线性支持向量机

姓名:李星沛 学号:22026056 学院:计算机学院

SVM 概念:

数据挖掘中常用的分类算法. 对于给定的训练数据集，我们可能有多条不同的线性方程可以将数据分隔开。SVM 分类算法的目标，就是找到最优划分的那条线性方程

SVM 类代码:

```
class LinearSVM(object):
    """
    线性 SVM 的实现类
    """
    def __init__(self, dataset_size, vector_size):
        self.__multipliers = np.zeros(dataset_size, np.float_)
        self.weight_vec = np.zeros(vector_size, np.float_)
        self.bias = 0

    def train(self, dataset, iteration_num):
        dataset = np.array(dataset,dtype=object)
        for k in range(iteration_num):
            self.__update(dataset, k)

    def __update(self, dataset, k):
        for i in range(dataset.__len__() // 2):
            j = (dataset.__len__() // 2 + i + k) % dataset.__len__()
            record_i = dataset[i]
            record_j = dataset[j]
            self.__sequential_minimal_optimization(dataset, record_i, record_j, i, j)
            self.__update_weight_vec(dataset)
            self.__update_bias(dataset)

    def __sequential_minimal_optimization(self, dataset, record_i, record_j, i, j):
        label_i = record_i[-1]
        vector_i = np.array(record_i[0])
        label_j = record_j[-1]
        vector_j = np.array(record_j[0])

        # 计算出截断前的记录 i 的‘拉格朗日乘子’unclipped_i
        error_i = np.dot(self.weight_vec, vector_i) + self.bias - label_i
        error_j = np.dot(self.weight_vec, vector_j) + self.bias - label_j
        eta = np.dot(vector_i - vector_j, vector_i - vector_j)
        Unclipped_i = self.__multipliers[i] + label_i * (error_j - error_i) / eta

        # 截断记录 i 的`拉格朗日乘子`并计算记录 j 的`拉格朗日乘子`
        constant = -self.__calculate_constant(dataset, i, j)
        multiplier = self.__quadratic_programming(Unclipped_i, label_i, label_j, i, j)
        if multiplier >= 0:
            self.__multipliers[i] = multiplier
            self.__multipliers[j] = (constant - multiplier * label_i) * label_j

    def __update_bias(self, dataset):
        sum_bias = 0
        count = 0
        for k in
range(self.__multipliers.__len__()):
            if self.__multipliers[k] != 0:
                label = dataset[k][-1]
                vector = np.array(dataset[k][0])
                sum_bias += 1 / label -
np.dot(self.weight_vec, vector)
                count += 1
        if count == 0:
            self.bias = 0
        else:
            self.bias = sum_bias / count

    def __update_weight_vec(self, dataset):
        weight_vector =
np.zeros(dataset[0][0].__len__())
        for k in range(dataset.__len__()):
            label = dataset[k][-1]
            vector = np.array(dataset[k][0])
            weight_vector +=
self.__multipliers[k] * label * vector
        self.weight_vec = weight_vector

    def __calculate_constant(self, dataset, i,
j):
        label_i = dataset[i][-1]
        label_j = dataset[j][-1]
        dataset[i][-1] = 0
        dataset[j][-1] = 0
        sum_constant = 0
        for k in range(dataset.__len__()):
            label = dataset[k][-1]
            sum_constant +=
self.__multipliers[k] * label
        dataset[i][-1] = label_i
        dataset[j][-1] = label_j
        return sum_constant
```

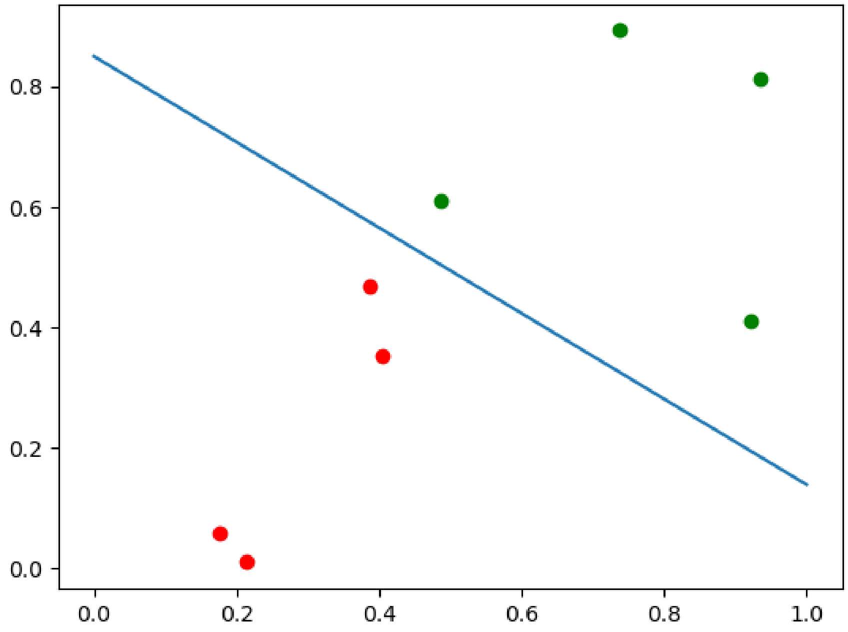
```
    def __quadratic_programming(self,
unclipped_i, label_i, label_j, i, j):
        multiplier = -1
        if label_i * label_j == 1:
            boundary = self.__multipliers[i] +
self.__multipliers[j]
            if boundary >= 0:
                if unclipped_i <= 0:
                    multiplier = 0
                elif unclipped_i < boundary:
                    multiplier = unclipped_i
                else:
                    multiplier = boundary
        else:
            boundary = max(0,
self.__multipliers[i] - self.__multipliers[j])
            if unclipped_i <= boundary:
                multiplier = boundary
            else:
                multiplier = unclipped_i
        return multiplier

    def predict(self, vector):
        result = np.dot(self.weight_vec,
np.array(vector)) + self.bias
        if result >= 0:
            return 1
        else:
            return -1

    def __str__(self):
        return "multipliers:" +
self.__multipliers.__str__() + '\n' + \
               "weight_vector:" +
self.weight_vec.__str__() + '\n' + \
               "bias:" + self.bias.__str__()
```

SVM 执行代码：

```
from LinearSVM import LinearSVM
import numpy as np
from matplotlib import pyplot as plt

dataset = [
    [[0.3858, 0.4687], 1],
    [[0.4871, 0.611], -1],
    [[0.9218, 0.4103], -1],
    [[0.7382, 0.8936], -1],
    [[0.1763, 0.0579], 1],
    [[0.4057, 0.3529], 1],
    [[0.9355, 0.8132], -1],
    [[0.2146, 0.0099], 1]
]

linearSVM = LinearSVM(dataset.__len__(),
dataset[0][0].__len__())
linearSVM.train(dataset, 100)
print(linearSVM)
```

```
for record in dataset:
    vector = record[0]
    label = record[-1]
    if label == 1:
        plt.plot(vector[0], vector[1], 'r-o')
    else:
        plt.plot(vector[0], vector[1], 'g-o')

    predict = linearSVM.predict(vector)
    print(record.__str__() + predict.__str__()
+ '\n')

x1 = np.linspace(0, 1, 50)
x2 = (-linearSVM.bias - linearSVM.weight_vec[0]
* x1) / linearSVM.weight_vec[1]
plt.plot(x1, x2)
plt.show()
```

实现结果：


0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0