Multi-class SVM using Pegasos

Vladimir Ventaniuc

May 25, 2019

Python code

```
import numpy as np

def generateWeightArray(d, norm):
    inv_d = 1.0 / d

    gauss = np.random.normal(size=d)
    length = np.linalg.norm(gauss)
    if length < norm:
        W = gauss
    else:
        r = np.random.rand() ** inv_d
        W = np.multiply(gauss, r / length)
    return W

def preprocessing():
    train_read = open(’mnist_train.txt’, ’r’)
    test_read = open(’mnist_test.txt’, ’r’)
    train_vectors = []
    train_answers = []
    test_vectors = []
    test_answers = []
    for line in train_read:
        new_vector = np.fromstring(line, dtype=int, sep=’,’)
        train_answers.append(new_vector[0])
        train_vectors.append([((2 * i) / 255) for i in new_vector[1:]])
    for line in test_read:
        new_vector = np.fromstring(line, dtype=int, sep=’,’)
        test_answers.append(new_vector[0])
        test_vectors.append([((2 * i) / 255) for i in new_vector[1:]])
    train_read.close()
    test_read.close()
    np.save(’train_save.npy’, train_vectors)
    np.save(’train_ans_save.npy’, train_answers)
    np.save(’test_save.npy’, test_vectors)
    np.save(’test_ans_save.npy’, test_answers)


def pegasos_svm_train(data, lam, label, trainingAnswers, W):
    print("Training classificator with label " + str(label))
    epoch = 0
    for i in range(100):
        for X, Y in zip(data, trainingAnswers):
            if Y != label:
                Y = -1
            epoch = epoch + 1
            learning_rate = 1 / (epoch * lam)
            if (Y * np.dot(W, X) < 1):
                W2 = np.add(W * (1 - learning_rate * lam), (learning_rate)*(
                    X*Y))
                W = minForMultiplycation(lam,W2) * W2

    return W

def pegasos_svm_test(testingAnswers, data, w):
    vectorCount = 0
    mislabel = 0
    for vec, answer in zip(data, testingAnswers):
        vectorCount += 1
        prediction_values = []
        for wvec in w:
            prediction_values.append(np.dot(vec, wvec))
        prediction = np.argmax(prediction_values)
        if (prediction == answer):
            continue
        else:
            mislabel += 1
    return (mislabel / vectorCount)

def show_final_results(trainingVectors, trainingAnswers, testingVectors,
    testingAns):
    final_vectors = []

    lambdas = [2 ** -5, 2 ** -4, 2 ** -3, 2 ** -2, 2 ** -1, 2 ** 0, 2 ** 1]
    for lam in lambdas:
        d = 784
        norm = 1/lam**0.5
        initial_weight = generateWeightArray(d, norm)
        print("Calculate weights for lambda = " + str(lam))
        for i in range(10):
            final_vectors.append(pegasos_svm_train(trainingVectors,lam, i,
                trainingAnswers, initial_weight))
        print("Final Testing Error with lambda " + str(lam)+": ",
            pegasos_svm_test(testingAns, testingVectors, final_vectors))

def minForMultiplycation(lam, W):
    val = (1/lam**0.5)/np.linalg.norm(W)
    return min(1,val)

def main():
    preprocessing()

    trainingVectors = np.load(’train_save.npy’)
    trainingAns = np.load(’train_ans_save.npy’)
    testingVectors = np.load(’test_save.npy’)
    testingAns = np.load(’test_ans_save.npy’)

    show_final_results(trainingVectors, trainingAns, testingVectors,
        testingAns)


if __name__ == "__main__":
    main()
```

Results

Epochs	λ	Error	Clasificator	Error for clasificator
200	2^{-5}	0.124	0	0.0879
			1	0.0465
			2	0.2151
			3	0.1538
			4	0.0777
			5	0.2447
			6	0.0435
			7	0.1282
			8	0.1136
			9	0.1569
	2^{-4}	0.276	0	0.3187
			1	0.1473
			2	0.3118
			3	0.3187
			4	0.2039
			5	0.3723
			6	0.0761
			7	0.2393
			8	0.3636
			9	0.4608
	2^{-3}	0.288	0	0.3407
			1	0.155
			2	0.3226
			3	0.3297
			4	0.2039
			5	0.4043
			6	0.0761
			7	0.2479
			8	0.3864
			9	0.4706

Epochs	λ	Error	Clasificator	Error for clasificator
500	2^{-5}	0.125	0	0.0989
			1	0.0465
			2	0.2151
			3	0.1538
			4	0.0583
			5	0.2553
			6	0.0435
			7	0.1368
			8	0.1136
			9	0.1569
	2^{-4}	0.287	0	0.3187
			1	0.1473
			2	0.3118
			3	0.3077
			4	0.2136
			5	0.3723
			6	0.0761
			7	0.265
			8	0.4318
			9	0.4804
	2^{-3}	0.3	0	0.3187
			1	0.155
			2	0.3333
			3	0.3297
			4	0.2233
			5	0.4043
			6	0.0761
			7	0.2735
			8	0.4545
			9	0.4902