1. **Logistic回归**

线性回归是线性模型用于回归分析，在分类问题中，给出的是真实值，需要将其转换为0/1值。对数几率函数Logistic就是常用的映射函数。



Logistic回归的优点：

算法复杂度低，易于实现，适合分布式使用

Logistic回归的缺点：

1. 容易欠拟合，要求数据集符合伯努利分布，分类精度不高。
2. 数据集的特征很多，而样本数据量少的情况下，效果很差。
3. **线性回归**

线性回归的方程：



线性回归试图使：



确定和，就是线性回归主要的工作。实际过程中，一般采用使用均方误差函数最小化来确定和。

均方误差函数最小化：



参考：优达机器学习3-梯度下降.docx，优达机器学习5-公式推导.docx

1. **隐马尔科夫模型**

项目中的隐马尔科夫模型：主要是隐马尔科夫模型中的最优序列问题，即解码问题，给定隐状态初始概率，隐状态之间转移概率矩阵，混淆矩阵（给定隐藏状态下，观察状态的概率矩阵），采用维特比算法。

观察状态：字母，数字，单个字符，冒号，点号，逗号，句号，DOI等14种状态。

隐藏状态：标题，作者，label start，label，label end，类型，年份，来源等34种状态。

隐藏状态初始概率矩阵的计算：

给定大约20多个参考文献的样例，将其隐状态进行编码，并利用字典存储各个隐状态和其出现的次数。计算初始概率时，如果字典中包含某个状态，则用其出现的次数除以样例参考文献中所有的状态数，如果不包含某个状态，则视为0.

混淆矩阵的计算：

给定大约20多个参考文献的样例，将其隐状态和观察状态分别进行编码（编码后的隐状态序列和观察状态序列长度肯定是相同的），然后利用字典进行存储，其中键为隐状态和其对应的观察状态组成的状态对，值为这种状态对出现的次数。计算时，针对所有（不光是样例中的隐状态）的隐状态，对于某一个隐状态s，统计字典中所有隐状态为状态s的键对应的总次数（字典中可能有多个键的隐状态都是s，但对应的观察状态不同），然后针对所有观察状态，计算s和所有观察状态组成状态对的概率，如果字典中包含相应的状态对，则用对应的次数除以隐状态s出现的总次数。如果不存在，不能视为0，而是赋予它一个很小的概率0.00000001.这表明即使知道s和某个观察状态不太可能组成状态对，也要给予相应的小概率。这有点类似强化学习中Q-Leanring中epision-greedy策略中采用的方法。

隐状态转移概率矩阵的计算：

给定大约20多个参考文献的样例，将其隐状态进行编码，然后利用字典进行存储，其中键为隐状态和隐状态(前一个隐状态和后一个隐状态)组成的状态对，值为这种状态对出现的次数。计算时，针对所有的隐状态，对于某一个隐状态s，统计字典中所有第一个隐状态为状态s的键对应的总次数（字典中可能有多个键的隐状态都是s，但对应的另一个隐状态不同），然后计算s和所有隐状态组成状态对的概率，如果字典中包含相应的状态对，则用对应的次数除以s出现的总次数。如果不存在，不能视为0，而是赋予它一个很小的概率0.00000001.这表明即使知道s不太可能转移到某个隐状态，也要给予相应的小概率。

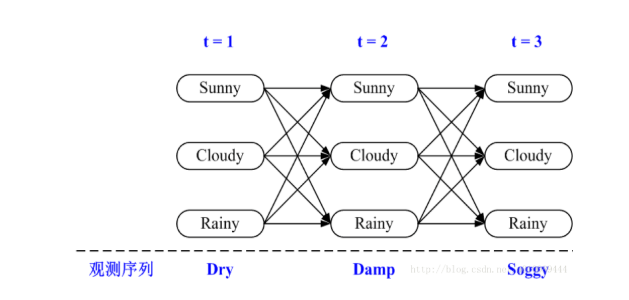
维特比算法：

输入：初始概率，转移概率，观察状态混淆矩阵概率，观察序列（在本项目中是由观察状态对应的一组整数值，字对应0，数字对应1等）。

其中，为隐状态总数，为观察状态总数，为某条参考文献对应的观察状态数，也可以理解为时刻总数，每个时刻对应观察序列的一个状态。

维特比算法采用动态规划的方式。对于一个观察序列，时刻其观察状态对应的隐状态都有种可能，只是概率不同，将这些不同的隐状态看成一个个结点，根据观察状态序列查找最有可能的隐状态序列，其实就是在这些结点网络上查找一条最优路径。如下图所示。参考：

<https://blog.csdn.net/u010089444/article/details/72230745>



其中Sunny, Cloudy, Rainy为隐状态。

根据动态规划的原理：如果时刻，最优路径通过结点，那么这一路径从结点到终点的所有可能的局部路径来说，必须是最优的。如果不是最优的，从到另外一条更好的路径把它从到的部分组合起来，就会变成更优的路径。

根据上述的原理，从时刻开始，递推计算时刻状态为的各条局部路径的最大概率，直至得到时刻状态为的各条路径的最大概率。时刻的最大概率即为最优路径的概率，同时可以确定最优路径的终结点，从终结点开始，由后向前得到，最终得到最优路径，也就是最有可能的隐藏状态序列。

1. 局部最优路径概率：时刻到达指定状态的最有可能的路径的概率。局部路径分为两种情况，和。

先看的情况：如果希望计算时刻状态为的最可能的路径，而该路径在时刻有可能经过三种可能的隐状态，根据一阶马尔可夫假设：当前系统处于某特定状态的概率仅仅取决于此前的1个状态，从到的概率为：



其中，为时刻最优路径通过结点的概率，为从到的状态转移概率，为时刻观察状态为observation，隐状态为的混淆概率。

因此，时刻状态的局部最优路径概率为：



对上述进行推广：时刻，观察状态为，到达隐状态为的局部最优路径概率



其中，为转移概率，为观察状态为，隐状态为的混淆概率。

的情况：由于没有状态转移，所以结合初始状态概率来计算。



其中，为状态为的初始概率，为观察状态为，隐状态为的混淆概率。

C#编程

DELTA = new *Double*[OB.Length, N]; // 局部概率

PSI = new *Int32*[OB.Length, N]; // 反向指针

// 1. 初始化

// t=1的情况

// 本项目中，隐状态值为012345...

for (*Int32* j = 0; j < N; j++)

{

// B[j, OB[0]]为隐状态j，观察状态为观察序列第一个值的混淆概率

DELTA[0, j] = PI[j] \* B[j, OB[0]];

}

// 2. 递归

// t>1的情况

for (*Int32* t = 1; t < OB.Length; t++)

{

for (*Int32* j = 0; j < N; j++)

{

*Double* MaxValue = DELTA[t - 1, 0] \* A[0, j];

*Int32* MaxValueIndex = 0;

for (*Int32* i = 1; i < N; i++)

{

*Double* Value = DELTA[t - 1, i] \* A[i, j];

if (Value > MaxValue)

{

MaxValue = Value;

MaxValueIndex = i;

}

}

DELTA[t, j] = MaxValue \* B[j, OB[t]];

// 记录下最有可能到达此状态的上一个状态

PSI[t, j] = MaxValueIndex;

}

}

// 3. 终止

*Int32*[] Q = new *Int32*[OB.Length]; // 定义最佳路径

// 时刻t=T，最大概率，默认隐状态为0

Probability = DELTA[OB.Length - 1, 0];

Q[OB.Length - 1] = 0; // 终结点默认为隐状态为0的结点

// 查找时刻t=T的最优路径概率，并记录最优终结点

// 这里的i表示的是隐状态对应的编码

// for循环结束后，Q[OB.Length-1]存储的是概率最大// 的隐状态，也就是终结点对应的隐状态

for (*Int32* i = 1; i < N; i++)

{

if (DELTA[OB.Length - 1, i] > Probability)

{

Probability = DELTA[OB.Length - 1, i];

Q[OB.Length - 1] = i;

}

}

// 4. 路径回溯

// PSI记录是时刻t隐状态为j最有可能的上一个状态

// 最优终结点已知晓，所以可以反向推导出上一个最优状态结点

// 从而推导出最优路径

for (*Int32* t = OB.Length - 2; t >= 0; t--)

{

Q[t] = PSI[t + 1, Q[t + 1]];

}

return Q;

1. **决策树**

1.1 基本概念

假设是一个离散型随机变量，其概率分布函数为，则的信息量为：概率对数的负数，概率越小，信息量越大



信息熵：对于某个事件，有n种可能性，每一种可能性的概率为，则信息熵为所有信息量的期望：



联合熵：如果随机变量有多个，则信息熵就会变化成联合熵：



条件概率：



<https://www.cnblogs.com/fantasy01/p/4581803.html>

条件熵：为已知随机变量的条件下，随机变量的不确定性：



信息增益：待分类的集合的熵减去按某个条件（特征）分类后的条件熵



在决策树算法中，分类效果越好，就会越小，相应的信息增益就会越大。

常见的决策树算法有：ID3，C4.5，CART

设数据集为，类别为，其中一个特征为，共有个值。则数据集的信息熵为：



其中为第类的样本子集。

特征对数据集的条件熵为：



其中表示数据集中特征取第个值的子集，表示中属于第类的子样本。

对照之前的和，二者是相通的，就是在计算概率。

信息增益：



1. ID3

将信息增益最大化。具体过程：

1. 从根节点开始，对结点计算所有可能的特征的信息增益，选择信息增益值最大的特征作为结点的划分特征；
2. 由该特征的不同取值建立子结点；
3. 再对子结点递归地调用以上方法，构建决策树；
4. 到所有特征的信息增益都很小或者没有特征可以选择为止，得到最终的决策树

ID3算法简单，易于实现，但往往会倾向取值较多的特征。特征的取值越多，划分后确定性就越高，条件熵就越小，信息增益就越大。在极端情况下，如果样本的某个特征具有独一无二性，则条件熵为0（），ID3就会选用这个特征，但这样的决策树泛化能力很差。

1. C4.5

将信息增益比最大化。

信息增益比：



其中，，称为数据集关于特征的信息熵。

C4.5的基本流程与ID3类似，只是采用了信息增益比。相比于ID3，如果某个特征的取值较多，则会更大，信息增益比就会越小，相当于给信息增益乘上了惩罚系数。

ID3和C4.5只能用于分类

1. CART

最大基尼系数。

基尼系数：表示数据集的不确定度



条件基尼系数：



1. 从根结点开始，对结点计算现有特征的基尼指数，对每一个特征，例如，再对其每个可能的取值如,根据样本点对的结果的是与否，划分为两个部分，利用进行计算；
2. 在所有可能的特征以及该特征所有的可能取值中，选择基尼指数最小的特征及其对应的取值作为最优特征和最优切分点。然后根据最优特征和最优切分点，将本结点的数据集二分，生成两个子结点
3. 对两个子结点递归地调用上述步骤，直至结点中的样本个数小于阈值，或者样本集的基尼指数小于阈值，或者没有更多特征后停止；
4. 生成CART分类树；

CART树是一颗二叉树（ID3，C4.5则未必）。

CART回归树：CART分类树是依据基尼系数作为分类的条件，CART回归树是根据样本的最小均方差作为分类条件，选择是样本均方差最小的特征进行分裂。

回归树：采用最小平方误差准则。

决策树的剪枝：直接生成的决策树过于复杂，泛化能力差，所以需要剪枝。剪枝的目的是提升其泛化能力，而不是最小化损失函数。

剪枝分为预剪枝和后剪枝。

预剪枝：树结点扩展之前剪枝。在Udacity使用决策树时经常会设置相关的参数。

1. 当树达到一定深度，停止树的生长
2. 当前结点的样本数量小于某个阈值时，停止树的生长
3. 计算每次分裂对训练集的准确度提升，小于某个阈值时，停止树的生成

后剪枝：对一颗完全生长的决策树进行剪枝。流行的CART剪枝

参考：

<https://www.jb51.net/article/131084.htm>

<https://blog.csdn.net/beauty0522/article/details/82726866>

1. 从完整的决策树开始，生成一系列子树序列。其中由生成。裁剪中关于训练集误差增加率最小（剪枝是为了提高泛化能力）的分支得到。具体地，当树在结点处剪枝时，误差增加可以用表示，其中可以用增益或基尼系数表示，表示剪枝之后该结点的误差(即剪掉该结点的子树后，该结点的会产生的误差)，表示未进行剪枝时子树的误差(即该结点的子树存在并进行分类后所产生的误差)。误差增长率为：



其中表示子树的叶节点个数。

1. 经过步骤1)的剪枝，直到只剩下一个结点，最终会产生一个子树序列，通过k折交叉验证，k-1份生成决策树，最后一份用于选择最优子树，最终得到剪枝后的决策树。

决策树的优缺点：

优点：模型易于理解，分类速度快，不需要参数假设。

缺点：容易过拟合，忽略属性之间的相关性

1.2 梯度提升决策树(Gradient Boosting Decision Tree, GBDT)

这里的梯度提升并不是梯度上升，实际GBDT算法在降低损失函数时仍然是梯度下降。

GBDT采用加法模型（即多个弱分类器叠加），以决策树（一般为CART回归树）为基函数，采用前向分步算法。这些都类似Adaboost。与Adaboost不同的是如何识别并提升模型。Adaboost采用错分数据点来识别问题，通过调整错分数据点的权重来改进模型。GBDT采用损失函数的负梯度来识别问题，通过负梯度来改进模型。

GBDT可以用于分类问题和回归问题，实质上会将分类问题转化为回归问题。在回归问题中，GBDT采用前向分步算法：







其中，是一个弱分类器，表示当前的模型。可以看出最终的模型是由一系列的弱分类器线性组合而成，GBDT是一个加法模型。

从上述公式可以看出，每轮的弱分类器是在上一轮分类器的残差的基础上进行训练，其拟合的实际为。

每一步需要求解：



从而得到第m颗树的参数。其中代表损失函数。

如果采用平方误差作为损失函数（也可以采用其他损失函数），

则：



其中为当前模型的残差。

GBDT每一轮训练所关注的重点是本轮产生的残差，下一轮以本轮残差为输入，尽可能使残差减小。

对于一般损失函数，不像平方误差那样简单，因此使用损失函数对当前模型的负梯度作为残差的近似值，然后去拟合一个回归树。

Adaboost与GBDT的区别与联系：

1. 二者都是加法模型（弱分类器级联），采用前向分步算法
2. Adaboost一般只用于分类，GBDT可以用于分类和回归
3. Adaboost通过修改错误分类点的权重来改进模型，而GBDT通过损失函数的负梯度模拟残差来改进模型。

1.3 XGBOOST

GBDT只利用了损失函数的一阶导数信息（即梯度），XGBOOST将损失函数进行二阶泰勒展开，并加入正则项。



其中正则项为：



其中为叶子结点个数，表示第个叶子结点的预测值。

XGBOOST与GBDT的联系与区别：

1. GBDT是机器学习算法，XGBoost是该算法的工程实现。
2. 在使用CART作为基分类器时，XGBoost显式地加入了正则项来控制模型的复杂度，有利于防止过拟合，从而提高模型的泛化能力。
3. GBDT在模型训练时只使用了代价函数的一阶导数信息，XGBoost对代价函数进行二阶泰勒展开，可以同时使用一阶和二阶导数。
4. 传统的GBDT采用CART作为基分类器，XGBoost支持多种类型的基分类器，比如线性分类器。
5. 传统的GBDT在每轮迭代时使用全部的数据，XGBoost则采用了与随机森林相似的策略，支持对数据进行采样。
6. 传统的GBDT没有设计对缺失值进行处理，XGBoost能够自动学习出缺失值的处理策略。

<https://blog.csdn.net/baimafujinji/article/details/50554664>

<https://www.cnblogs.com/maybe2030/p/4585705.html>

<https://www.cnblogs.com/fionacai/p/5894142.html>

Bootstrap抽样：“有放回地全抽”，（其实样本量也要视情况而定，不一定非要与原样本量相等），抽取的Bootstrap样本量与原样本相同，只是在抽样方式上采取有放回地随机抽，这样对于一个原始数据集就可以抽样多次，每次抽样的数据集就很有可能有重复数据，相应的也会剔除部分数据。

随机森林：为了解决决策树过拟合的问题，如果数据集的特征非常多，决策树算法就容易陷入过拟合（特征很多，会根据不同特征将训练集分隔成多个方块，不具备泛化能力）。

1. 采用Bootstrap抽样对训练集进行抽样；
2. 从样本的M个特征维度中，随机选取m（m<<M）个特征，只根据这个m个特征建立决策树；
3. 对根据之前两个步骤建立的多个决策树进行投票分析，得票最多的就是最终预测类别。

决策树的一些超参数：

1. 最大深度（max\_depth）：当决策树深度为时，它最多可以拥有片叶子。较大的深度往往会导致过拟合，这是因为过深的决策树可以记忆数据。而较小的深度会使得模型过于简单，导致欠拟合。
2. 每片叶子的最小样本数（min\_samples\_leaf）：当分裂结点时，每片叶子允许的最小样本数。当每片叶子的样本数量较小时，叶子上的样本数量也有可能过于稀少，此时模型将记忆数据，也就是过拟合。当每片叶子的样本数量较大时，决策树能够获得足够的弹性进行构建，这也许会导致欠拟合。
3. 每次分裂的最小样本数（min\_samples\_split）：结点允许分裂最小样本数，即如果结点样本数>min\_samples\_split，则允许分裂，否则不允许。
4. 最大特征数（max\_features）：寻找最佳分裂时，考虑的最大特征数目。
5. **朴素贝叶斯分类：假设各特征之间相互独立，输入数据符合正态分布。**

贝叶斯公式：



为先验概率：即没有条件下，事件A的概率；

为后验概率：条件B发生后，事件A的概率；

<https://www.cnblogs.com/marc01in/p/4775440.html>

<https://www.cnblogs.com/by-dream/p/7884606.html>

例：

一所学校里面有60%的男生，40%的女生。男生总是穿长裤，女生则一半穿长裤一半穿裙子。

P(男)=0.6 P(女)=0.4

P(长裤|男)=1 P(长裤|女)=0.5

根据贝叶斯公式求P(男|长裤)，即穿长裤的人是男生的概率。

P(男|长裤)=P(男)P(长裤|男)/P(长裤)

P(长裤)=P(男,长裤)+P(女,长裤)=P(男)P(长裤|男)+P(女)P(长裤|女)=0.8

则：

P(长裤|男)=0.6\*1/0.8=0.75

P(长裤|女)=0.4\*0.5/0.8=0.25

朴素贝叶斯分类方法：

1. 设为一个待分类项，其中为的特征属性；
2. 设类别集合为，表示总共有n类。
3. 求出，，…
4. 如果，则。

朴素贝叶斯分类的核心就是计算第3步的条件概率。根据贝叶斯公式：



对于所有类别而言，是一个常数，可以忽略掉。

根据贝叶斯假设，的各个特征之间相互独立（“朴素”两个字的来源），所以：



拉普拉斯平滑：上述的公式会存在一个问题，如果某个特征没有再某个分类中出现过，如特征没有在中类别出现过，则为0，从而导致整个公式结果为0，显示不能如此计算。拉普拉斯平滑就是在计算不为0，而是赋一个很小的概率。具体操作如下：

令为在垃圾邮件类别中，单词出现的概率；定义：

令为垃圾邮件中，单词出现的个数。

令为垃圾邮件，所有单词总数目。则：



拉普拉斯平滑：分子+1，分母加特征总数，这样可以保证概率和为1



Udacity中的例子，患病的概率：

人患某种病的概率

P(S)=0.00001

健康的概率

P(H)=0.9999

检测为阳性的准确率为99%（这个概率指的是如果人检测为阳性，准确率为99%，还有%1是本身为阴性，结果误检测为阳性），即：

患病时检测为阳性的概率

P(+|S)=0.99

健康时误检测为阳性时的概率

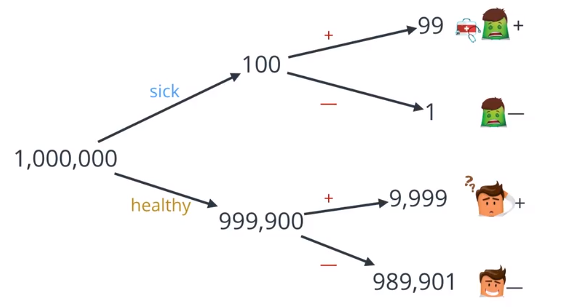
P(+|H)=0.01

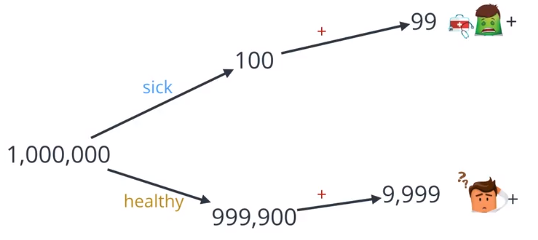
求P(S|+)

P(S|+)=P(S)P(+|S)/P(+)

P(+)=P(S)P(+|S)+P(H)P(+|H)=0.0001\*0.99+0.9999\*0.01

P(S|+)=0.0001\*0.99 / (0.0001\*0.99+0.9999\*0.01)=0.0098





从上述计算来看，即使检测准确率达到99%，检测为阳性并且人患病的准确率最终为0.0098，小于1%，而检测为阳性但人健康的概率（假阳性）却超过了99%，这是由于检测的误差1%远大于人患病的概率。

所以在医学上检测准确率必须非常高，如果准确率达到99.999%，则检测为阳性并且人患病的概率为：

P(S|+)=0.91

朴素贝叶斯算法是一种生成模型，分别计算样本属于某一个类别的概率，然后概率最大的就是样本的类别。

垃圾邮件分类案例：

Udacity中的垃圾邮件示例machine\_learn/bayesian，是根据样本中垃圾邮件和非垃圾邮件中单词出现的频率数作为输入，采用朴素贝叶斯分类进行模型训练，然后对测试数据进行判断是否为垃圾邮件。实际上垃圾邮件中某些单词的频率数会比较高，朴素贝叶斯通过计算后验概率来区分邮件是否为垃圾邮件。

朴素贝叶斯的优缺点：

优点：

1. 算法易于实现，速度快
2. 能够处理大量特征
3. 对缺失数据不敏感，模型比较稳定

缺点：

1. 需要假设特征之间相互独立，数据假设为符合正态分布。
2. 类别如果很多，效果不好（类别概率相乘之后会太小，容易受干扰）
3. **支持向量机（SVM）**

在Logistic回归中：



当时，，预测y=1。

当时，，预测y=0。

如果，那么我们就能很准确的预测y=1，反之，如果，就能很准确的预测y=0.

通过Logistic回归学习得到的参数，如果使得y=1时，，y=0时，，那么可以认为模型是良好的。

如果数据集一定可以线性分隔，Logistic回归考虑全局，得到的分隔模型将正类和负类区别开，而SVM则希望靠近分隔线的元素能尽可能的远离分隔线，不关心那些已经远离分隔线的元素，注重局部。这也是SVM和Logistic的不同。

在SVM中，符号约定：







函数间隔：定义一个训练样本，是特征，是结果标签。



如果希望函数间隔取一个较大的值（这样根据之前的推断，可以更好的区分样本是正类，还是负类），当时，，当时，。因此，函数间隔代表了样本是正类还是负类的确信度。

全局样本的函数间隔：



所有训练样本中，最坏的情况。

几何间隔：训练样本到分隔面的垂直距离



其中表示向量的模。

当时，几何间隔就变成了函数间隔。函数间隔存在一个问题，我们可以通过将和乘以一个倍数来提高函数间隔，尽管这样对模型并无意义。而几何间隔则不存在这个问题，提高了和，则也会跟着一起提高。

全局几何间隔：



最优间隔分类器（optimal margin classifier）：

目标函数1：使得几何间隔最大



约束条件：

其中，保证了函数间隔=几何间隔。

上述的优化问题中，约束条件难以进行局部最优搜索，所以需要变换：

目标函数2：函数间隔除以向量的模最大



约束条件：

其中，

凸函数：

<https://www.cnblogs.com/always-fight/p/9377554.html>

目标函数不是凸函数，难以使用梯度下降等局部搜索算法求解，需要继续变换。

将全局函数间隔取为1，即。目标函数2变为：



演变为目标函数3：



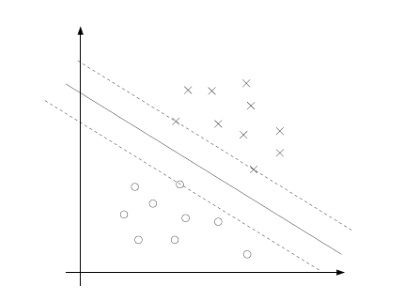
约束条件：

根据拉格朗日对偶问题（不细究其具体推导了）：

目标函数3的拉格朗日对偶问题为：



其中，称为拉格朗日算子。



上图中实线是最大间隔超平面，虚线是函数间隔为1的超平面，其上的点的拉格朗日算子，其它点的拉格朗日算子，上图中两条虚线上的3个点称为支持向量。

对上述的拉格朗日对偶问题求偏导：



令其分别为0：



从而得到如下问题：其中，，均为实数



进一步推导：



其中将向量内积表示为。

目标问题4：



约束条件：

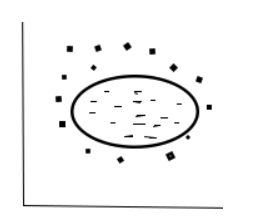


根据之前的求解：



之前如果有新的样本需要分类，需要根据和进行一次线性运算，现在有了后，只需要将新的样本与之前的样本进行运算就可以了，而且只有支持向量的，其它的，所以新的样本只要与支持向量进行运算就可以了。

核函数：对于非线性问题（与线性不可分不同，后续有说明），需要进行非线性变换，将非线性问题变成线性问题。



上图中，利用直线无法进行分隔，但利用椭圆则能有效的分隔。

令表示映射后的特征向量：

则目标函数4转换为目标函数5：



约束条件：



核函数：由于映射到特征空间的维数可能很高，直接计算比较困难，令：



目标函数5：



约束条件：





常用的核函数：

线性核：



多项式核：



高斯核（）：



也就是所谓的径向基核函数，其中为sklearn中的gamma参数，越大曲线越尖锐，越小曲线越平缓。

拉普拉斯核（）：



软间隔支持向量机和松弛变量（）：

如果出现线性不可分的情况，意味着某些样本点的函数间隔不满足大于等于1的条件，样本点落在最大间隔超平面和边界超平面（函数间隔=1的超平面）之间了。

对每个样本点引入一个松弛变量：

目标函数3转变为：



约束条件：

拉格朗日对偶问题为：



其中，为拉格朗日算子。

求其偏导数：



令其分别为0：



目标函数4最终转变为：



约束条件：



sklearn中SVM的超参数：

1. 松弛变量（），也叫惩罚因子，代表对错误分类的惩罚力度，C越大表示惩罚越大。
2. 核函数kernel
3. gamma参数，核函数为rbf，即径向基函数
4. degree参数，核函数为poly

SVM的优缺点：

优点：

1. 相比对其他的机器学习算法，SVM没有对特征数据的分布进行任何假设，如果不清楚特征数据的分布，选择SVM相对比较好。
2. 对于线性不可分的问题，可以通过核函数映射到高维特征空间实现线性可分。
3. 适用于小样本

缺点：

1. 针对大样本数据集难以使用，因为对计算资源消耗大（有大量的矩阵计算）。
2. 对于多分类问题效果不好，只能用于二分类。
3. **集成方法**

将各种模型连接起来，获取到更好的模型。有两种最常见的集成方法：bagging（boost aggregating，自助聚集）和boosting（增强学习）。

Bagging：

1. 从原始样本集中抽取训练集。每轮使用Bootstraping的方法抽取n个训练样本（sklearn中的方法可以控制是否有放回抽样，并控制抽取的样本数），共进行k轮抽取，得到k个训练集，并认为k个训练集是相互独立的。
2. 每次使用一个训练集得到一个模型，k个训练集得到k个模型（根据具体问题采用不同的分类或回归方法）
3. 对于分类问题：将上步得到的k个模型采用投票方式得到分类结果；对于回归问题，计算上述模型的均值作为最后的结果。所有模型的权重是相同的。

AdaBoosting：使用全部的样本，每轮训练改变样本的权值，减小训练正确的样本的权值，增大错误样本的权值。每轮只训练一个弱分类器。

公式推导：

训练样本集为，其中；

初始化训练样本的权值分布：

对于训练轮次：

1. 使用具有权值分布的训练数据集进行学习，得到弱分类器；
2. 计算在训练数据集上的分类误差率：



1. 计算在强分类器（最终的分类器）中所占的权重：

假设经过m-1轮计算后，得到，根据前向分步算法，则：



其中为所占的权重。

AdaBoosting采用指数损失，损失函数为：



令，为每一轮迭代的样本权值，则：



根据特性，如果分类正确与符号相同，如果分类错误与符号相反，则：



其中，分类误差率，则：



对求偏导，令，得到：



可以看出准确性越高的弱分类器，所占的权重就越高。