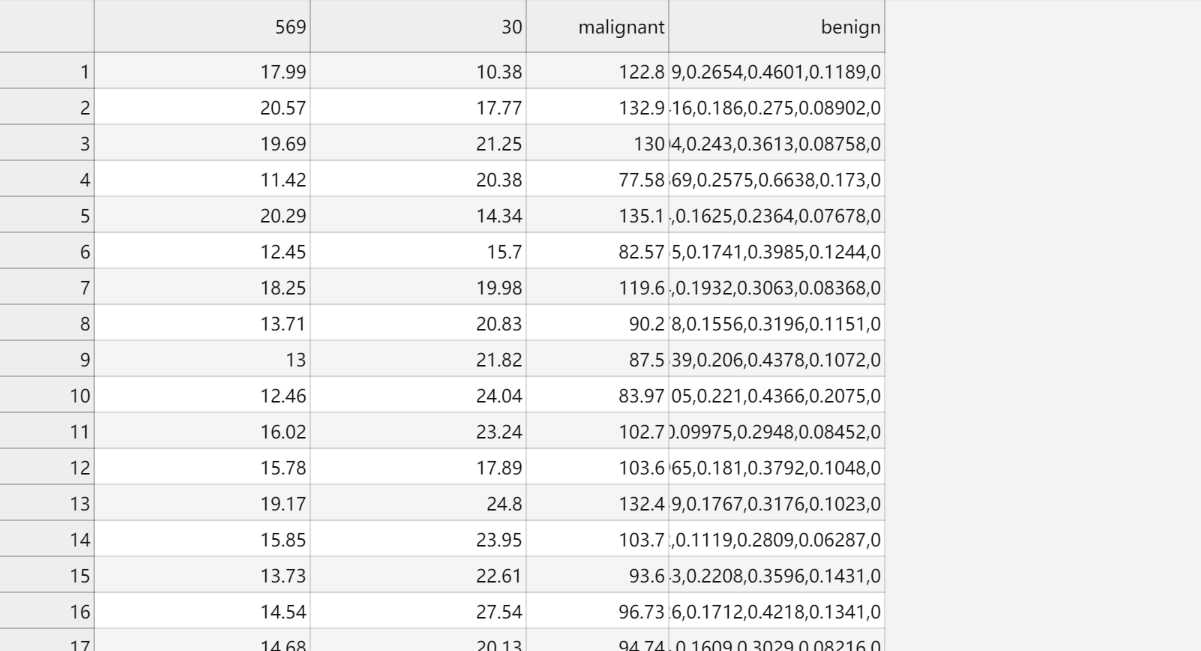
基于机器学习算法的乳腺癌预测

任务：本文使用多个机器学习和深度学习算法，建立模型，能够对病人的化验结果作出判断，是良性(benign)还是恶性(malignant)，并且对每一种算法进行调参并计算精确度，最后得到一个比较可靠的算法。

基本思路：

机器学习：用于训练和检测的数据集就在文件夹中的breast\_cancer.csv中，它包含在python scikit-learn.datasets的load\_breast\_cancer()函数中。通过使用scikit\_learn.model\_selection 中的test\_train\_split()函数将数据集分为X\_train,X\_test,y\_train,y\_test四个部分，分别用来训练和验证模型。最终使用score函数得到模型的精确度(accuracy)。最终发现，使用支持向量机(SVM)算法的测试集准确度最高。

数据集：



算法一：K近邻算法

算法原理：

1、KNN算法概述:

接下来对KNN算法的思想总结一下：就是在训练集中数据和标签已知的情况下，输入测试数据，将测试数据的特征与训练集中对应的特征进行相互比较，找到训练集中与之最为相似的前K个数据，则该测试数据对应的类别就是K个数据中出现次数最多的那个分类，其算法的描述为：

1）计算测试数据与各个训练数据之间的距离；

2）按照距离的递增关系进行排序；

3）选取距离最小的K个点；

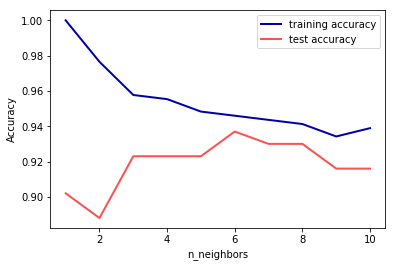
4）确定前K个点所在类别的出现频率；

5）返回前K个点中出现频率最高的类别作为测试数据的预测分类。

算法实现：在jupyter notebook 中运行：



输出：



发现 当n=6时，最高性能准确率约为0.93

线性回归：

在乳腺癌数据集上使用LogisticRegression

In[5]:

*#根据不同的c值，计算预测模型的精度，默认为1*

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.datasets** **import** load\_breast\_cancer

cancer = load\_breast\_cancer()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

cancer.data, cancer.target, stratify=cancer.target, random\_state=42)

logreg = LogisticRegression().fit(X\_train, y\_train)

print("Training set score: **{:.3f}**".format(logreg.score(X\_train, y\_train)))

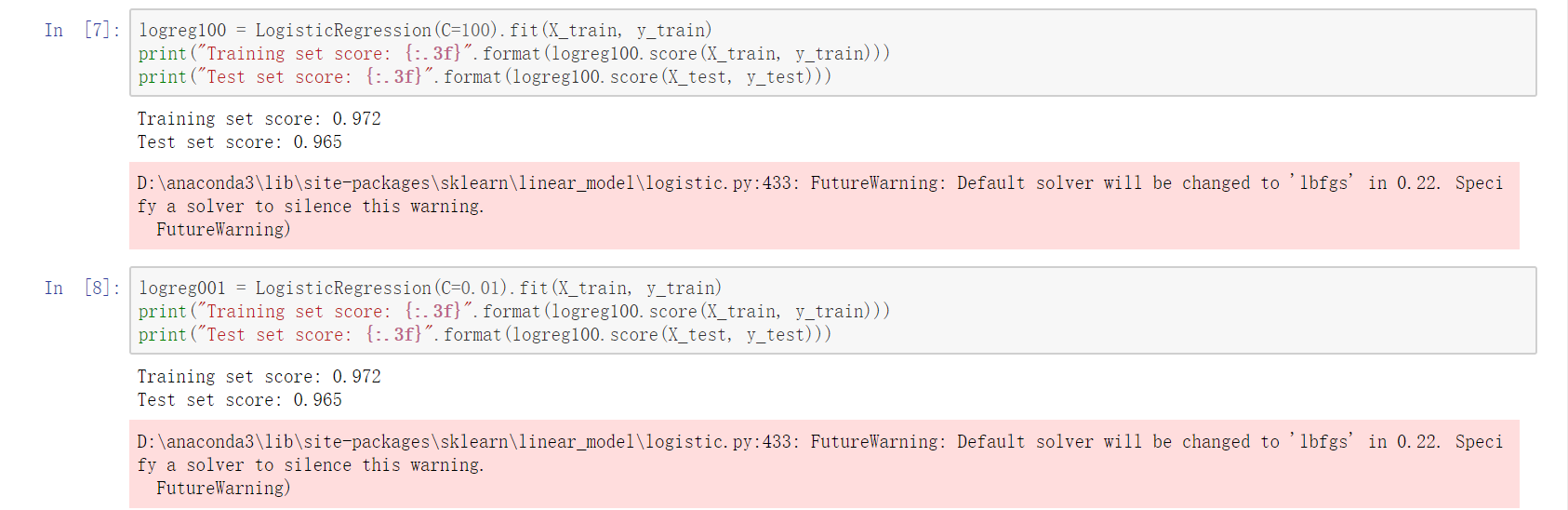
print("Test set score: **{:.3f}**".format(logreg.score(X\_test, y\_test)))

Out[5]:

Training set score: 0.955

Test set score: 0.958

C=1的默认值已经达到了百分之95的精度，但是两者非常接近，可能存在过拟合情况，于是我们尝试改变C的大小



使用更高的c值可以得到更高的训练集精度以及测试集精度。

最后，来看一下正则化参数取三个不同的值时模型学到的系数。

In[9]:

*#L2正则化*

plt.plot(logreg.coef\_.T, 'o', label="C=1")

plt.plot(logreg100.coef\_.T, '^', label="C=100")

plt.plot(logreg001.coef\_.T, 'v', label="C=0.001")

plt.xticks(range(cancer.data.shape[1]), cancer.feature\_names, rotation=90)

xlims = plt.xlim()

plt.hlines(0, xlims[0], xlims[1])

plt.xlim(xlims)

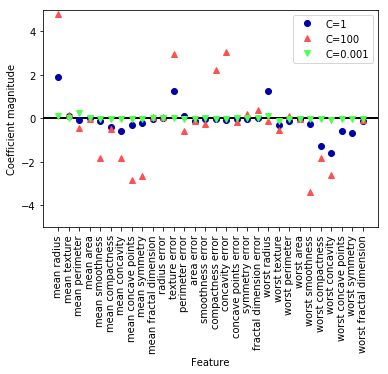
plt.ylim(-5, 5)

plt.xlabel("Feature")

plt.ylabel("Coefficient magnitude")

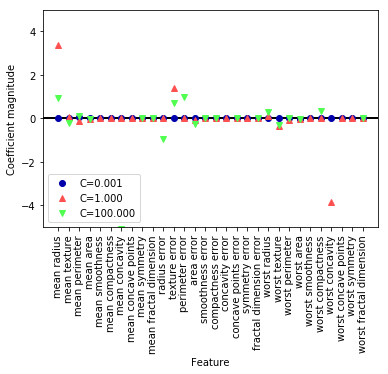
plt.legend()

Out[9]:



接下来使用L1正则化

解释性更强的L1正则化的系数图像与分类精度：

、

决策树：

决策树是广泛用于分类和回归任务的模型，本质上，他通过一层层的if/else问题进行学习，并得出结论。为了防止过拟合的情况出现，我们一般需要剪枝，这里我们实现了预剪枝处理，设置最大深度。

#使用默认设置的决策树模型

from sklearn.tree import DecisionTreeClassifier

cancer = load\_breast\_cancer()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

cancer.data, cancer.target, stratify=cancer.target, random\_state=42)

tree = DecisionTreeClassifier(random\_state=0)

tree.fit(X\_train, y\_train)

print("Accuracy on training set: {:.3f}".format(tree.score(X\_train, y\_train)))

print("Accuracy on test set: {:.3f}".format(tree.score(X\_test, y\_test)))

Accuracy on training set: 1.000

Accuracy on test set: 0.937

很容易发现，出现了过拟合问题，我们考虑剪枝。

#设置最高深度为4

tree = DecisionTreeClassifier(max\_depth=4, random\_state=0)

tree.fit(X\_train, y\_train)

print("Accuracy on training set: {:.3f}".format(tree.score(X\_train, y\_train)))

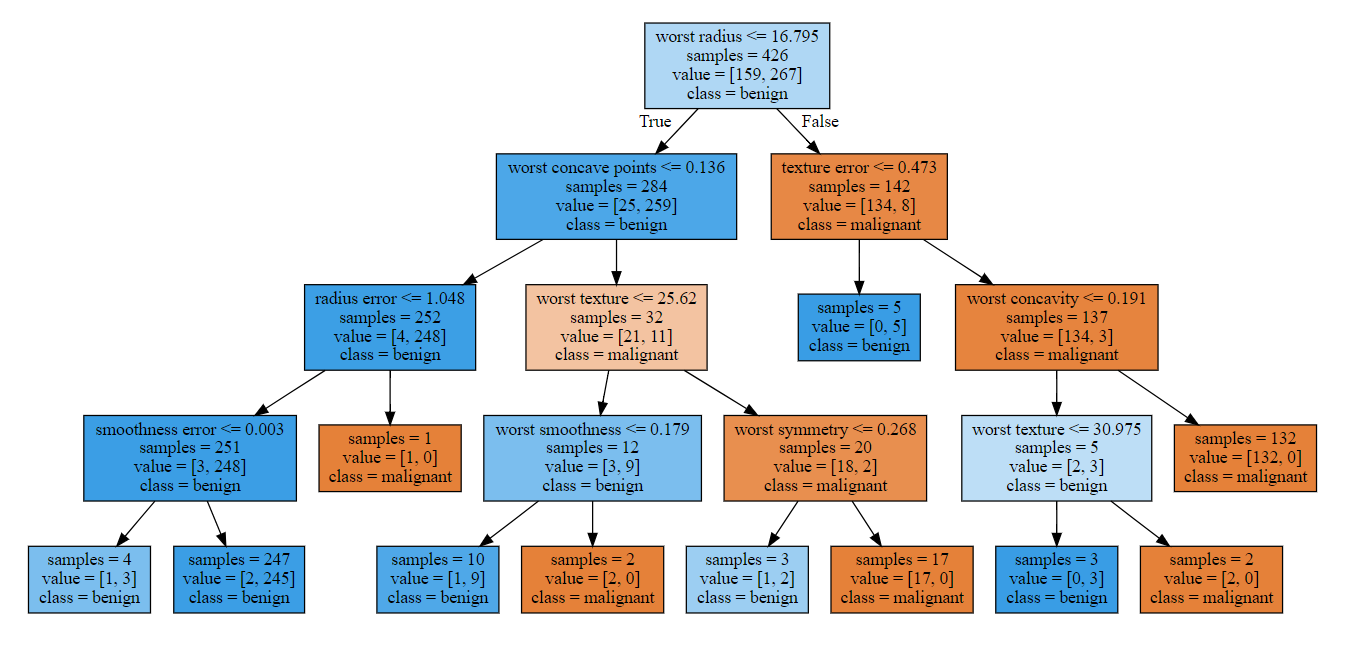
print("Accuracy on test set: {:.3f}".format(tree.score(X\_test, y\_test)))

Accuracy on training set: 0.988

Accuracy on test set: 0.951

测试集精度明显提高了，这是因为限制树的高度可以有效减少过拟合。

接下来将决策树可视化：



查看整个树有时候比较费劲，所以可以利用**特征重要性**（feature importance）来总结树的工作原理。

print("Feature importances:")

print(tree.feature\_importances\_)

Feature importances:

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.01 0.048

0. 0. 0.002 0. 0. 0. 0. 0. 0.727 0.046 0. 0.

0.014 0. 0.018 0.122 0.012 0. ]

还可以将特征重要性可视化：

def plot\_feature\_importances\_cancer(model):

n\_features = cancer.data.shape[1]

plt.barh(np.arange(n\_features), model.feature\_importances\_, align='center')

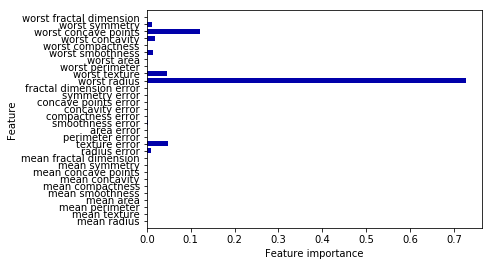
plt.yticks(np.arange(n\_features), cancer.feature\_names)

plt.xlabel("Feature importance")

plt.ylabel("Feature")

plt.ylim(-1, n\_features)

plot\_feature\_importances\_cancer(tree)



(“worst radius”)是最重要的特征，这也可以观察到第一层划分时已经将两个类别区分的很好。

决策树的优缺点：

得到的模型很容易可视化，并且完全不受数据缩放的影响，所以决策树算法不需要特征预处理，但是即使做了预剪枝，决策树模型仍然很容易过拟合，泛化能力很差。

为了解决以上问题，引入决策树集成模型：

随机森林模型：

将包含100棵树的随机森林应用到乳腺癌数据中。

from sklearn.ensemble import RandomForestClassifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

cancer.data, cancer.target, random\_state=0)

forest = RandomForestClassifier(n\_estimators=100, random\_state=0)

forest.fit(X\_train, y\_train)

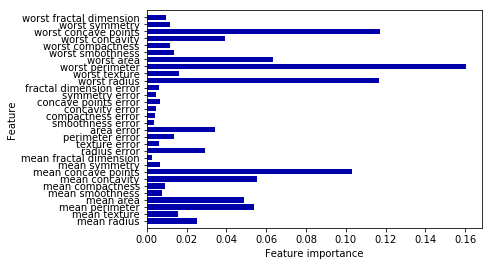
print("Accuracy on training set: {:.3f}".format(forest.score(X\_train, y\_train)))

print("Accuracy on test set: {:.3f}".format(forest.score(X\_test, y\_test)))

Accuracy on training set: 1.000

Accuracy on test set: 0.972

plot\_feature\_importances\_cancer(forest)



可以发现随机森林不需要反复调参就可以达到很好的效果。

下面是kaggle竞赛中常用的一种模型：

梯度提升回归树：

下面是在乳腺癌数据集上应用GradientBoostingClassifier的示例，默认使用100棵树，

最大深度是3，学习率为0.1:

from sklearn.ensemble import GradientBoostingClassifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

cancer.data, cancer.target, random\_state=0)

gbrt = GradientBoostingClassifier(random\_state=0)

gbrt.fit(X\_train, y\_train)

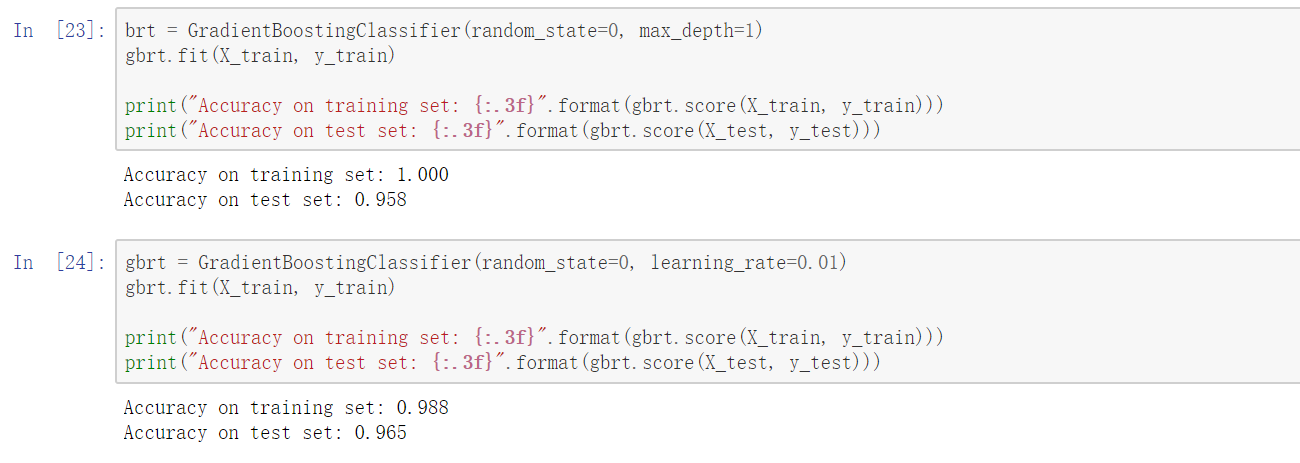
print("Accuracy on training set: {:.3f}".format(gbrt.score(X\_train, y\_train)))

print("Accuracy on test set: {:.3f}".format(gbrt.score(X\_test, y\_test)))

Accuracy on training set: 1.000

Accuracy on test set: 0.958

由于训练集精度百分之百，出现了过拟合现象，为了降低过拟合，我们可以调整最大深度和学习率两个参数。



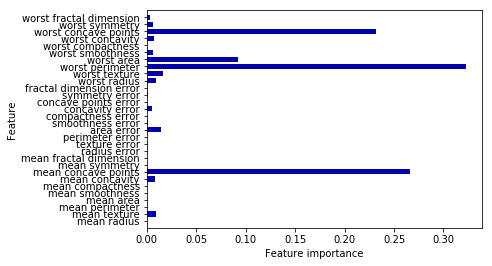
两种方法都达到了提升测试集精度的效果，其中减少max\_depth效果更好。

接下来将树可视化

gbrt = GradientBoostingClassifier(random\_state=0, max\_depth=1)

gbrt.fit(X\_train, y\_train)

plot\_feature\_importances\_cancer(gbrt)



最后我们使用强大的支持向量机模型：

一、什么是支持向量机(SVM)?

1、支持向量机（Support Vector Machine，常简称为SVM）是一种监督式学习的方法，可广泛地应用于统计分类以及回归分析。支持向量机属于一般化线性分类器，这族分类器的特点是他们能够同时最小化经验误差与最大化几何边缘区，因此支持向量机也被称为最大边缘区分类器。

2、支持向量机将向量映射到一个更高维的空间里，在这个空间里建立有一个最大间隔超平面。在分开数据的超平面的两边建有两个互相平行的超平面，分隔超平面使两个平行超平面的距离最大化。假定平行超平面间的距离或差距越大，分类器的总误差越小。

3、假设给定一些分属于两类的2维点，这些点可以通过直线分割， 我们要找到一条最优的分割线，如何来界定一个超平面是不是最优的呢?

如下图：

在上面的图中，a和b都可以作为分类超平面，但最优超平面只有一个，最优分类平面使间隔最大化。 那是不是某条直线比其他的更加合适呢? 我们可以凭直觉来定义一条评价直线好坏的标准:

距离样本太近的直线不是最优的，因为这样的直线对噪声敏感度高，泛化性较差。 因此我们的目标是找到一条直线（图中的最优超平面），离所有点的距离最远。 由此， SVM算法的实质是找出一个能够将某个值最大化的超平面，这个值就是超平面离所有训练样本的最小距离。这个最小距离用SVM术语来说叫做间隔(margin) 。

二、如何计算最优超平面?

1、线性分类：

我们通常希望分类的过程是一个机器学习的过程。这些数据点并不需要是\mathbb{R}^2中的点，而可以是任意\mathbb{R}^n的点（一个超平面，在二维空间中的例子就是一条直线）。我们希望能够把这些点通过一个n-1维的超平面分开，通常这个被称为线性分类器。有很多分类器都符合这个要求，但是我们还希望找到分类最佳的平面，即使得属于两个不同类的数据点间隔最大的那个面，该面亦称为最大间隔超平面。如果我们能够找到这个面，那么这个分类器就称为最大间隔分类器。

我们从下面一个图开始：

中间那条线是wx + b =0，我们强调所有点尽可能地远离中间那条线。考虑上面3个点A、B和C。从图中我们可以确定A是×类别的，然而C我们是不太确定的，B还算能够确定。这样我们可以得出结论，我们更应该关心靠近中间分割线的点，让他们尽可能地远离中间线，而不是在所有点上达到最优。因为那样的话，要使得一部分点靠近中间线来换取另外一部分点更加远离中间线。同时这个所谓的超平面的的确把这两种不同形状的数据点分隔开来，在超平面一边的数据点所对应的 y 全是 -1 ，而在另一边全是 1 。

我们可以令分类函数：

显然，如果 f(x)=0 ，那么 x 是位于超平面上的点。我们不妨要求对于所有满足 f(x)<0 的点，其对应的="" y="" 等于="" -1="" ，而="" f(x)="">0 则对应 y=1 的数据点。如下图。

最优超平面可以有无数种表达方式，即通过任意的缩放 w 和 b 。 习惯上我们使用以下方式来表达最优超平面

|\beta\_{0} + \beta^{T} x| = 1 =1

式中 x 表示离超平面最近的那些点，也可以就可以得到支持向量的表达式为：y(wx + b) = 1，

上面说了，我们令两类的点分别为+1, -1，所以当有一个新的点x需要预测属于哪个分类的时候，我们用sgn(f(x))，就可以预测了，sgn表示符号函数，当f(x) > 0的时候，sgn(f(x)) = +1, 当f(x) < 0的时候sgn(f(x)) = –1。

通过几何学的知识，我们知道点 x 到超平面 (\beta, \beta\_{0}) 的距离为:

特别的，对于超平面, 表达式中的分子为1，因此支持向量到超平面的距离是

||w||的意思是w的二范数。

刚才我们介绍了间隔(margin),这里表示为 M, 它的取值是最近距离的2倍:

M = 2 / ||w||

最大化这个式子等价于最小化||w||, 另外由于||w||是一个单调函数，我们可以对其加入平方，和前面的系数，熟悉的同学应该很容易就看出来了，这个式子是为了方便求导。

最后最大化 M 转化为在附加限制条件下最小化函数：

即：

这是一个拉格朗日优化问题，可以通过拉格朗日乘数法得到最优超平面的权重向量W和偏置 b 。

PS

1、咱们就要确定上述分类函数f(x) = w.x + b（w.x表示w与x的内积）中的两个参数w和b，通俗理解的话w是法向量，b是截距;

2、那如何确定w和b呢？答案是寻找两条边界端或极端划分直线中间的最大间隔（之所以要寻最大间隔是为了能更好的划分不同类的点，下文你将看到：为寻最大间隔，导出1/2||w||^2，继而引入拉格朗日函数和对偶变量a，化为对单一因数对偶变量a的求解，当然，这是后话），从而确定最终的最大间隔分类超平面hyper plane和分类函数；

3、进而把寻求分类函数f(x) = w.x + b的问题转化为对w，b的最优化问题，最终化为对偶因子的求解

下面是实现：

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

cancer.data, cancer.target, random\_state=0)

svc = SVC()

svc.fit(X\_train, y\_train)

print("Accuracy on training set: {:.2f}".format(svc.score(X\_train, y\_train)))

print("Accuracy on test set: {:.2f}".format(svc.score(X\_test, y\_test)))

Accuracy on training set: 1.00

Accuracy on test set: 0.63

plt.boxplot(X\_train, manage\_xticks=False)

plt.yscale("symlog")

plt.xlabel("Feature index")

plt.ylabel("Feature magnitude")

Text(0, 0.5, 'Feature magnitude')

# Compute the minimum value per feature on the training set

min\_on\_training = X\_train.min(axis=0)

# Compute the range of each feature (max - min) on the training set

range\_on\_training = (X\_train - min\_on\_training).max(axis=0)

# subtract the min, divide by range

# afterward, min=0 and max=1 for each feature

X\_train\_scaled = (X\_train - min\_on\_training) / range\_on\_training

print("Minimum for each feature\n", X\_train\_scaled.min(axis=0))

print("Maximum for each feature\n", X\_train\_scaled.max(axis=0))

Minimum for each feature

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.]

Maximum for each feature

[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 1. 1. 1.]

# use THE SAME transformation on the test set,

# using min and range of the training set. See Chapter 3 (unsupervised learning) for details.

X\_test\_scaled = (X\_test - min\_on\_training) / range\_on\_training

svc = SVC()

svc.fit(X\_train\_scaled, y\_train)

print("Accuracy on training set: {:.3f}".format(

svc.score(X\_train\_scaled, y\_train)))

print("Accuracy on test set: {:.3f}".format(svc.score(X\_test\_scaled, y\_test)))

Accuracy on training set: 0.948

Accuracy on test set: 0.951

svc = SVC(C=1000)

svc.fit(X\_train\_scaled, y\_train)

print("Accuracy on training set: {:.3f}".format(

svc.score(X\_train\_scaled, y\_train)))

print("Accuracy on test set: {:.3f}".format(svc.score(X\_test\_scaled, y\_test)))

Accuracy on training set: 0.988

Accuracy on test set: 0.972

增大c可以显著改进模型，得到0.972的精度