hasNaN(row):
    for i in row:
        if i == "NaN":
            return True
    return False

def replaceNaN(row):
    return [(0 if i == "NaN" else i) for i in row]

def buildMerge(a, bdict):
    return np.array(np.concatenate(( [a[0], a[5], (1 if a[3] == "high␣" else 0), convertDillu
bdict[np.int(a[0])], a[6:] ))).astype(float)

def simpleMerge(a, bdict):
    return np.concatenate(( [convertDillusion(a[1])],  bdict[np.int(a[0])] ))

def readFiles(f1 = "merge1.txt", f2 = "merge2.txt"): #f2 is molecular description

    #get table
    with open(f2) as csvfile:
        r = csv.reader(csvfile, delimiter='\t')
        data_description = [replaceNaN(i) for i in r]

    #remove first useless row (name of columns)
    [data_description.pop(0) for i in range(1)]

    #convert to numpy arrays
    data_description = np.array(data_description).astype(float)

    #filter zeros rows or rows with same value
    data_description = filterSameColumns(data_description)

    #build dict
    d = dict.fromkeys(data_description[:, 0])
    for i in range(len(data_description[:,0])):
        d[data_description[i,0]] = data_description[i, 1:]

    #open first file
    data = []
    with open(f1) as csvfile:
        r = csv.reader(csvfile, delimiter='\t')
        #append non NaN rows
        data = [i for i in r if not hasNaN(i)]
```

```
    #remove first useless row (name of columns)
    [data.pop(0) for i in range(1)]

    #merge and remove second and third useless columns
    #change order of rows so and intensity is just 0/1 and dellusion is decimal
    newData = np.vstack([buildMerge(i,d) for i in data])



    return (newData, d)


def avgOnesZeros(i, data, ones):
    return np.average(data[ np.logical_and(data[:, 0] == i, ones) ], axis=0)

def avgData(data):
    ids = np.unique(data[:,0])
    ones = data[:, 2] == 1
    zeros = data[:, 2] == 0
    avgData0 = [ avgOnesZeros(i, data, zeros) for i in ids]
    avgData1 = [ avgOnesZeros(i, data, ones) for i in ids]
    return np.vstack((avgData0, avgData1))

def splitDataVertical(data):
    X = data[:,3:-21]
    Y = data[:,-21:]
    return (X, Y)

def splitDataHorizontal(X, Y, percentage=0.85):
    l = X.shape[0]
    bound = np.round(l * percentage)
    return (X[:bound], Y[:bound], X[bound:], Y[bound:])

def getShape(arr):
    return (np.shape(arr)[0], np.shape(arr)[1])

###### MODELS #############
def ridgeModel(X, Y, alpha=0.1):
    ridge = Orange.regression.RidgeRegressionLearner(copy_X=True, fit_intercept=True, normali
    data = Orange.data.Table(X, Y)
    return ridge(data)

def ridgeModels(X, Y, alpha=0.1):
    models = []
    ridge = Orange.regression.RidgeRegressionLearner(copy_X=True, fit_intercept=True, normali
    for i in range(21):
        data = Orange.data.Table(X[:, :], Y[:, i])
        model = ridge(data)
        models.append(model)
    return models

def ridgeModelsPCA(X, Y, alpha=0.1):
    models = []
    ridge = Orange.regression.RidgeRegressionLearner(copy_X=True, fit_intercept=True, normali
    for i in range(21):
        data = Orange.data.Table(X[:, :], Y[:, i])
        model = ridge(data)
        models.append(model)
    return models
```

```
def elasticModels(X, Y, alpha=0.1, ratio=0.15):
    models = []
    elastic = Orange.regression.ElasticNetLearner(copy_X=True, fit_intercept=True, normalize=
    for i in range(21):
        data = Orange.data.Table(X[:, :], Y[:, i])
        model = elastic(data)
        models.append(model)
    return models

def forestModels(X, Y, n):
    models = []
    tree = Orange.classification.SimpleRandomForestLearner(n_estimators=n)
    bltree = BaggedLearner(tree, k=50)
    for i in range(21):
        data = Orange.data.Table(X[:, :], Y[:, i])
        model = bltree(data)
        models.append(model)
    return models


def knnModels(X, Y, neigbors=10):
    models = []
    knn =   Orange.classification.KNNLearner(n_neighbors=neigbors)
    for i in range(21):
        data = Orange.data.Table(X[:, :], Y[:, i])
        model = knn(data)
        models.append(model)
    return models

def regressionTreeModels(X, Y, alpha):
    models = []
    tree = Orange.classification.TreeLearner()
    for i in range(21):
        data = Orange.data.Table(X[:, :], Y[:, i])
        model = tree(data)
        models.append(model)
    return models

###### Cross Validation ########

def pearson(x,y):
    x,y = np.hstack(x), np.array(y)
    r = scipy.stats.pearsonr(x,y)[0]
    return 0. if math.isnan(r) else r

def final_score(rs):
    NORM_STD = [ 0.18, 0.16, 0.06 ]
    zs = rs/NORM_STD
    return np.mean(zs)

def evaluate(predicted, real):
    rint = pearson(predicted[:, 0], real[:, 0])
    rval = pearson(predicted[:, 1], real[:, 1])
    rdecall = [ pearson(predicted[:,i], real[:,i]) for i in range(2,21) ]
    rdec = np.mean(rdecall)
    return final_score(np.array([rint, rval, rdec]))

def CV(X, Y, model, alpha):
    score = 0.0
```

```
    K = 10
    for i in range(K): #K random splits
        trainX, testX, trainY, testY = train_test_split( X, Y, test_size=0.15, random_state=r
        models = model(trainX, trainY, alpha)
        predictions = predict(models, testX)
        score += evaluate(predictions, testY)
        print(i)

    return score/K

def getAlpha(X, Y, model):
    # alphas = np.hstack((np.linspace(0.000001, .1, 10), np.linspace(1, 5000, 10)))
    # alphas = np.hstack((np.linspace(0.01, 1, 10), np.linspace(1, 100, 10)))
    maxScore = (0, 0)
    for i,j in enumerate(alphas):
        score = CV(X, Y, model, j)
        print(j, score)
        if score > maxScore[0]:
            maxScore = (score, i)
    return alphas[maxScore[1]]

################################
def predict(models, X):
    predictions = []
    for i in X:
        predictions.append([models[j](i) for j in range(21)])
    return np.array(predictions)

def predictOne(models, X):
    return np.hstack([i(X) for i in models])

def predictFromFile(models, d, f="predict.txt",):
    predictions = []
    global names
    with open(f) as csvfile:
        r = csv.reader(csvfile, delimiter='\t')
        for i in r:
            build = simpleMerge(i, d)
            for j in range(21):
                predictions.append([i[0], names[j], models[j](build)])

    return predictions

def limit(a):
    a = np.abs(a)
    return 100 if a >= 100 else a

def printPredictions(predictions, printer=1):
    if printer:
        for i in predictions:
            print('%s\t%s\t%f' % (i[0], i[1], limit(i[2])))
    else:
        with open('prdictions.txt', 'wb') as f:
            for i in predictions:
                f.write('%s\t%s\t%f\n' % (i[0], i[1], limit(i[2])))

def lvl1(X, Y):
    l = len(Y)
    k = 4
    newY = [np.zeros((l,21)) for i in range(6)]
```

```
    kf = KFold(l, n_folds=k)

    global stacking_models
    global stacking_alphas

    for train, test in kf:
        for i, model in enumerate(stacking_models):
           m = model(X[train], Y[train], stacking_alphas[i])
           newY[i][test, :] = np.hstack(predict(m, X[test])).transpose()  #shape (129, 21)

    return newY

def lvl2(X, Y):
    models = []
    l = len(X)
    for i in range(21):
        newX = np.zeros((X[0].shape[0], l))
        for j in range(l):
            newX[:, j] = X[j][:, i]

        models.append(ridgeModel(newX, Y[:, i]))
    return models #21 models


def stacking(X, Y, lvl2model, f, d):
    global names
    global stacking_models
    global stacking_alphas

    predictions = []
    models = []

    #train lvl1
    for i, model in enumerate(stacking_models):
        models.append(model(X, Y, stacking_alphas[i]))

    with open(f) as csvfile:
        r = csv.reader(csvfile, delimiter='\t')
        for i in r:
            build = simpleMerge(i, d)

            # newY = [np.zeros((1, 21)) for i in range(6)]
            newX = [np.zeros((21)) for i in range(6)]

            for j,k in enumerate(models):
                newX[j]= predictOne(k, build)

            for j in range(21):
                #lvl2 build
                newY = np.zeros((6), dtype=np.int)
                for k in range(6):
                    newY[k] = newX[k][j]

                prediction = lvl2model[j](newY)[0]
                print("%s\t%s\t%f" % (i[0], names[j], limit(prediction)))

def meanStacking(X, Y, f, d):
    global names
    global stacking_models
    global stacking_alphas
```

```
    predictions = []
    models = []

    #train lvl1
    for i, model in enumerate(stacking_models):
        models.append(model(X, Y, stacking_alphas[i]))

    with open(f) as csvfile:
        r = csv.reader(csvfile, delimiter='\t')
        for i in r:
            build = simpleMerge(i, d)

            # newY = [np.zeros((1, 21)) for i in range(6)]
            newX = [np.zeros((21)) for i in range(6)]

            for j,k in enumerate(models):
                newX[j]= predictOne(k, build)

            for j in range(21):
                #lvl2 build
                newY = np.zeros((6), dtype=np.int)
                for k in range(6):
                    newY[k] = newX[k][j]

                prediction = np.mean(newY)
                print("%s\t%s\t%f" % (i[0], names[j], limit(prediction)))

#code start
#pca must be at the beginning
stacking_models = [knnModels, regressionTreeModels, ridgeModelsPCA, forestModels, ridgeModels
stacking_alphas = [10, 1, 3200, 20, 0.5, 0.01]
# stacking_alphas = [20, 0.5, 0.5, 0.5]

(data, d) = readFiles("TrainSet-hw2.txt", "molecular_descriptors_data.txt")
# newdata = bestHighLow(data, "predict.txt")

data = avgData(data)
(X, Y) = splitDataVertical(data)

# pca = PCA()
# Xpca = pca.fit_transform(X)

#first layer
newY = lvl1(X, Y)
#second layer
lvl2model = lvl2(newY, Y)
#final stacking
stacking(X, Y, lvl2model, "predict.txt", d)
# meanStacking(X, Y, "predict.txt", d)
```