Gradientni boosting

Tomazic Tomaz (63100281)

31. maj 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

Implementiral sem gradientni boosting ki za funkcijo izgube uporablja KL-divergenco (Kullback Leibler divergence). Razred GradBoost ob inicializaciji sprejme kot parameter šibak regresijski model, ki mora imeti implementirani funkciji 'predict' in 'fit'. Kot parameter sprejme tudi ime funkcije izgube. Trenutno je implementirana samo KL izguba. Razred 'GradBoost' deluje za vecrazredno klasifikacijo.

V fit metodi kot osnovno ucenje najprej izracunam frekvenco razredov nato pa izboljsujem napoved z regresijskim drevesom oz podanim ucnim modelom.

Napovedne modele sem ovrednotil z precnim preverjanjem v katerem sem iskal najboljse parametre za dane podatke. Ti parametri so: stevilo modelov za posamezni razred, globina regresijskega drevesa in regularizacijski parameter 'ro'. V precnem preverjanju sem poizkusal 30 modelov med 10 in 1000, globine drevesa od 1 do 5 in 5 regularizacijskij prametrov med 0.01 in 1.

Za zdruzevanje napovedi nisem uporabil metode stacking ampak samo povprecenje rezultatov mojih nevronskih mrez in rezultate gradient boostinga, ker sem vseeno pricakoval nekoliko boljse rezultate na lestvici.

4 Rezultati

Presenetilo me je da sem najboljse rezultate dobil z regularizacijskim parametrom vrednosti 1.

V tabeli 1 so najboljsi rezultati pri izbiri paramerov za doloceno globino drevesa. V tabeli-2 pa rezultati precnega preverjanja

Tabela 1: Rezultati povprecja treh foldov pri razlicnih parametrih.

st iteracij	globina drevesa	ro	rezultat
500	1	1	0.66513236171212353
500	2	1	0.57993880037618517
500	3	1	0.54497348832603998
500	4	1	0.53358386235816502
319	5	1	0.53697762802747351
625	4	1	0.53264532029139089

Tabela 2: Rezultati precnega preverjanja.

ime	st iteracij	globina drevesa	ro	vrednost modela	oddaja na strezniku (9b7)
boosting	625	4	1	0.51243599428	0.51648
nevronske mreze		51	0.003	0.54092656750	0.53926
povprecenje					0.49849

Rezultati ocitno kazejo da je boosting boljsa metoda v primerjavi z nevronskimi mrezami za dane podatke.

5 Izjava o izdelavi domače naloge

Domačo nalogo in pripadajoče programe sem izdelal sam.

Priloge

A Programska koda

```
import numpy as np
import sklearn
import copy
from Frequency import Frequency

class GradBoost:
    """Gradient Boosting for Classification."""

    def __init__(self, learner, n_estimators=100, loss="KL", epsilon=1e-5):
        self.n_estimators = n_estimators
        self.learner      = learner
        self.epsilon      = epsilon

        losses = {
            "huber": self.grad_huber_loss,
            "squared": self.grad_squared_loss,
            "abs": self.grad_abs_loss,
            "KL": self.kl_loss
        }
        self.loss = losses[loss]

    def grad_squared_loss(self, y, f):
        """Negative gradiant for squared loss."""
        return y - f

    def grad_abs_loss(self, y, f):
        """Negative gradient for absolute loss."""
        return np.sign(y - f)

    def grad_huber_loss(self, y, f, delta=0.5):
        """Negative gradient for Huber loss."""
        r0 = y - f
        r1 = delta * np.sign(r0)
        return np.vstack((r0, r1)).T[np.arange(y.shape[0]), (np.abs(r0)>delta).astype(int)]

    def kl_loss(self, y, f):
        """Negative gradient for kullback leibler."""
        return y - f

    def fit(self, X, Y, ro=1):
        num_c = Y.shape[1]

        #average
        models = [[Frequency().fit(X, Y[:, i]) for i in range(num_c)]]
        F      = np.hstack([i.predict(X) for i in models[0]])

        grad = self.loss(Y, F)

        for i in range(self.n_estimators):

            models.append([])

            for j in range(num_c):
                #fit regression tree
                dtc = copy.copy(self.learner)
                dtc.fit(X, grad[:, j])
                models[i+1].append(dtc)

                F[:, j] += dtc.predict(X) * ro

            P    = self.softmax(F)
            grad = self.loss(Y, P)

        self.models = np.array(models)
        return self


    def softmax(self, f):
        s = np.exp(f - np.max(f, axis=1)[:,None])
        s /= np.sum(s, axis=1)[:, None]
        return s
```

```
def test_grad(self, Y):
    F = np.zeros(Y.shape)
    P = self.softmax(F)
    grad = self.loss(Y, P)

    # print(self.init_thetas)
    approx = np.absolute(self.grad_approx(Y, P))
    real = np.absolute(self.loss(Y, P))
    print( np.absolute(sum((approx - real))) )
    print( np.sum((approx - real)**2) )

def KL(self, Q, P):
    return np.sum(P * np.log(P / (np.exp(Q)/np.sum(Q))))

def grad_approx(self, Y, P, e=1e-4):
    num_grad = np.zeros_like(Y)
    perturb = np.zeros(Y.shape[0])
    for j in range(Y.shape[1]):
        for i in range(Y.shape[0]):
            perturb[i] = e
            j1 = self.KL(Y[:,j], P[:,j] + perturb)
            j2 = self.KL(Y[:,j], P[:,j] - perturb)
            num_grad[i, j] = (j1 - j2) / (2. * e)
            perturb[i] = 0
    return self.softmax(num_grad)

def predict_proba(self, X):
    return self.predict(X)

def predict(self, X):
    models = self.models
    num_c  = models.shape[1]

    P = np.hstack([i.predict(X) for i in models[0]])

    for i in range(num_c):
        P[:, i] += np.sum([j.predict(X) for j in models[1:, i]], axis=0)

    return self.softmax(P)

def get_params(self, deep=False):
    return {
        "learner": self.learner,
        "n_estimators": self.n_estimators
    }


##################### Frequency.py ####################
import numpy as np

class Frequency:
    def fit(self, X, y):
        self.model = np.sum(y, axis=0) / y.shape[0]
        return self

    def predict(self, X):
        return np.vstack([self.model for i in range(X.shape[0])])

##################### CV.py ####################
```

```
from GradBoost import GradBoost
import IO
import numpy as np
import Orange
from sklearn import preprocessing, metrics, grid_search
from sklearn.tree import DecisionTreeRegressor
from sklearn.cross_validation import cross_val_score, ShuffleSplit,
train_test_split


def cv_score(X, Y, split, n_estimators, depth):
    learner = DecisionTreeRegressor(max_depth=depth)
    gb = GradBoost(learner, n_estimators)
    return np.absolute(np.mean(cross_val_score(gb, X, Y, \
        cv=split, scoring='log_loss', n_jobs=4)
    ))

def find_params(X, Y): #n_estimators && depth
    n_estimators = list(map(int, np.linspace(10, 500, 20)))
    depths = list(map(int, np.linspace(1, 5, 5)))

    cv_split = ShuffleSplit(Y.shape[0], n_iter=3, test_size=0.3,
random_state=42)

    scores = []
    for j in depths:
        scores.append( min([(cv_score(X, Y, cv_split, i, j), i, j) for i in
 n_estimators]) )
        print(scores[-1])
    print(scores)
    score = min(scores)
    print("best n_estimators", score[1], "depth", score[2], " got mean
score ", score[0], " for 3 folds")
    return (score[1], score[2])

def eval_model(X, Y):
    '''evaluate model with best lambda on unseen data'''

    X_train, X_test, Y_train, Y_test = train_test_split(
        X, Y, test_size=0.2, random_state=42
    )

    n_estimators, depth = find_params(X_train, Y_train)
    learner = DecisionTreeRegressor(max_depth=depth)
    gb = GradBoost(learner, n_estimators)
    gb.fit(X_train, Y_train)
    y = gb.predict(X_test)
    result = metrics.log_loss(Y_test, y)
    print("this model got score ", result, " with n ", n_estimators)
    return (n_estimators, depth)

def predict(X_train, Y_train, X, filename="result"):
    n_estimators, depth = eval_model(X_train, Y_train)

    learner = DecisionTreeRegressor(max_depth=depth)
    gb = GradBoost(learner, n_estimators)
    gb.fit(X_train, Y_train)
    prediction = gb.predict(X)
    IO.savePrediction(prediction, filename)
    print("prediction finished")
```

```
def simple_prediction_test(X, Y, lambda_=0):
    learner = DecisionTreeRegressor(max_depth=4)
    gb = GradBoost(learner)

    X_train, X_test, Y_train, Y_test = train_test_split(
        X, Y, test_size=0.2, random_state=42
    )
    gb.fit(X_train, Y_train)
    prediction = gb.predict(X_test)
    # print(prediction)
    print(metrics.log_loss(Y_test, prediction))

#################### IO.py ###################

import csv
import numpy as np
from sklearn import preprocessing

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]
    dataY = data[:, -1] #from 1 to 9
    dataY = np.eye(np.max(dataY))[dataY.astype(int)-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)
    return data

def savePrediction(p, name="result"):
    f = open(name + ".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 normalize(X, X_test=None):
    p = preprocessing.Normalizer().fit(X)
    X = p.transform(X)
    if X_test is None:
        return (X)
    else:
        X_test = p.transform(X_test)
        return (X, X_test)
```

```
##################### boost.py ####################

import Orange
import numpy as np
import sklearn
import CV
import IO
from GradBoost import GradBoost
from sklearn.tree import DecisionTreeRegressor as DTC

numClasses = 3
iris = Orange.data.Table("iris")
X = iris.X
Y = np.eye(numClasses)[iris.Y.astype(int)]

# print(GradBoost().fit(X, y))
# print(CV.simple_prediction_test(X, Y))
# print(CV.eval_model(X, Y))


# X, Y = IO.readFile()
# print(CV.simple_prediction_test(X, Y))
# print(CV.eval_model(X, Y))

X, Y = IO.readFile("train.csv")
Y_test = IO.readTestFile()
CV.predict(X, Y, Y_test, "result")
```