

AI学习笔记--sklearn--perceptron算法

• 概述

感知器（perceptron）。是一种前向结构的人工神经网络，映射一组输入向量到一组输出向量。感知机是一种线性分类模型。可以被看做是一个有向图。除了输入节点，每个节点都是一个带有非线性激活函数的神经元（或称处理单元）。感知器无法实现对线性不可分数据识别。

• 算法解析

感知机实际上表示为输入空间到输出空间的映射函数，如下所示：

$$f(x) = \text{sign}(w \bullet x + b)$$

其中， w 和 b 称为感知机的模型参数， $w \in \mathbb{R}^n$ 叫做权值（weight）或权值向量（weight vector）， $b \in \mathbb{R}$ 叫做偏置（bias）， $w \bullet x$ 是 w 和 x 的内积， sign 是符号函数，其定义形式如下：

$$\text{sign} = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

假如，我们的训练数据集是一个线性可分的数据模型，那么，感知机学习数据的目的是为了取得一个能够将训练集正实例化点和负实例化点准确分开的一个超平面。为了求得这个超平面函数，我们需要依据算法来确定 w 和 b 两个参数。需要定义损失函数，并且尽量符合实际的数据趋势。

损失函数是使用的误分类点到超平面 S 的总距离，因此输入空间 R 中任何一点到超平面 S 的距离满足公式：

$$\frac{1}{\|w\|} |wx + b|$$

在这里， $\|w\|$ 是 w 的L2范数。对于输入分类数据 (x_i, y_i) 来说，满足以下公式：

$$-y_i(wx_i + b) > 0$$

因为当 $wx_i + b > 0$ 时， $y_i = -1$ ，而当 $wx_i + b < 0$ 时， $y_i = +1$ ，因此分类点 x_i 到超平面 S 的距离值可以推为：

$$-\frac{1}{\|w\|} y_i (wx_i + b)$$

这样，假定超平面 S 的误分类点集合为 M ，那么所有误分类点到超平面 S 的总距离可以表示为（其中 M 为误分类点的集合）：

$$-\frac{1}{\|w\|} \sum_{x_i \in M} y_i (wx_i + b)$$

如果不考虑 $1/\|w\|$ ，就可以求出感知机学习的损失函数为：

$$L(w, b) = - \sum_{x_i \in M} y_i (wx_i + b)$$

• 感知机算法损失函数最优解问题

感知机算法是使得损失函数 $L(w, b)$ 最小的优化问题，有几种常用的方法，比如可以使用梯度下降法来优化。假设误分类点集合 M 是固定的，那么损失函数 $L(w, b)$ 的梯度由：

$$\begin{aligned} \nabla_w L(w, b) &= - \sum_{x_i \in M} y_i x_i \\ \nabla_b L(w, b) &= - \sum_{x_i \in M} y_i \end{aligned}$$

给出随机选取一个误分类点 (x_i, y_i) ，对 w ， b 进行更新：

$$\begin{aligned} w &\leftarrow w + \eta y_i x_i \\ b &\leftarrow b + \eta y_i \end{aligned}$$

其中

η ($0 < \eta \leq 1$) 称为学习率 (learning rate)，这样通过迭代可以使得损失函数 $L(w, b)$ 不断减小，直到为0。

- 算法demo

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

"""
author: Yang Liu

The primitive form of the perceptron
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

def loadData():
    data = np.loadtxt('../data/testSet.txt')
    dataMat = data[:, 0:2]
    labelMat = data[:, 2]
    return dataMat, labelMat

def sign(val):
    if val >= 0:
        return 1
    else:
        return -1

def trainPerceptron(dataMat, labelMat, eta):
    """
    训练模型
    eta: learning rate
    """
    m, n = dataMat.shape
    weight = np.zeros(n)
    bias = 0

    flag = True
    while flag:
        for i in range(m):
            if np.any(labelMat[i] * (np.dot(weight, dataMat[i]) + bias) <= 0):
                weight = weight + eta * labelMat[i] * dataMat[i].T
                bias = bias + eta * labelMat[i]
                print("weight, bias: ")
                print(weight)
                print(bias)
                flag = True
                break
            else:
                flag = False

    return weight, bias

# 可视化展示分类结果
def plotResult(dataMat, labelMat, weight, bias):
    fig = plt.figure()
    axes = fig.add_subplot(111)

    type1_x = []
    type1_y = []
    type2_x = []
    type2_y = []
    for i in range(len(labelMat)):
        if (labelMat[i] == -1):
            type1_x.append(dataMat[i][0])
            type1_y.append(dataMat[i][1])
        if (labelMat[i] == 1):
```

```

type2_x.append(dataMat[i][0])
type2_y.append(dataMat[i][1])

type1 = axes.scatter(type1_x, type1_y, marker='x', s=20, c='red')
type2 = axes.scatter(type2_x, type2_y, marker='o', s=20, c='blue')

y = (0.1 * -weight[0] / weight[1] + -bias / weight[1], 4.0 * -weight[0] / weight[1] + -bias /
weight[1])
axes.add_line(Line2D((0.1, 4.0), y, linewidth=1, color='blue'))


plt.xlabel('X')
plt.ylabel('Y')

plt.show()

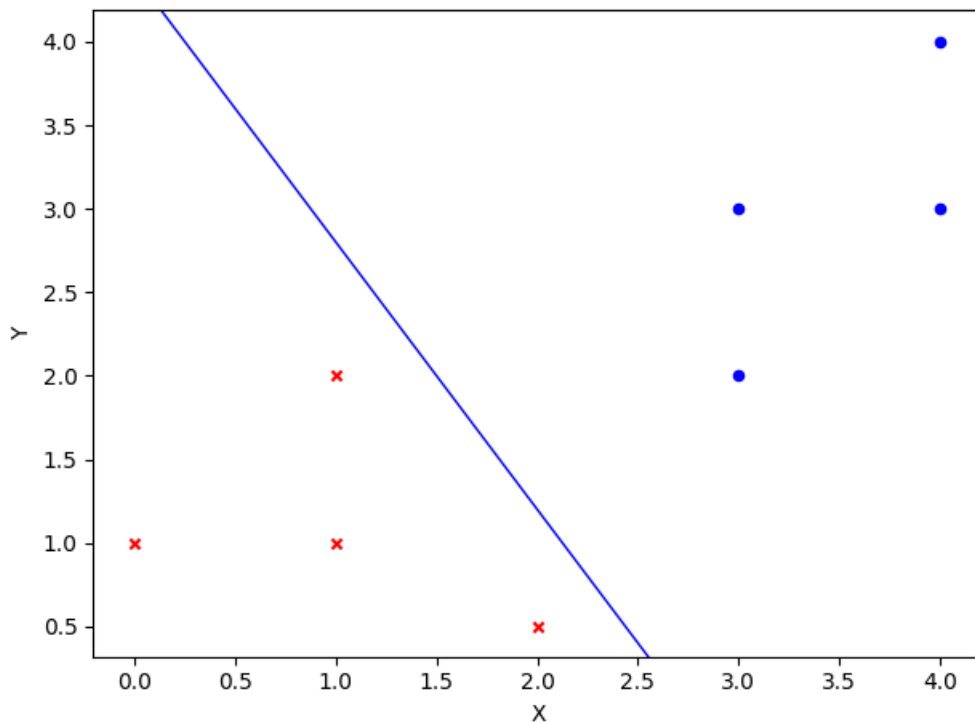
if __name__ == "__main__":
    dataMat, labelMat = loadData()
    weight, bias = trainPerceptron(dataMat, labelMat, 1)
    plotResult(dataMat, labelMat, weight, bias)

```

训练数据：

 testSet.txt

运行结果：



• Demo2:

```

# encoding=utf-8

import pandas as pd
import random
import time

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

```

```

class Perceptron(object):

    def __init__(self):
        self.learning_step = 0.001 # 学习率
        self.max_iteration = 5000 # 分类正确上界, 当分类正确的次数超过上界时, 认为已训练好, 退出训练

    def train(self, features, labels):

        # 初始化w, b为0, b在最后一位
        self.w = [0.0] * (len(features[0]) + 1)

        correct_count = 0 # 分类正确的次数

        while correct_count < self.max_iteration:

            # 随机选取数据(xi, yi)
            index = random.randint(0, len(labels) - 1)
            x = list(features[index])
            x.append(1.0) # 加上1是为了与b相乘
            y = 2 * labels[index] - 1 # label为1转化为正实例点+1, label为0转化为负实例点-1

            # 计算w*xi+b
            wx = sum([self.w[j] * x[j] for j in range(len(self.w))])

            # 如果yi(w*xi+b) > 0 则分类正确的次数加1
            if wx * y > 0:
                correct_count += 1
                continue

            # 如果yi(w*xi+b) <= 0 则更新w(最后一位实际上b)的值
            for i in range(len(self.w)):
                self.w[i] += self.learning_step * (y * x[i])

    def predict_(self, x):
        wx = sum([self.w[j] * x[j] for j in range(len(self.w))])
        return int(wx > 0) # w*xi+b>0则返回1, 否则返回0

    def predict(self, features):
        labels = []
        for feature in features:
            x = list(feature)
            x.append(1)
            labels.append(self.predict_(x))
        return labels

if __name__ == '__main__':

    print("Start read data")

    time_1 = time.time()

    raw_data = pd.read_csv('../data/train_binary.csv', header=0) # 读取csv数据, 并将第一行视为表头, 返回DataFrame类型
    data = raw_data.values

    features = data[:, 1:]
    labels = data[:, 0]

    # 避免过拟合, 采用交叉验证, 随机选取33%数据作为测试集, 剩余为训练集
    train_features, test_features, train_labels, test_labels = train_test_split(features, labels,
                                         test_size=0.33, random_state=0)

    time_2 = time.time()
    print('read data cost %f seconds' % (time_2 - time_1))

    print('Start training')

```

```
p = Perceptron()
p.train(train_features, train_labels)

time_3 = time.time()
print('training cost %f seconds' % (time_3 - time_2))

print('Start predicting')
test_predict = p.predict(test_features)
time_4 = time.time()
print('predicting cost %f seconds' % (time_4 - time_3))

score = accuracy_score(test_labels, test_predict)
print("The accruacy score is %f" % score)
```

输出结果：

```
Start read data
3.0
read data cost 6.213838 seconds
Start training
training cost 1.102169 seconds
Start predicting
predicting cost 3.904045 seconds
The accruacy score is 0.984429
```