svm

1. 1. 介绍

1.1 概念

支持向量机(support vector machines，svm)是二分类模型，svm 还包括核技巧，这使 svm 成为实质上的非线性分类器

基本模型：定义在特征空间上的间隔最大的线性分类器

学习策略：就是间隔最大化，最终形成一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题

学习算法：求解凸二次规划的最优化算法

支持向量：在决定分离超平面时只有支持向量起作用，而其他的实例点并不起作用，所以称为支持向量机，与逻辑回归不同的是 lr 是通过全局样本数据训练得到的，svm 是少数支称向量

1.2 模型介绍

线性可分 svm：当训练数据线性可分时，通过硬间隔最大化，学习一个线性分类器，又称为硬间隔 svm

线性 svm：当训练数据近似线性可分时，通过软间隔最大化，学习一个线性分类器，又称为软间隔 svm

非线性 svm：当训练数据线性不可分时，通过核技巧和软间隔最大化，学习非线性 svm

注：

核函数：将输入空间映射到特征空间得到的特征向量之间的内积

核方法：隐式的在高维空间中学习线性 svm

2. 线性可分 svm

2.1 定义：

定义 7.1（线性可分支持向量机） 给定线性可分训练数据集，通过间隔最大化或等价地求解相应的凸二次规划问题学习得到的分离超平面为

$$w^* \cdot x + b^* = 0 \tag{7.1}$$

以及相应的分类决策函数

$$f(x) = \text{sign}(w^* \cdot x + b^*) \tag{7.2}$$

称为线性可分支持向量机。

法向量指向的一侧为正类，另一侧为负类

2.2 函数间隔和几何间隔

引言：一个点到分类超平面的距离可以表示分类预测的确信度

函数间隔：

我们希望函数间隔越大越好，即:

if y(i)=1, want wTx(i)+b≫0 ,

if y(i)=−1, want wTx(i)+b≪0.

并且有，若 y(i) (wTx(i)+b) > 0 则样本(x(i),y(i))分类正确

几何间隔：

是为了解决函数间隔成倍增长的问题，几何间隔就是点到直线的举例

函数间隔

定义 7.2（函数间隔） 对于给定的训练数据集 T 和超平面 (w,b)，定义超平面 (w,b) 关于样本点 (x_i,y_i) 的函数间隔为

当label yi 是正例时(yi = 1)，说明wx+b>=0,我们也希望wx+b 非常大(远离分类超平面)，也就是确信度越高，反之越小，因此函数间隔反映了确信度

$$\hat{\gamma}_i = y_i(w \cdot x_i + b) \tag{7.3}$$

定义超平面 (w,b) 关于训练数据集 T 的函数间隔为超平面 (w,b) 关于 T 中所有样本点 (x_i,y_i) 的函数间隔之最小值，即

对于全局来说 函数间隔就是所有样本点中最小的那个
svm的目的就是让这个最小间隔尽量大

$$\hat{\gamma} = \min_{i=1,\cdots,N} \hat{\gamma}_i \tag{7.4}$$

几何间隔

定义 7.3（几何间隔） 对于给定的训练数据集 T 和超平面 (w,b)，定义超平面 (w,b) 关于样本点 (x_i,y_i) 的几何间隔为

$$\gamma_i = y_i\left(\frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|}\right) \tag{7.5}$$

定义超平面 (w,b) 关于训练数据集 T 的几何间隔为超平面 (w,b) 关于 T 中所有样本点 (x_i,y_i) 的几何间隔之最小值，即

$$\gamma = \min_{i=1,\cdots,N} \gamma_i \tag{7.6}$$

函数间隔与几何间隔的关系

从函数间隔和几何间隔的定义（式 (7.3)~式 (7.6)）可知，函数间隔和几何间隔有下面的关系：

$$\gamma_i = \frac{\hat{\gamma}_i}{\|w\|} \tag{7.7}$$

针对某一个样本点

分子是函数间隔

$$\gamma = \frac{\hat{\gamma}}{\|w\|} \tag{7.8}$$

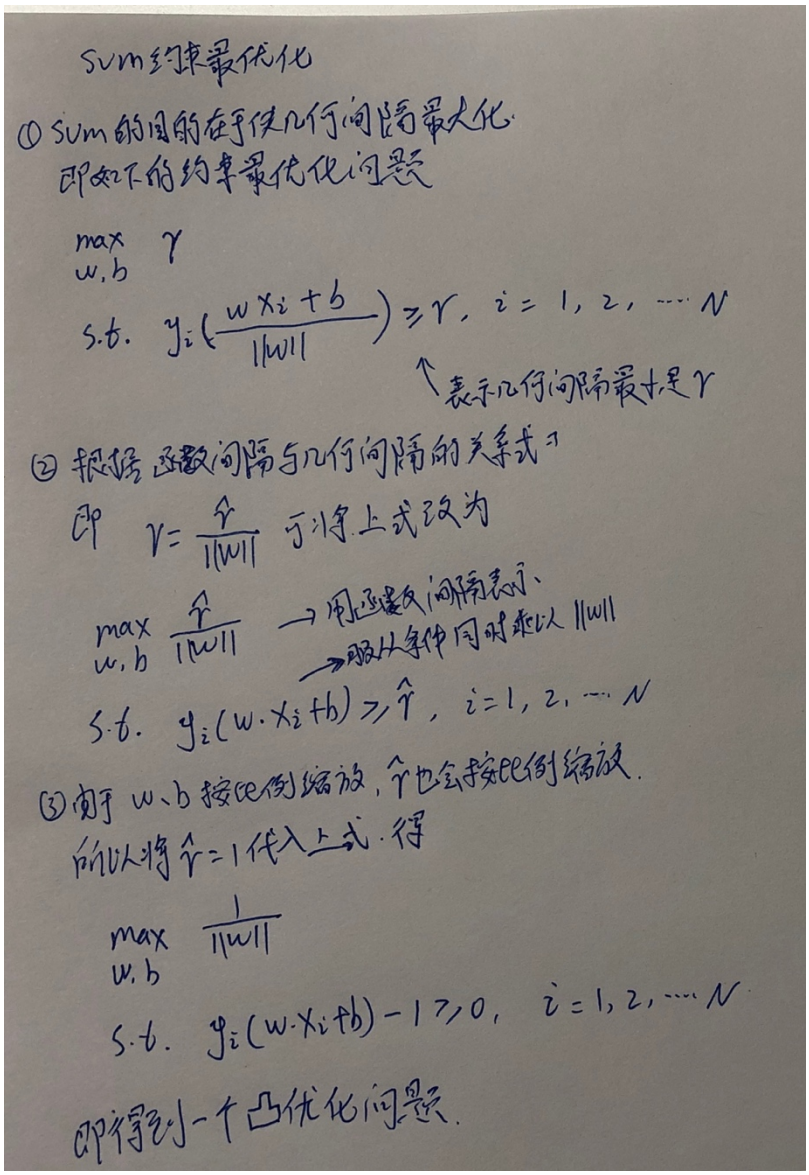
针对全局

2.3 间隔最大化

svm 的基本思想就是求解正确划分训练数据集，并且几何间隔最大的分离超平面

间隔最大化的直观解释是：对训练数据集找到几何间隔最大的超平面意味着以充分大的确信度对训练数据进行分类。也就是说，不仅将正负实例点分开，而且对最难分的实例点（离超平面最近的点）也有足够大的确信度将它们分开。这样的超平面应该对未知的新实例有很好的分类预测能力。

2.1.1. 凸优化的由来



2.4 使用

sklearn 中的 svm 使用

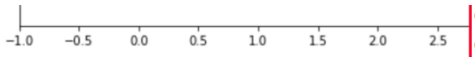
导入：from sklearn.svm import SVC

参数：sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma=0.0,

coef0=0.0, shrinking=True, probability=False, tol=0.001,

cache_size=200, class_weight=None, verbose=False, max_iter=-1,

random_state=None)



C越大表示支撑线距离超半面越小
C越大代表分类越强硬，泛化能力低
C越小代表可以容忍一些分类错误的出现
一般可以尝试0.01 0.1 100

- C越大，分类越严格，不能有错误，有过拟合的嫌疑
- C越小，意味着有更大的错误容忍度，刚刚好
- C越大分类效果越好相应的泛化能力越低，C越小，我们的决策边界更大一些，即在训练时容忍一些样本的误差，拿一些边界更宽的样本作为SVM

- C: float参数 默认值为1.0
- 错误项的惩罚系数。C越大，即对分错样本的惩罚程度越大，因此在训练样本中准确率越高，但是泛化能力降低，也就是对测试数据的分类准确率降低。相反，减小C的话，容许训练样本中有一些误分类错误样本，泛化能力强。对于训练样本带有噪声的情况，一般采用后者，把训练样本集中错误分类的样本作为噪声。
- kernel: str参数 默认为'rbf'
- 算法中采用的核函数类型，可选参数有：
  - 'linear':线性核函数
  - 'poly'：多项式核函数
  - 'rbf'：径像核函数/高斯核
  - 'sigmod':sigmod核函数
  - 'precomputed':核矩阵
- precomputed表示自己提前计算好核函数矩阵，这时候算法内部就不再用核函数去计算核矩阵，而是直接用你给的核矩阵。核矩阵为如下形式：
  - 还有一点需要说明，除了上面限定的核函数外，还可以给出自己定义的核函数，其实内部就是用你自己定义的核函数来计算核矩阵。
- degree: int型参数 默认为3

这个参数只对多项式核函数有用，是指多项式核函数的阶数n，如果给的核函数参数是其他核函数，则会自动忽略该参数。

- gamma: float参数 默认为auto，核函数系数，只对'rbf','poly','sigmod'有效。
  - 如果gamma为auto，代表其值为样本特征数的倒数，即1/n_features.
- coef0: float参数 默认为0.0，核函数中的独立项，只有对'poly'和'sigmod'核函数有用，是指其中的参数c
- probability： bool参数 默认为False，是否启用概率估计。 这必须在调用fit()之前启用，并且会fit()方法速度变慢。
- shrinking： bool参数 默认为True，是否采用启发式收缩方式
- tol: float参数 默认为1e^-3， svm停止训练的误差精度
- cache_size： float参数 默认为200，指定训练所需要的内存，以MB为单位，默认为200MB。
- class_weight： 字典类型或者'balance'字符串。默认为None，给每个类别分别设置不同的惩罚参数C，如果没有给，则会给所有类别都给C=1，即前面参数指出的参数C.如果给定参数'balance'，则使用y的值自动调整与输入数据中的类频率成反比的权重。
- verbose： bool参数 默认为False，是否启用详细输出。 此设置利用libsvm中的每个进程运行时设置，如果启用，可能无法在多线程上下文中正常工作。一般情况都设为False，不用管它。
- max_iter： int参数 默认为-1，最大迭代次数，如果为-1，表示不限制
- random_state： int型参数 默认为None，伪随机数发生器的种子,在混洗数据时用于概率估计。