## SVM开源库LIBSVM和LIBLINEAR的总体理解

### 1.基本资料

1.Libsvm与Liblinear区别，简单源码剖析。

[http://blog.csdn.net/zhzhl202/article/details/7438160](http://blog.csdn.net/zhzhl202/article/details/7438160" \t "http://blog.csdn.net/app_12062011/article/details/_blank)

[http://blog.csdn.net/zhzhl202/article/details/7438313](http://blog.csdn.net/zhzhl202/article/details/7438313" \t "http://blog.csdn.net/app_12062011/article/details/_blank)  
 Libsvm是一个整合了支持向量机(C-SVC, nu-SVC)、回归、分布估计(one-class SVM)的软件。并且支持多类别的分类。

LIblinear，主要专门为百万级别的数据和特征实现的线性分类器。

他们两个都是用来做分类的，相对来说Libsvm应用的范围较广， 而Liblinear主要用于处理大数据量的训练过程。在什么样的情况下，该选择Liblinear而不是Libsvm呢？作者给出几点建议：  
 1. 当你面对海量的数据时，这里的海量通常是百万级别以上。海量数据分为两个层次：样本数量和特征的数量。  
 2.使用线性和非线性映射训练模型得到相近的效果。  
 3.对模型训练的时间效率要求较高。

在这类情况下，建议你使用Liblinear，而不是libsvm

2.Liblinear使用，Java版本

[http://www.cnblogs.com/tec-vegetables/p/4046437.html](http://www.cnblogs.com/tec-vegetables/p/4046437.html" \t "http://blog.csdn.net/app_12062011/article/details/_blank)

3.Liblinear使用，官方翻译。

[http://blog.csdn.net/zouxy09/article/details/10947323/](http://blog.csdn.net/zouxy09/article/details/10947323/" \t "http://blog.csdn.net/app_12062011/article/details/_blank)

[http://blog.csdn.net/zouxy09/article/details/10947411](http://blog.csdn.net/zouxy09/article/details/10947411" \t "http://blog.csdn.net/app_12062011/article/details/_blank)

4.这里有篇文章，写的蛮好。转自：[http://blog.chinaunix.net/uid-20761674-id-4840097.html](http://blog.chinaunix.net/uid-20761674-id-4840097.html" \t "http://blog.csdn.net/app_12062011/article/details/_blank)

### 2.LIBSVM与LIBLINEAR的渊源

在过去的十几年里，支持向量机（Support Vector Machines）应该算得上是机器学习领域影响力最大的算法了。而在SVM算法的各种实现工具中，由国立台湾大学林智仁老师开发的工具包LIBSVM，又无疑是影响力最大的。2011年LIBSVM的系统介绍论文“LIBSVM: a library for support vector machines”发表在了期刊ACM TIST(ACM Transactions on Intelligent Systems and Technology)上。2011年的时候，这个期刊的影响因子还不到1，但到了2014年，它的影响因子居然达到了9.39，把TPAMI都甩出了一大截。这其中贡献最大的当然是关于LIBSVM的这篇论文，在google scholar上，这篇文章的引用量居然已经接近20000。

仔细想来其实也并不奇怪，各种研究工作，只要涉及到分类，大部分都会用到SVM算法或者与SVM的算法进行对比，而此时LIBSVM往往是首选工具。其实不但是学术界，在工业界LIBSVM也有非常广泛的应用。这一方面得益于算法实现的稳定与高效，另一方面也是因为LIBSVM提供了丰富的接口与灵活的使用方式。一些非常有名的机器学习工具，如**基于java的Weka和基于python的scikit-learn，其提供的SVM算法在底层也是基于LIBSVM的实现。**

简单来说，LIBSVM是一套完整的SVM模型实现。用户可以在LIBSVM中使用和核函数来训练非线性的分类器，当然也能使用更基础的线性SVM。而LIBLINEAR是一个针对线性分类场景而设计的工具包，除了支持线性的SVM外，还可以支持线性的Logistic Regression等模型，但是无法通过定义核函数方式实现非线性的分类器。**由于支持核函数的扩展，LIBSVM理论上具有比LIBLINEAR更强的分类能力，能够处理更为复杂的问题。**但是，很多人因此就只使用LIBSVM，甚至最简单的线性分类器都是用LIBSVM来训练和预测，这也是不可取的。因为LIBLINEAR设计的初衷就是为了提高线性分类的效率，其优化算法与LIBSVM中的优化算法有着根本的区别。虽然在进行线性分类时LIBSVM和LIBLINEAR都可以达到类似的结果，但是LIBLINEAR无论是在训练上还是在预测上，都比LIBSVM高效得多。此外，**受限于算法，LIBSVM往往在样本量过万之后速度就比较慢了，如果样本量再上升一个数量级，那么通常的机器已经无法处理了**。但使用LIBLINEAR，则完全不需要有这方面的担忧，即便百万千万级别的数据，LIBLINEAR也可以轻松搞定，因为LIBLINEAR本身就是为了解决较大规模样本的模型训练而设计的。

不解的是，LIBSVM早在2000年就已经发布了，而LIBLINEAR直到2007年才发布第一个版本。根据常识，应该是先有一个简单的工具，然后再逐渐完善，但是功能更加强大的LIBSVM却早于LIBLINEAR很久发布。要回答这个问题，还得从机器学习以及SVM的历史说起。

早期的机器学习分类算法可以追溯到Perception（感知机）。Perception的基本思想和Logistic Regression类似，只不过是用在线学习的方法训练出一个线性分类器。很多问题都非常复杂，比如图像或者语音的识别。但是另一方面，受限于当时获取数据与计算存储能力的限制，这些数据集的规模通常非常的小，有的只有几千甚至几百个样本。对于这样相对复杂的问题，可以想到将特征直接用简单的线性分类器进行分类，肯定不会取得太好的效果。这个时候，机器学习领域一个里程碑Multi-Layer Neural Networks（多层神经网络）出现了。多层神经网络引入了Hidden Layer （隐含层），模型的表达能力大大增强，可以训练出各种复杂的分类器。然而神经网络也有一个致命的弱点，由于模型本身的局限性，非常容易过拟合，尤其是在训练样本量较少的情况下。

而这时SVM应运而生，完美的解决了这个问题。**一方面，SVM的目标函数是一个凸函数，可以保证得到问题的全局最优解，避免了神经网络优化频繁陷入局部最优的困扰。另一方面，SVM的背后有一套结构化风险最小化的理论，给定了训练样本和训练参数，是可以从理论上计算出模型在真实数据上误差的bounds。在SVM中，通过对参数的调节和样本量的选择，可以在模型方差与训练误差之间方便的做出权衡。此外，SVM可以定义不同的核函数来构造非线性分类器，可以得到与神经网络方法大体相当的分类能力，从而适应不同的问题。**因此，在上个世纪末到这个是基础，SVM横扫了各种分类的应用场景，成为了当时最炙手可热的机器学习算法。

然而，SVM也存在局限性。首先，**基于核函数的SVM求解相对比较复杂，需要存储一个稠密的样本间的Kernel矩阵，当样本量很大时，存储量相当可观。**而到目前为止，一直没有一个非常有效的并行SVM训练方法能够从根本上提升SVM模型的训练。在十几年前，样本量最多只是上万级别的时候，这个问题并不显得重要。但是在十多年后，随着互联网的爆炸式发展，随便一个模型的训练样本量都可能数以亿计，这时SVM在大数据训练上的不足就凸显无疑了。SVM之所以效果好，主要是得益于非线性核函数的引入。但是新的问题不断出现，而这些问题又涉及到的不同领域知识与业务场景，很多时候仅仅依靠常见的几种kernel函数并不能解决问题。但是SVM本身过分依赖于核函数，而核函数又存在着很多的限制，其灵活性当然不如人工的特征构造方法。另一方面，随着数据量的不断增加，即使这些样不能直接被标注用于模型的训练，但是可以**很多机器学习方法可以从大量的样本中进行特征的自动学习**。比如早年的流形学习，还有文本上的主题模型，图像上的稀疏编码与字典学习等。**通过这些非线性的方法学习的样本特征，往往已经是样本的高层语义表达，有数据充足的情况下，只需要使用较为简单的线性分类器，就可以达到比较好的效果。**这时的**主要矛盾变成了分类器必须有能力处理足够大量的样本**，而在方法上，可以是简单地现行方法。也就是在这个时候，LIBLINEAR应运而生。

LIBSVM的整体框架都是针对训练kernel SVM模型来训练的。但是如果只是需要训练一个线性的SVM模型，那么算法可以简单的多，也可以高效的多。因此LIBLINEAR在保持基本接口和调用方式一致的情况下，采用了新的训练算法，支持了线性SVM和Logistic Regression的训练。LIBSVM和LIBLINEAR的作者林智仁老师在后来的很多演讲中，都在大力的推广LIBLINEAR，并且给出了很多实际的例子证明，**人工构造特征+线性模型的方式可以达到甚至超过kernel SVM的表现，同时大大降低训练的时间和消耗的资源。**

其实就在最近几年，情况又有了新的变化。**人工构造特征+线性分类器的方式在很多问题上又遇到了瓶颈。与此同时，一方面可供使用数据量更大了，另一方面，计算机的计算能力又有了突飞猛进的增长。此时曾经被SVM狠狠压在地上的神经网络又重新焕发了生机。**与SVM相比，神经网络模型的优势在于可以通过控制模型的层数和每一层函数的类型，设计出各种灵活的分类器。同时神经网络的优化算法比kernel SVM更适合并行化。当时影响神经网络发展的主要问题是计算资源的限制和样本量少引起的过拟合。但现在这两项限制都几乎不存在了。基于GPU的并行计算技术现在已经比较成熟，可以支持高速的并行计算。而造成过拟合的原因从根本上说，是因为训练样本的比真实样本数少的太多，不能够反应真实的数据情况。但是如果把我们拥有的所有样本都作为训练样本，其实机会已经就是真实的样本集了，因此过拟合的事实基础就不存在了。虽然神经网络在理论上还有缺陷，但是通过计算能力和数据的增加，这些缺陷已经不是问题。正是因为上面的原因，这些年机器学习的热点又重新回到了神经网络。

仔细看每一次算法的革新，其实真正的推动力都是具体的问题需求与当时的技术条件。因此，脱离具体的应用场景去单纯比较算法的优劣并没有太多意义。对于使用者来说，最适合这个问题场景的算法就是最好的算法。具体到LIBSVM和LIBLINEAR，我尝试总结下面几个原则：

**1.凡是确定使用线性分类器的场景，一定使用LIBLINEAR而不是LIBSVM   
 2.如果样本量较大，比如达到10万以上的规模，这时LIBSVM已经很难处理了。如果线性分类器的效果实在不好，只能采用人工构造特征+LIBLINEAR的方式，或者采用其他的分类器，如神经网络，随机森林等。  
 3.对于高维稀疏数据，典型的如文本的向量空间表示，一般都采用线性的分类器。   
 4.对于样本量和维度都不算太大的问题，且没有对预测的效率有过分的需求，都可以用LIBSVM尝试一下kernel SVM的分类器，很多情况下用kernel SVM比直接用libear SVM还是能达到更高的精度。**

### 3.两种模型解决的问题与算法

**LIBSVM**

LIBSVM支持的5种模型。其中模型0和1对应的都是SVM的分类模型，2对应的是one-class分类器，也就是只需要标注一个标签，模型3和4对应的是SVM的回归模型。

首先来看最基础的C-SVC模型。

**当模型使用linear kernel，也就是φ(x)=x时，上面的问题一个标准的二次凸优化问题，可以比较方便的对每一个变量进行求导。求解这样的问题是有很多快速的优化方法的，这些方法在LIBLINEAR中都有应用。**但是如果是引入kernel的SVM，情况就大不一样了。因为**很多时候我们既不能得到核函数的具体形式，又无法得到特征在核空间中新的表达。**这个时候，之前用在线性SVM上的的求解思路就完全不work了。**为了解决这个问题，就必须采用标准的SVM求解思路，首先把原问题转化为对偶问题。**

通过对偶变化，上面的目标函数变成了一个关于变量α的二次型。很显然，上面目标函数中最重要的常亮是矩阵Q，既训练样本的Kernel Matrix，满足Qi.j=**φ**(xi)T**φ**(xj)。先看好的一方面，根据核函数的定义，能够保证Q是一个正定的矩阵。也就是说，上面的目标函数还是一个凸函数，优化收敛后能保证得到的解是全局最优解， 这也是SVM的重要优势之一。但是问题也随之而来，**使用常用的核函数，只要任意给出两个向量，总是能够计算出一个非0的距离。这也就意味着矩阵Q将会是一个非常稠密的矩阵，如果训练样本足够多，那么矩阵Q的存储和计算将成为很大的问题，这也是SVM的优化算法中的最大挑战。**

由于矩阵Q过大，所以想一次性优化整个α是比较困难的。所以常用的方法都是先把Q大卸八块，每次选择一部分的Q，然后update与这部分Q相关的α的值。这其中最著名的算法就是1998由John C. Platt提出的**SMO算法**，而LIBSVM的优化过程也是基于SMO算法进行的。SMO算法的每一步迭代都选择最小的优化单元，也就是固定其他的α，只挑选两个α的值进行优化。之所以不选择一个，是因为有yTα=0的约束，至少选择两个α的坐标才有可能进行更新。本文主要目的是介绍LIBSVM，所以就不详细讨论SMO的细节了。至于LIBSVM中的具体算法实现，在LIBSVM的官方论文中介绍的很详细，这里总结部分关键问题：

1.Working Set，也就是需要优化的α部分的选取

2.迭代停止条件的设置

3.α的更新算法，也就是每一步子问题的求解方法

4.Shrinking，即移除一些已经满足条件的α，加快收敛速度

5.Cache，当Q矩阵过大时，需要对矩阵进行缓存。

上面的每个问题，处理起来都不简单。作为使用者，或许也没有必要深谙里面的所有细节。我觉得最需要认识的两个问题是：**1) SVM的目标函数看起来好像是一个标准的优化问题，但实际求解却要复杂得多。为了提高求解的速度，既要做算法上的优化，也需要做工程上的改进。如果只是简简单单按照教科书的方法，甚至直接调用一些优化的工具包来实现的SVM算法，最多也就算个demo。要能够真正写一个高效稳定、能处理大规模数据的SVM工具还是非常不容易的。**所以用LIBSVM还是比自己实现算法要简单靠谱不少。**2)SVM的求解之所以要优化，就是因为这个问题本身计算和存储比较麻烦。所以虽然做了这么多的优化，整个算法求解的效率仍然较低。所以我们在使用时还要注意各种程序的细节，提高运行的效率。另外，样本量过大时，有时候为了充分利用数据，也不得不忍痛割爱，放弃kernel的使用。**

除了标准的C-SVM，LIBSVM也提供了对其他一些SVM方法的支持。**其中ν-SVM与C-SVM的算法与应用场景基本是相同的，唯一的区别是原本的参数C变成了参数ν。C-SVM中参数C调整范围在[0,+∞)，而ν-SVM中与之对应的参数ν的调整范围变成了 (0,1]。**这样的设置使得ν-SVM更具解释性，有时在参数设置上也能提供一定的方便。但ν-SVM与C-SVM并不存在本质上的差别，通过参数的调节，两者可以达到完全相同的效果。所以在使用LIBSVM处理分类问题是，选择上面任何一种方法都是OK的，只需要遵循自己的习惯就好了。

**One-Class SVM也是LIBSVM所支持的一种分类方法。顾名思义，使用One Class时，只需要提供一类样本，算法会学习一个尽量小的超球面包裹所有的训练样本。One-Class SVM看起来很有诱惑力，因为我们经常会遇到有一类样本而需要学习分类器的情况。但事实上，一方面很多时候我们得到的正样本在采样过程中存在很大的偏差，导致学习出的One Class分类器不一定考虑到了所有正样本的情形；另一方面，大部分问题还是存在很多构造人工负样本的办法。根据我的经验，采用普通的SVM效果通常还是会好过One-Class SVM，而One-Class SVM在真实场景中的使用也并算不上多。因此在使用这个方法前也需要对问题进行更深入的研究。**

最后，LIBSVM也支持基于SVM的回归模型，即SVR。与分类模型类似，SVR也分为C-SVR和ν-SVR。SVR的目标函数与SVM的分类模型稍有区别。由于回归问题预测值与目标值的偏差可大可小，因此SVR使用了两个slack variable用来刻画预测的误差边界。虽然存在这样的差别，但是两者的基本思路和优化算法与还是基本一致的。

在LIBSVM的实现中，上面五种模型，即C-SVM，ν-SVM，One-class SVM，C-SVR，ν-SVR，最终都可以转化为一个更通用的优化框架，然后用同样的策略进行求解，这也是LIBSVM所实现的主要功能。**在实际使用中，最常用到的方法还是C-SVM，这是最传统的SVM分类模型。**

**LIBLINEAR**

LIBLINEAR是在LIBSVM流行多年后才开发的，要解决的问题本质上也比LIBSVM更简单，其优势主要在于效率与scalablility。之所以存在这样的优势，是因为线性SVM的求解要比kernel SVM简单许多。

还从上面的对偶问题说起，**之前SVM的求解很大程度上受到yTα=0的困扰，因此每次必须选择一组 α进行优化。如果对这一约束项追根述源，可以发现这一项是通过令模型的常数项b导数为0而得到的。而在线性模型中，我们可以通过一个简单地trick，令x=[x,1]和w=[w,b]，这样，在模型中的常数项就不存在了。当然，这样的trick只能在线性模型中才适用。**

**这个时候，就可以每次只选择一个αi进行优化，每一轮遍历α的所有维度，多轮迭代，直至最后收敛。这样的优化算法叫做coordinate descent（坐标下降法）。利用线性函数的特殊性，直接根据α就可以计算出w的向量表示，于是大大提高了算法的效率。**

大量的监督学习模型都可以写成loss function + regularizer的形式，而参数C则控制了两者在最终损失函数中所占的比重。不同loss function与regularizer的选取以及两者之间的平衡，是机器学习的最重要主题之一。

**对于上面的问题，有很多成熟的算法可以进行模型的求解，比如最速梯度法，牛顿法等，对于样本量较大时，也可以采用随机梯度的方法进行训练。** 一般来说，由于考虑了二阶导数，牛顿法本身的优化效率要高于只考虑一阶导数的最速梯度法。**但由于牛顿法本身在计算量和收敛性上存在很多局限性，**所以很少直接使用，而是在牛顿法思想基础上进行一定的改进。其中普遍使用的算法有BFGS和L-BFGS等。**具体到liblinear软件包，作者采用的是Trust Region Newton (TRON) method对模型对传统牛顿法进行了改进，该方法被证明比L-BFGS训练更加高效。**

LIBLINEAR中实现了基于TRON方法的L-2 SVM和Logistical Regression模型训练。其中的L2-loss SVM是标准SVM的变种，loss function变成了：

?(yi,wTxi)=(max(0,1?yiwTxi))2

从实际效果来说，L2-loss SVM与标准的L1-loss SVM并没有太大的区别。但是在计算上，前者的求导形式更加简单，便于梯度的计算与优化。**LIBLINEAR并没有实现Trust Region Newton法的标准L1-loss SVM实现，一方面是因为直接对hinge loss求导需要分段讨论比较复杂，另一方面L2-loss SVM基本可以直接替代L1-loss SVM。不过在其他的一些软件包中，如SVMLIN中，则实现了L1-loss SVM的原问题求解，但使用的优化算法是L-BGFS而不是TRON。**

### 4.总结

简单总结一下不同算法的应用场景吧：

1.所有线性问题都是用LIBLINEAR，而不要使用LIBSVM。

2.LIBSVM中的不同算法，如C-SVM和nu-SVM在模型和求解上并没有本质的区别，只是做了一个参数的变换，所以选择自己习惯的就好。

3.LIBLINEAR的优化算法主要分为两大类，即求解原问题(primal problem)和对偶问题(dual problem)。**求解原问题使用的是TRON的优化算法，对偶问题使用的是Coordinate Descent优化算法。**总的来说，两个算法的优化效率都较高，但还是有各自更加擅长的场景。**对于样本量不大，但是维度特别高的场景，如文本分类，更适合对偶问题求解，因为由于样本量小，计算出来的Kernel Matrix也不大，后面的优化也比较方便。**而如果求解原问题，则求导的过程中要频繁对高维的特征矩阵进行计算，如果特征比较稀疏的话，那么就会多做很多无意义的计算，影响优化的效率。相反，**当样本数非常多，而特征维度不高时，如果采用求解对偶问题，则由于Kernel Matrix过大，求解并不方便。反倒是求解原问题更加容易**。

多分类问题

LIBSVM和LIBLINEAR都支持多分类（Multi-class classification）问题。所谓多分类问题，就是说每一个样本的类别标签可以超过2个，但是最终预测的结果只能是一个类别。比如经典的手写数字识别问题，输入是一幅图像，最后输出的是0-9这十个数字中的某一个。

LIBSVM与LIBLINEAR但实现方式却完全不同。**LIBSVM采取的one vs one的策略，也就是所有的分类两两之间都要训练一个分类器。这样一来，如果存在k个class，理论上就需要训练 k(k-1)/2个分类器。**实际上，libsvm在这一步也进行了一定的优化，利用已有分类的关系，减少分类器的个数。尽管如此，LIBSVM在多分类问题上还是要多次训练分类器。但是，考虑到前面说的LIBSVM的优化方法，随着样本数量的增加，训练的复杂度会非线性的增加。而**通过1VS1的策略，可以保证每一个子分类问题的样本量不至于太多，其实反倒是方便了整个模型的训练。**

**而LIBLINEAR则采取了另一种训练策略，即one vs all。每一个class对应一个分类器，副样本就是其他类别的所有样本。由于LIBLINEAR能够和需要处理的训练规模比LIBSVM大得多，因此这种方式要比one vs one更加高效。此外，LIBLINEAR还实现了基于Crammer and Singer方法的SVM多分类算法，在一个统一的目标函数中学习每一个class对应的分类器。**

**输出文件**

一般来说，我们**使用LIBLINEAR或者LIBSVM，可以直接调用系统的训练与预测函数，不需要直接去接触训练得到的模型文件。**但有时候我们也可能需要在自己的平台实现预测的算法，这时候就不可避免的要对模型文件进行解析。

由于LIBLINEAR与LIBSVM的训练模型不同，因此他们对应的模型文件格式也不同。LIBLINEAR训练结果的格式相对简单，例如：

1 solver\_type L2R\_L2LOSS\_SVC\_DUAL 2 nr\_class 3 3 label 0 1 2 4 nr\_feature 5 5 bias -1 6 w 7 -0.4021097293855418 0.1002472498884907 -0.1619908595357437 8 0.008699468444669581 0.2310109611908343 -0.2295723940247394 9 -0.6814324057724231 0.4263611607497726 -0.4190714505083906 10 -0.1505088594898125 0.2709227166451816 -0.1929294695905781 11 2.14656708009991 -0.007495770268046003 -0.1880325536062815

上面的solver\_type表示求解算法，w以下表示求解得到的模型权重。其中每一列对应一个class的分类器，而每一行对应特征的一个维度。其中nr\_class表示求解的个数，nr\_feature表示特征的维度，bias表示模型的bias，可以人工设置权重。这里容易产生误解的是label这个字段，表示的是每一个用户训练文件中label对应w的列数。比如在上面的模型中，用户指定编号为0的分类器对应w的第一列。但是上面的对应关系并不是一定存在的，比如在二分类场景中，用将整样本标为1，负样本标为0，但在模型训练中，LIBLINEAR会按照自己的编号系统进行训练，因而有可能出现负样本在前，正样本在后的情形。这时候，就必须要根据label 1 0将LIBLIENAR内部的编号体系与真实的用户标签进行对应。当然，后来LIBLINEAR和LIBSVM做了一些优化，在二分类时，如果正负样本标签分别是-1和+1，那么可以始终保证正样本出现在w的第一列。但是这个机制也不是完全靠谱，比如说在LIBLINEAR的spark实现代码中，就没有实现这个特性，曾经把我整的很惨。因此在这里还是需要十分注意。

LIBSVM的训练结果格式就更复杂一些，例如：

1 kernel\_type rbf 2 gamma 0.0769231 3 nr\_class 3 4 total\_sv 140 5 rho -1.04496 0.315784 1.03037 6 label 1 0 -1 7 nr\_sv 2 2 1 8 SV 9 0 1 1:0.583333 2:-1 3:0.333333 4:-0.603774 5:1 6:-1 7:1 8:0.358779 9:-1 10:-0.483871 12:-1 13:1 10 0 0.6416468628860974 1:0.125 2:1 3:0.333333 4:-0.320755 5:-0.406393 6:1 7:1 8:0.0839695 9:1 10:-0.806452 12:-0.333333 13:0.5 11 0 1 1:0.333333 2:1 3:-1 4:-0.245283 5:-0.506849 6:-1 7:-1 8:0.129771 9:-1 10:-0.16129 12:0.333333 13:-1 12 0.2685466895842373 0 1:0.583333 2:1 3:1 4:-0.509434 5:-0.52968 6:-1 7:1 8:-0.114504 9:1 10:-0.16129 12:0.333333 13:1 13 0 1 1:0.208333 2:1 3:0.333333 4:-0.660377 5:-0.525114 6:-1 7:1 8:0.435115 9:-1 10:-0.193548 12:-0.333333 13:1

上面参数的意义都比较直接，需要注意的是SV后面就是训练出的模型参数，以支持向量的方式进行存储。nr\_sv给出了每一个支持向量所对应的模型，比如“2 2 1”就表示前两行是标签为1类的支持向量，其后面两行是标签为0类的支持向量，最后一行是标签为-1类的支持向量。而具体每一行支持向量，在上面的模型中，由于存在三类，所以每一个支持向量有可能都会存在于两个分类器中，所以前两列的数分别对应了对剩下两个分类作为支持向量时候的α值，后面才是真正的支持向量。

## LIBSVM和LIBLINEAR库剖析

### 逻辑回归logistic regression

Logistic回归的名字让人容易误解，它虽然叫“回归”但实际上是一种用于分类问题的算法。由于实现简单、计算速度快，logistic回归被广泛应用于大规模数据集的分类问题，如广告点击率预估、文本分类。

Logistic回归可以直接输出样本属于某一类的概率。对于二分类问题，给定参数和样本的特征向量，无论是正样本还是负样本，该样本属于每个类的概率可以统一写成如下形式：



Logistic回归只能用于二分类问题，我们将它进行推广，得到处理多类分类问题的softmax回归，也称为多类logistic回归。主要思路是：

softmax回归按照下面的公式估计一个样本属于每一类的概率：



将一般的logistic似然函数，推广为多类的损失函数，相当于把多个类的损失函数做累加（每一个类的损失函数都有待优化的一组权重），然后优化推广得到的损失函数使其最小（最后得到所有类的各组权重值）。

在预测的时候，将待预测样本输入到每个类的预测模型中，取概率最大的哪一个模型作为分类结果。

### 线性可分支持向量机

支持向量机的目标是寻找一个分类超平面，该平面不仅能正确的分类每一个样本，并且要离每一类距离超平面最近的样本的距离尽可能远。





### 线性不可分/软间隔支持向量机

上面的线性可分支持向量机不具有太多的实用价值，因为在现实问题中，样本往往都不是线性可分的。为了处理这个问题，当线性不可分时，通过加上松弛变量和惩罚因子对错误分类的样本进行惩罚，可以得到如下最优化问题：



### 核向量支持向量机

即使是加入了松弛变量和惩罚因子，支持向量机还是一个线性分类器，只不过允许错分样本的存在。本节要介绍的核映射使得支持向量机成为非线性分类器，其决策边界不再是线性的超平面，而可以是形状非常复杂的曲面。

当样本线性不可分时，可以对特征向量做映射（一般是非线性映射），将向量转化到更高维的空间，使得在该空间中是线性可分的，这种方法在机器学习中被称为核技巧[4]。对于特征向量，核映射将其变换到更高维的空间。这种映射为：



在对偶问题的表述中，我们出现的都是两个样本向量之间的内积，因此有：



直接计算效率太低，而且不容易构造。如果映射函数选取得当，存在函数，使得下面等式成立：



### SMO求解算法

前面我们给出了支持向量机的对偶问题，但并没有说明对偶问题怎么求解。当训练样本的数量很大的时候，矩阵的规模很大，求解二次规划问题的经典算法将会遇到性能问题。支持向量机最优化问题的高效求解算法-经典的SMO算法。

SMO算法[5]由John C. Platt等人在1998年提出，这是一种求解支持向量机对偶问题的高效方法。SMO算法的思路是，每次在向量中挑出两个分量进行优化，而让其他分量固定（这样才能保证满足等式约束条件），这是一种分治法的思想。

每次固定两个分量优化时，求解算法得到是在可行域里的最小值，因此每次求解更新和的值之后，都能保证目标函数值小于或者等于初始值，即函数值下降。

### 优化变量的选择（违反KKT原则）

选择方法是在训练样本中选取违反KKT条件最严重的那个样本。

在最优点处，训练样本是否满足KKT条件的判据是：







其中定义为：

。



### 多类分类问题

前面讲述的支持向量机都只能解决二分类问题，事实上，它可以用于多分类问题。对于多分类问题，有以下几种解决方案：

1对剩余方案。对于个类的分类问题，训练个分类器。在训练时，第个分类器的正样本是第类样本，负样本是除第类之外其他所有类型的样本，这个分类器的作用就是判断样本是否属于第类。在进行分类时，对于待预测样本，用每个分类器计算输出值，取输出值最大那个作为最终的预测结果。

1对1方案。如果有个类，训练个二分类器。训练时，将第类作为正样本，第类作为负样本，总共有种组合。每个分类器的作用是判断样本是属于第类还是第类。对样本进行分类时采用投票的方法，依次用每个二分类器进行预测，如果判定为第类，则类的投票数加1，得票最多的那个类作为最终的判定结果。

除了通过二分类器来构造多类分类器之外，还有其他一些思路，通过直接优化多类分类的目标函数得到多分类器[13]。

### LIBSVM开源库

libsvm实现了5种类型的支持向量机，用于分类和回归问题，它们分别是：

（-support vector classification），这就是我们前面讲述的用于分类问题的支持向量机，是源代码分析的重点。

（-support vector classification），用于分类问题，另外一种表述的支持向量机，不使用惩罚因子，而是使用了另一个变量替代它，不做重点介绍。

one class SVM，单类SVM，用于估计特征向量的分布，不做重点介绍。

（-support vector regression），用于回归问题，不做重点介绍。

（-support vector regression），用于回归问题，不做重点介绍。

libsvm对-SVC，-SVR和one-class SVM的求解采用了SMO算法，在工作集的选择上做了改进，利用了二阶导数信息加快算法收敛速度。

对于分类问题，libsvm支持多类分类和交叉验证，并且可以输出概率值，对于多类分类问题，采用的是1对1的方案。

### LIBLINEAR开源库

Liblinear实现了以下机器学习算法：

L2正则化的logistic回归（简称L2-LR），求解原问题与对偶问题

L1正则化的logistic回归（简称L1-LR），只求解原问题

L2正则化L1-loss的线性支持向量机，只求解对偶问题

L2正则化L2-loss的线性支持向量机，求解原问题与对偶问题

L1正则化L2-loss的线性支持向量机，只求解原问题

多类线性支持向量机，求解对偶问题

表12.1 liblinear求解的各种问题

|  |  |  |
| --- | --- | --- |
| 算法类型 | 求解的问题 | 求解算法 |
| L1正则化LR | 原问题 | 坐标下降法 |
| L2正则化LR | 原问题 | 可信域牛顿法 |
| L2正则化LR | 对偶问题 | 坐标下降法 |
| L1正则化L2损失函数SVC | 原问题 | 坐标下降法 |
| L2正则化L2损失函数SVC | 原问题 | 可信域牛顿法 |
| L2正则化L1损失函数SVC | 对偶问题 | 坐标下降法 |
| L2正则化L2损失函数SVC | 对偶问题 | 坐标下降法 |
| L1损失函数SVR | 对偶问题 | 坐标下降法 |
| L2损失函数SVR | 原问题 | 可信域牛顿法 |
| L2损失函数SVR | 对偶问题 | 坐标下降法 |
| 多类SVM | 对偶问题 | 坐标下降法 |

多类分类问题的线性支持向量机。在前一章中，libsvm解决多分类问题是通过二分类器实现的，在liblinear中直接定义并求解多类问题的损失函数。

对于多类分类问题，liblinear的支持向量机采用了一对剩余的方案。

### Solver的选择

liblinear支持多种solver模式，以下直接列举liblinear支持的几种典型solver模式对应的结构风险函数（结构风险函数由损失函数和正则化项/罚项组合而成，实际即为求解结构风险函数最小值的最优化问题），以方便说明和理解。

L2-regularizedL1-loss Support VectorClassification

IMG_256

L2-regularizedL2-loss Support Vector Classification

IMG_256

L1-regularizedL2-loss Support Vector Classification

IMG_256

L2-regularized Logistic Regression

IMG_256

L1-regularized Logistic Regression

IMG_256

Liblinear中同时支持线性svm和logisticregression，两者最大区别即在于损失函数（loss function）不同，损失函数是用来描述预测值f(X)与实际值Y之间差别的非负实值函数，记作L(Y, f(X))，即上述公式中的项。

另一个重要选择是正则化项。正则化项是为了降低模型复杂度，提高泛化能力，避免过拟合而引入的项。当数据维度很高/样本不多的情况下，模型参数很多，模型容易变得很复杂，表面上看虽然极好地通过了所有样本点，但实际却出现了很多过拟合，此时则通过引入L1/L2正则化项来解决。

一般情况下，L1即为1范数，为绝对值之和；L2即为2范数，就是通常意义上的模。L1会趋向于产生少量的特征，而其他的特征都是0，即实现所谓的稀疏，而L2会选择更多的特征，这些特征都会接近于0。

对于solver的选择，作者的建议是：一般情况下推荐使用线性svm，其训练速度快且效果与lr接近；一般情况下推荐使用L2正则化项，L1精度相对低且训练速度也会慢一些，除非想得到一个稀疏的模型（个人注：当特征数量非常大，稀疏模型对于减少在线预测计算量比较有帮助）。

### 原问题与对偶问题的选择

primal和dual分别对应于原问题和对偶问题的求解，**对结果是没有影响的，**但是对偶问题可能比较慢。作者有如下建议：对于L2正则-SVM，可以先尝试用dual求解，如果非常慢，则换用primal求解。

网上另一个可参考的建议是：对于样本量不大，但是维度特别高的场景，如文本分类，更适合对偶问题求解；相反，当样本数非常多，而特征维度不高时，如果采用求解对偶问题，则由于Kernel Matrix过大，求解并不方便。反倒是求解原问题更加容易。

### 训练数据归一化

作者是这样建议的：**在他们文档分类的应用中，归一化不但能大大减少训练时间，也能使得训练效果更好，因此我们选择对训练数据进行归一化。**同时在实践中，归一化使得我们能直接对比各特征的公式权重，直观地看出哪些特征比较重要。

### LIBSVM和LIBLINEAR求解算法的区别

**LIBSVM求解SVM问题：**

1）在libsvm中，支持向量机的最初问题原型（原问题和对偶问题）为：

 

可以看出，优化项为二次项，并带有约束条件。

2）引入松弛因子（原问题与对偶问题）为：

 

可以看出，优化项为二次项+惩罚项，并带有约束条件。

3）加入核函数为：



在LIBSVM中，几种支持向量机的**原问题都是求解一个带约束的二次优化问题**，它们的**对偶问题都是二次规划问题**，并且有相同的二次项系数矩阵。不同的只有一次项系数以及约束条件。

LIBSVM的求解算法主要是**SMO算法**；主要思路是利用KKT条件选择两个待优化参数（在libsvm中，工作集选择算法使用了二阶导数信息），固定其他参数，一直迭代收敛到最小；然后再选择两个待优化的参数，继续优化，直到满足条件。

整个训练算法的核心可以分为两部分，第一步选择工作集，第二部分对选定的工作集变量的子问题进行最优化求解。由于是一个密集矩阵，当问题的规模很大时，不能完全将存放在内存中，在实现时，采用了cache策略缓存矩阵。

**LIBLINEAR求解logistic回归分类问题：**

1）在liblinear中，logistic回归的原问题为：



为了防止过拟合，加入了正则化项：



其中二次项为正则化项，优化项为一次累加项。可以证明L2正则化的logistic回归的原问题是一个**凸优化问题，也是一个不带约束条件的优化问题**。常用的非线性优化方法如非线性共轭梯度法、拟牛顿法等。

2）上面问题的对偶问题为：



可以看出这个**对偶问题是一个带约束条件的优化问题**，**在liblinear中，求解该问题的方法是坐标下降法。**和其他最优化方法如共轭梯度法、拟牛顿法相比，坐标下降法有更快的速度，坐标下降法更适合大规模问题的求解。

坐标下降法的思路是每次迭代时更新部分变量的值，这比同时优化所有变量要简化很多。简单的说，就是只调整部分变量，将其他固定住不动，回忆一下求解支持向量机对偶问题的SMO算法，其实和上面的方法思想是一样的，都采用了分治的策略。

对于优化变量的选择，是依次优化每一个变量，上述算法的关键是求解子优化问题。

**3）L1正则化logistic回归求解如下不带约束的最优化问题：**



**在liblinear中，该问题的求解采用的也是坐标下降法。**

在上面的2）、3）两个问题中，坐标下降法的求解思路都是依次优化每一个变量，求解单个变量的子问题时，都是先显式求解或者用牛顿法求解，得到牛顿方向；然后进行直线搜索，得到合适的步长。

**LIBLINEAR求解线性SVM问题**

和libsvm不一样，在liblinear中，求解L2正则化的L2和L1损失函数的支持向量机对偶问题并没有采用SMO算法，而是都采用了坐标下降法。

如果每次只优化一个变量，即我们限定工作集只有一个元素，依次从集合中选择一个元素进行优化。

之所以应用的不是梯度法，是因为计算上面的梯度计算比较耗时，为此提出了高效的算法。

和前面的坐标下降法一样，依次优化每一个变量，优化时，采用了牛顿法和直线搜索。先计算出牛顿方向，然后寻找合适的步长。

总的来说，在LIBLINEAR中，logistic回归以及线性SVM的求解算法基本是可信域牛顿法和坐标下降法。

### 可调节的参数

简单做下总结参数与kernel的选择吧：

**\* 线性模型选择，选择LIBLINEAR，主要调节参数C，从小到大调节，如果再增加C对结果改变不大时，就可以不用进行测试下去了，选择较小的C就可以了。**

**\* kernel选择，如果需要使用kernel，对于一般问题，优先使用rbf kernel。LIBSVM提供的多项式和tanh核函数，都存在一些局限性，一般来说，rbf是使用方便性和模型效果都比较稳定的核函数了。 \* LIBSVM参数调节。如果使用rbf核，需要同时调节参数C和参数γ，问题要更复杂一些，最好的办法是自动遍历参数进行参数选择，比如利用grid.py。**

LIBSVM和LIBLINEAR工具包都包含很多需要调节的参数，参数的调节既需要足够的耐心，也有着很多的技巧。当然，还需要对参数本身的意义和对模型的影响了如指掌。下面主要讨论一些对模型影响较大的参数

**参数C**

参数C是在LIBLINEAR和LIBSVM的求解中都要用到的一个参数。前面说到的各种模型，可以写成统一的形式：

argminwΩ(?(w))+C∑i=1l?(yi,?(w)T?(xi))(1)

其中右边的一项是模型的损失项，其大小表明了分类器对样本的拟合程度。而左边的一项，则是人为加上的损失，与训练样本无关，被称作正则化项(Regularizer)，反映了对训练模型额外增加的一些约束。而**参数C则负责调整两者之间的权重。C越大，则要求模型能够更好地拟合训练样本数据（拟合更接近样本结果，但容易过拟合），反之，则要求模型更多的满足正则化项的约束（泛化能力越好）。**以LIBLINEAR为例，下面先讨论LIBLINEAR下?2norm的情况：

argminw∥w∥22+C∑i=1l?(yi,wTxi)(2)

之所以要增加正则化项，是因为在设计模型的时候，我们对于样本的质量以及模型的泛化能力没有充分的自信，认为在没有其他约束的情况下，训练得到的模型会因为过于迁就已有的样本数据而无法对新的数据达到同样的效果。在这个时候，就必须在模型中增加人类的一些经验知识。比如上面对?(w)增加?2norm的约束就是如此。

如果上面公式中的损失函数对应一个回归问题，那么这个问题就被称作Ridge Regression，中文叫做脊回归或者岭回归。

我们可以站在不同的角度来理解?2norm正则化项的意义。如果把学习分类函数中w看作是一个参数估计的问题，那么不带正则化项的目标函数对应的就是对w进行最大似然估计的问题。为了使w的估计更加接近真实的情况，我们可以根据经验对w制定一个先验分布。当我们假设w先验分布是一个多元高斯分布，且不同维度之间是没有关联的(即协方差矩阵非对角线元素为0)，而每一个维度特征的方差为某一固定制，那么推导出来的最大后验概率就是上面的带正则化项的目标函数。而C与w先验分布的方差相关。**C越大，就意味着正则化的效果偏弱，w的波动范围可以更大，先验的方差也更大；而C越小，则意味着正则化的效果更强，w的波动范围变小，先验的方差也变小。通过减小C的值，可以控制w的波动不至于过大，以免受一些数据的影响，造成模型的过拟合（overfitting）。**

另外也有一种更直观的解释，上面regularized形式的目标函数也可以根据KKT条件转为constraint形式，也就是：

argminws.t.∑i=1l?(yi,wTxi)∥w∥22<s2(3)

通过参数s限制w的大小，而s与C也存在着一定正向相关的关系。因此，当C较小时，w的取值也被限制在的一个很小的范围内。下面的图给了一个非常直观的解释：

由于有了对w取值的限制，就出现了两种情况。第一种是当s不够大的时候，此时如果沿梯度下降的方向一直搜索，找到全局最优解，就已经找出圈外，不满足下面的约束项。这个时候，只能在满足约束的条件下找到尽量好的解。根据KKT条件，此时的最优解一定是划定范围的圆圈与目标函数等梯度线相切的位置，如上图左边所示。如果把梯度图看成一座山的等高线，那边最优解的位置一定是等高线中凸起的部分，类似于一座山上的山脊或者山岭，这也是脊（岭）回归的由来。另一种情况是当s足够大的时候，这个时候，在由s所划定的范围内已经能够达到全局最优解，这个时候，下面的约束项其实并没有起到作用，就如上图右边所示。

**因此在调参过程中，如果数据量较少，或者我们对数据的质量信心不足，就应该减少C的大小，增加先验的重要性，反之则可以增加C的大小，让数据本身起更大的作用。而在优化过程中，C越大，需要搜索的w的范围也越大，计算的代价也会越高。而从前面的分析中可以看出，当C增加到一定程度后，模型已经能确保达到全局最优，此时继续增加C对提高算法的表现已经没有帮助。**因此，**在LIBLINEAR中，实际推荐的做法是将C的值从小向大来调，当C增加之后已经无法改变算法的效果时，说明C对模型已经没有影响，就没有必要继续调下去了。选择较小的C反而可以提高模型收敛的速度。**

L1norm的使用

在LIBLINEAR中，除了提供上面提到的L2norm正则化项之外，还提供了L1norm的选项。L1norm一般写成∥w∥1，其实就是对向量w中的所有元素的绝对值进行求和。**与L2norm相比，L1norm也具有对w本身大小的约束，使得w的某些维度值不至于过大而导致过拟合。这个特性在统计学上也称作收缩”shrinkage”。而此外，L1norm还有另外一个非常有用的特征，即能够使学习到的w比较稀疏（sparse），也就是存在很多的0项，而且可以通过系数C控制0项的的个数。当C减小时，w的非0项就增多，当C无限小时，由于完全没有拟合损失的压力，w也可以变成全部是0了。**

为什么L1norm还有这样的功能呢？说来话长，下次可以专门再写博客讨论了。简单来说，是由L1norm的特殊性质决定的。L1norm所对应的绝对值函数是连续的，但并非处处可导，因为在0点存在特殊的情况，即左右导数实不相等的。因此可以认为定义sub\_gradient来描述这种特殊情况的导数，这样的结果是在这个特殊点上，导数不是一个值，而是左右倒数中间的一个范围。**由于在0点导数取值非常灵活，使得在模型求解的过程中，很容易在这样的点达到极值，也就会使得学习到的w尽量的稀疏。**

**基于L1的线性回归模型也被叫做LASSO。采用这个惩罚项的基本动机是认为只有少数特征是与分类结果真正相关，而绝大多数特征是无关的。**这一假设确实存在一定的道理，但是在实际分类准确度上，L1norm正则化项对L2norm项并没有绝对的优势，这是因为L2norm虽然不能直接去除那些与分类结果无关的特征，但是模型学习的结果也会让这些特征的权重很低，所以他们能起到的影响不大。**但是L1norm的一个好处是可以作为一个特征选择的工具，从高维特征空间中只选取少数一些与分类结果最为相关的特征进行计算。这在处理大量高维数据或者实时计算问题是，还是非常有帮助的，可以大大减少存储，提高计算的效率。**

**Kernel相关参数**

如果使用LIBSVM的话，参数调节的工作就会更复杂一些。首先是kernel的使用。**一般来说rbf kernel是被鼓励优先使用的。**如果使用rbf kernel效果都无法调到满意，那么采用poly或linear也无济于事。在一些特殊场景下，可以考虑自定义kernel，这在LIBSVM中也是支持的。

rbf的全称是Radial Basis Function，中文叫做径向基函数。而一般在SVM使用的rbf是Gaussian RBF，也就是我们一般所说的高斯核函数。给定两个点x1和x2，Gaussian RBF定义为：

K(x1,x2)=exp(?γ∥x1?x2∥2)(4)

可见，高斯核函数是对两点之间的欧氏距离进行了一定的变换，且变换受到参数γ的控制。应该怎样理解高斯核函数的意义与γ的作用的？

K(x1,x2)=exp(?γ∥x1?x2∥2)=exp(?γ(?∥x1∥2+2xT1x2?∥x2∥2))=exp(?γ∥x1∥2)exp(?γ∥x2∥2)exp(2γxT1x2)=exp(?γ∥x1∥2)exp(?γ∥x2∥2)∑n=0∞(2γxT1x2)nn!(5)

当γ较小，或者说在极端情况趋向于0的时候，可以有∑∞n=0(2γxT1x2)nn!≈2γxT1x2，也就是n>1以后的项远小于n=1项。这个时候，rbf kernel的效果其实和linear kernel相差无几。

相反，当γ增大时，n>1以后的项也产生作用，其基本思想和poly kernel差不多，只是rbf直接把维度从有限维度的多项式上升到了无穷维的多项式而已。当γ无限增大时，可以看到，除非∥x1?x2∥2为0，否则K(x\_1, x\_2) 都会无限趋近于0。也就是说，当γ趋近于无穷大的时候，每一个数据点除了和其自身外，和其他点得距离都为0。在这种情况下，模型训练的结果只能是把所有点都作为支持向量，因此在训练数据上精度可以达到100%。这样的模型也可以看成KNN的特殊情况，此时的K等于1。

上面分别讨论了问题的两个极端情况。当γ无限小的时候，rbf核SVM和线性SVM效果类似，因此模型的复杂度，或者说VC维较低，不容易过拟合。而当γ值无限增大时，所有点都变成支持向量，模型复杂多或者说VC维最高，也最容易过拟合。而一般情况下，γ取一个中间值，也就间距两者的意义，相比于线性模型，可以选择更多的支持向量，增加模型的复杂度与拟合能力。而相比于1-NN模型，也会适当降低模型的复杂度，避免过拟合的风险。此外，从上面的讨论也可以看出，通过参数调整，rbf核基本上可以达到与线性核以及poly核差不多的效果。所以，在不考虑计算效率的情况下，为了达到最优模型，只需要针对rbf模型进行调参就可以了。

上面在参数调整的讨论里，都假设参数C是固定的。但在实际SVM的调参过程中，C和γ是同时变化的，这进一步增加了调参的复杂性。关于两个变量之间的关系，也有很多理论上的分析与讨论，这里不过多进行讨论，可以参考文件：Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel。

在实际的应用场景下，我们可以通过cross-validate的方法对参数进行遍历，选择最佳的参数组合。LIBSVM也提供了grid.py工具，用于SVM的调参。在一般的应用中，我们只需要设置参数的可变范围，然后在训练数据中对参数组合进行遍历，选择最优参数组合就可以了。