从感知器到支持向量机

上一章我们介绍了感知器。作为一种二元分类器，感知器不能有效的解决线性不可分问题。其实在*第二章，线性回归*里面已经遇到过类似的问题，当时需要解决一个解释变量与响应变量存在非线性关系的问题。为了提高模型的准确率，我们引入了一种特殊的多元线性回归模型，多项式回归。通过对特征进行合理的组合，我们建立了高维特征空间的解释变量与响应变量的线性关系模型。

随着特征空间的维度的不断增多，在用线性模型近似非线性函数时，上述方法似乎依然可行，但是有两个问题不可避免。首先是计算问题，计算映射的特征，操纵高维的向量需要更强大的计算能力。然后是与算法归纳有关的问题，特征空间的维度的不断增多会导致维度灾难。从高维的特征变量中学习，要避免拟合过度，就需要呈指数级增长的训练数据。

这一章，我们将介绍一种强大的分类和回归模型，称为支持向量机（support vector machine，SVM）。首先，我们将学习高维空间的特征映射。然后，我们将介绍，在处理被映射到高维空间的数据时，支持向量机是如何缓解那些计算与综合问题的。有许多书整本整本的介绍SVM，相关的优化算法需要比前面章节里介绍其他算法更多的数学知识。我们不再用前面那些章节的小例子来演示算法，而是通过直观的案例来介绍scikit-learn如何有效的使用SVM去解决问题。

核与核方法

感知器是用超平面作决策边界对阳性和阴性类型进行分类的。其决策边界公式如下：

$$f(x)=+b$$
预测是通过下面的格式计算得出：

$$h(x) = sign(f(x))$$

其中，$是内积w^Tx$。为了与SVM的概念一致，我们要对上面的概念做一些调整。

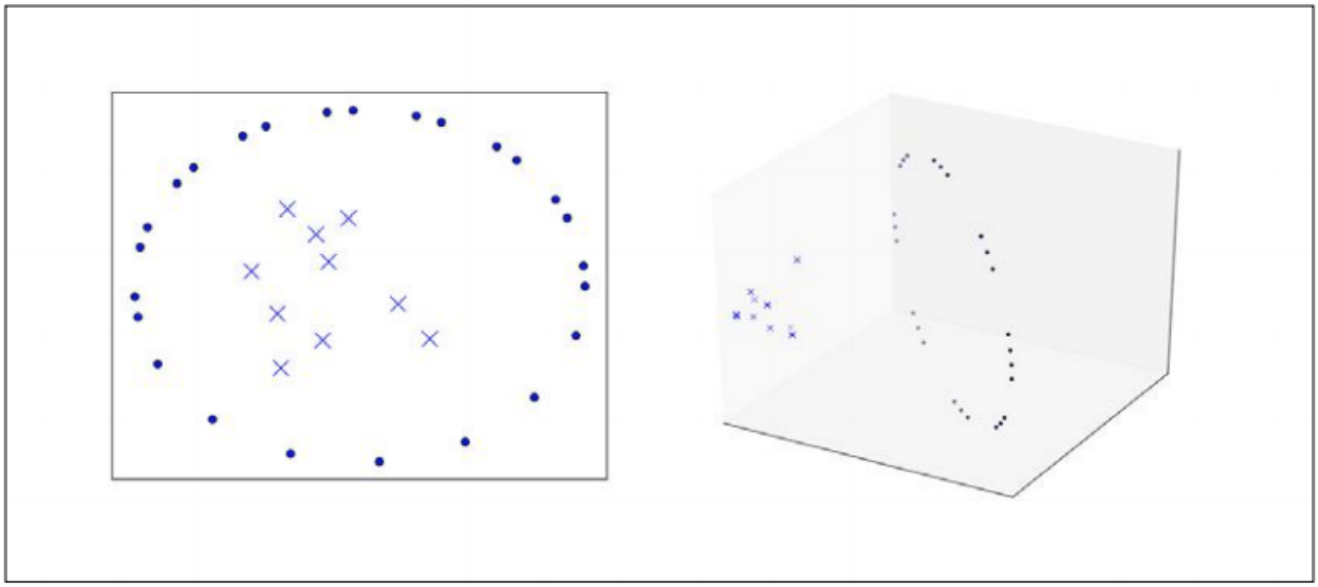
我们可以把模型写成另一种形式，其证明过程忽略。下面的模型表达式称为对偶型（dual form）。前面介绍的表达式称为原型（primal form）：

$$f(x)=+b=\sum a_iy_i+b$$
对偶型与原型的最重要区别就是原型计算模型参数（model parameters ）内积，测试样本的特征向量，而对偶型计算训练样本（training instances）的内积，测试样本的特征向量。总之，我们将利用对偶型的这个特性来解决线性不可分问题。首先，我们必须定义高维空间特征映射。

在*第二章，线性回归*介绍多元回归时，我们将特征映射成与响应变量线性相关的高维空间。映射将原来的特征进行组合，通过建立二次项增加特征的数量。这些综合特征允许我们用线性模型表示非线性函数。通常，映射表达式如下所示：

$$x \rightarrow \phi(x)$$
$$\phi : R^d \rightarrow R^D$$

下图中左边显示了一个线性不可分数据集。而右边就是将数据集映射到更高维空间后线性可分的结果。



让我们回到决策边界的对偶型，观察特征向量只以点积呈现的情况。我们可以用下面的方法将数据映射到一个高维空间：

$$f(x)=\sum a_iy_i+b

f(x)=\sum a_iy_i<\phi(x_i),\phi(x)>+b$$

这个映射允许我们表述更复杂的模型，但是它也引入了计算和综合相关的问题。映射特征向量并计算它们的点积需要极大的计算能力。

注意在第二个方程里，我们将特征向量映射到高维空间中，特征向量仍然是以点积呈现。点积是标量，如果一个标量已经被计算出来，我们就不需要去映射对应的特征向量了，这样我们在计算点击和映射特征向量这些事情上省点儿事儿。

有一种方法叫做核方法（ kernel trick）。核是一个考虑原始特征向量的函数，返回对应的映射后特征向量点积相同的值。核不直接将特征向量映射到高维空间，或者计算映射后特征向量点积。而且通过一组更有效的计算步骤得出同样的值。核更正式定义如下：

$$K(x,z) =< \phi(x), \phi(z) >$$

下面我们演示核是如何运行的。假设我们有两特征向量，x和z：

$$x = (x_1, x_2)$$
$$z = (z_1, z_2)$$

在我们的模型里面，我们想用下面的转换函数将特征向量映射到高维空间：

$$\phi(x) = x^2$$

映射后的标准化特征向量的点积等价于：

$$< \phi(x), \phi(z) >=< (x_1^2, x_2^2, \sqrt{2}x_1x_2), (z_1^2, z_2^2, \sqrt{2}z_1z_2) >$$

而下面公式的核可以产生与映射后的特征向量的点积同样的结果：

$$K(x,z)={}^2={(x_1z_1+x_2z_2)}^2=x_1^2z_1^2+2x_1z_1x_2z_2+x_2^2z_2^2

K(x,z)=<\phi(x),\phi(z)>$$

让我们把数值带入公式计算一下，让结果更清楚：

$$x = (4, 9)$$
$$z = (3, 3)$$

$$K(x,z)={}^2=
{(4\times3+9\times3)}^2=4^2\times3^2+2\times4\times3\times9\times3+9^2\times3^2=1521

<\phi(x),\phi(z)>=<(4^2,9^2,\sqrt 2\times4\times9),(3^2,3^2,\sqrt 2\times3\times3)>=1521$$

$K(x, z)$与$< \phi(x), \phi(z) >$映射后的特征向量的点积计算结果相同，但是不需要将特征向量映射到高维空间，这样减少了计算操作。这个例子只用了二维特征向量。具有中等数量特征的数据集经过映射后的特征空间将具有巨大的维度。scikit-learn提供了一些常用的核，包括多项式（polynomial），S形曲线（sigmoid），正态分布（Gaussian）和线性（linear）核。多项式核公司如下：

$$K(x, x') = (1 + x \times x')^k$$

平方核，就是多项式核的参数k是2的形式，在自然语言处理中经常用到。

S形曲线核公式如下。γ和r是可以通过交叉检验进行调整的超参数：

$$K(x, x') = tanh < \gamma(x, x') + r >$$

正态分布核是处理非线性问题的首选。正态分布核是一种径向基函数（radial basis function）(https://zh.wikipedia.org/wiki/%E5%BE%84%E5%90%91%E5%9F%BA%E5%87%BD%E6%95%映射后的特征空间里的超平面形成的决策边界与原始特征空间的超平面形成的决策边界类似。正态分布核产生的特征空间可以有无限维，这点在其他核中是不可能有的。正态分布核公式如下：

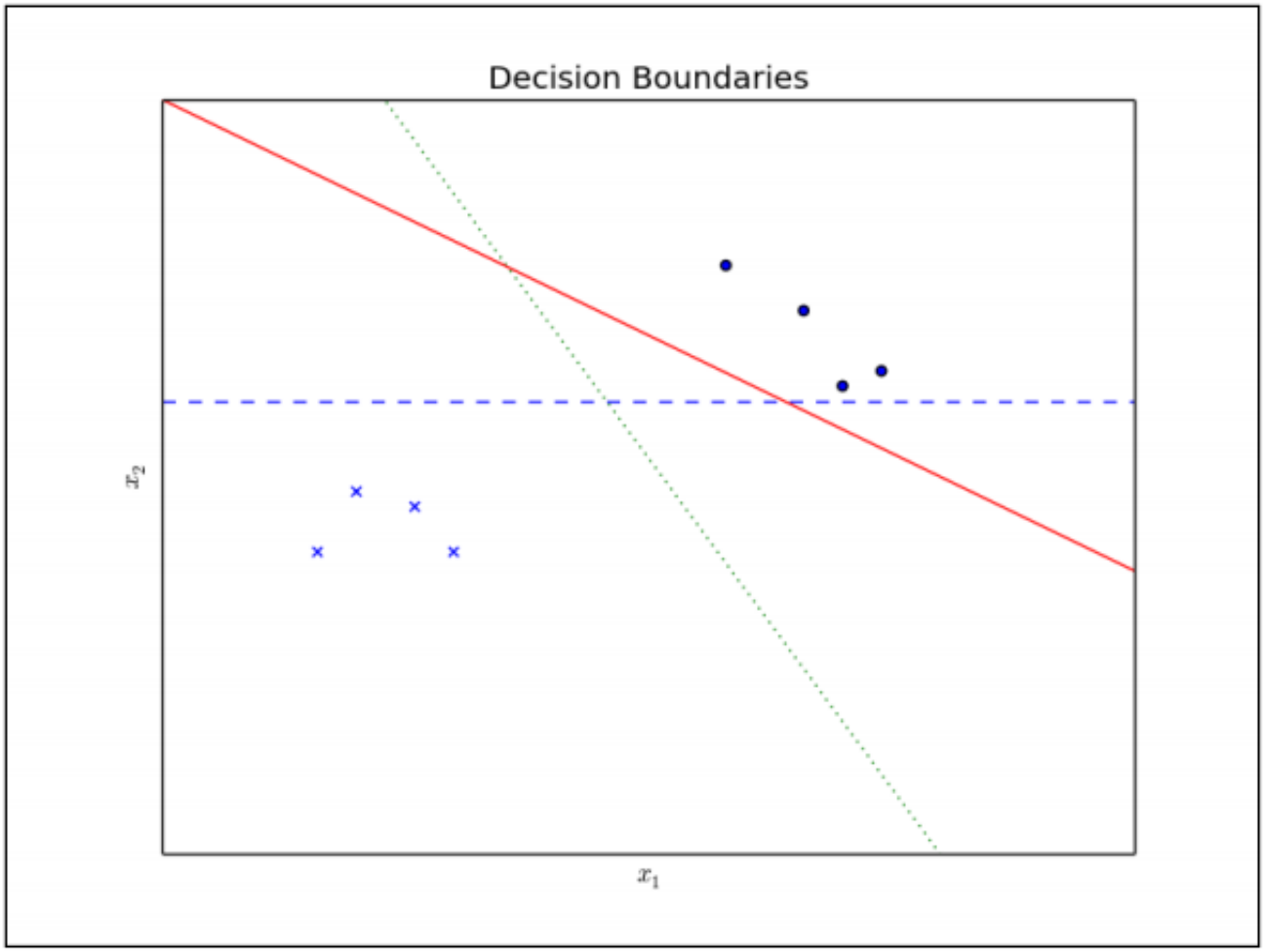
$$K(x, x') = exp(-\frac{\| x, x' \|^2}{2\sigma^2})$$

其中，σ是一个超参数。当使用SVM时，通常缩放特征是很重要的，但是使用正态分布核时缩放特征尤其重要。

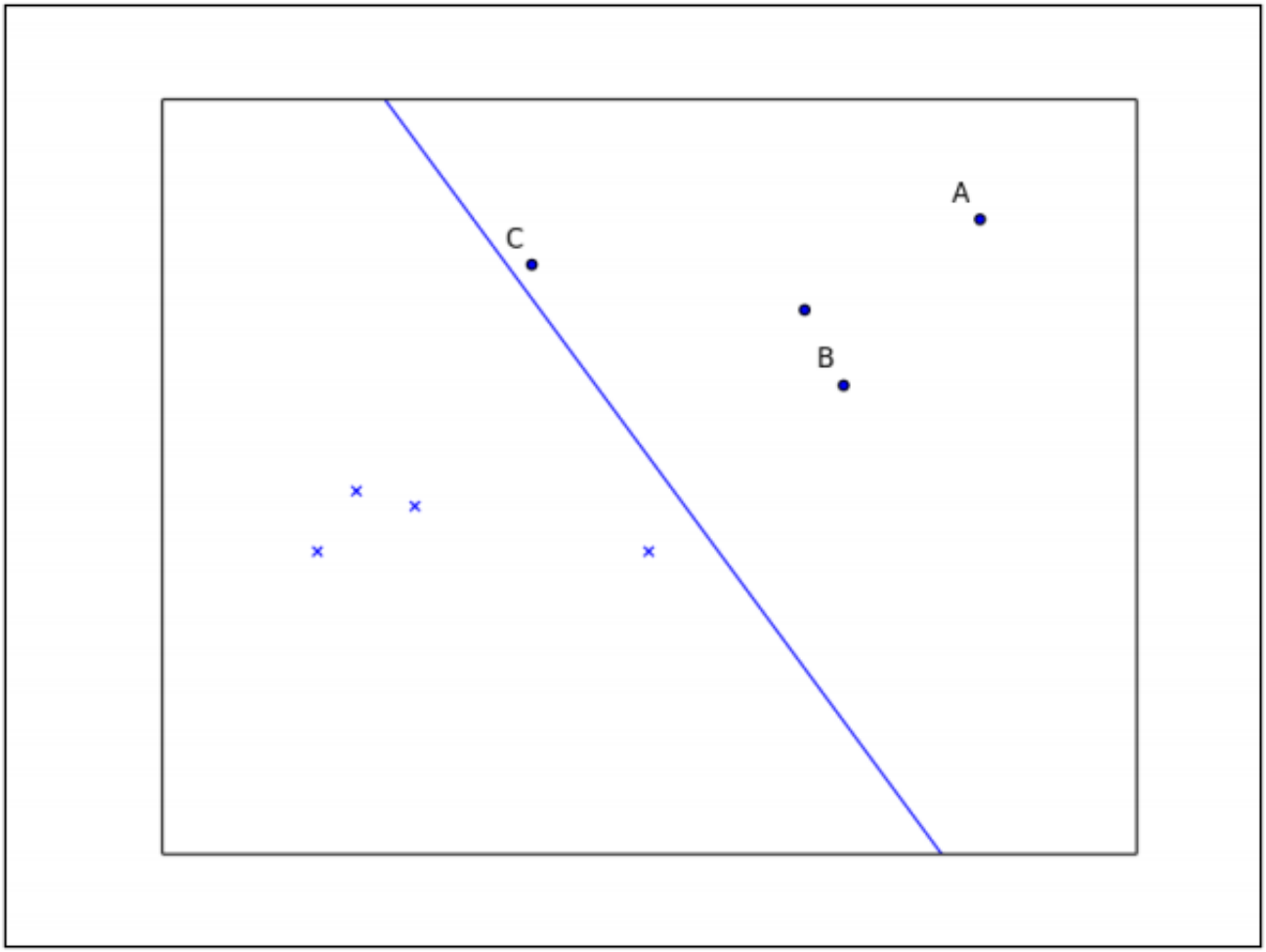
选择哪种核是很复杂的事情。理想情况下，一个核会以一种有益于解决问题的方式度量样本间的相似度。核经常用于SVM，也可用于任何能够用两个特征向量点积进行表述的模型，包括逻辑回归，感知器和主成分分析。下面，我们将解决由映射到高维特征空间引起的第二个问题：综合。

最大化间隔分类与支持向量

下图显示了两种线性可分的类型的样本集和三种可能的决策边界。所有的决策边界都可以把样本集分成阳性与阴性两种类型，感知器可以学习任何一种边界。那么，哪个决策边界对测试集数据的测试效果最好呢？



观察图中三条决策边界，我们会直观的认为点线是最佳边界。实线决策边界接近许多阳性类型样本。测试集中如果包含第一个解释变量x_1比较小的阳性样本，这个样本的类型将预测错误。虚线决策边界与大多数训练样本都很远，但是它接近一个阳性类型样本和一个阴性类型样本。下图提供了评估决策边界效果的不同视角：



假设上面画的这条线是一个逻辑回归分类器的决策边界。样本A远离决策边界，较高的概率预测它属于阳性类型。样本B仍然可以预测它属于阳性类型，但是概率可能比远离决策边界的样本A要低。最后，样本C可能以较低的概率预测它属于阳性类型，甚至训练数据集的一点儿小变化都会改变预测结果。最可靠的预测是远离决策边界的样本。我们可以用函数间隔（functional margin）来估计预测的置信水平。训练集的函数间隔公式如下：

$$funct = min(y_i f(x_i))$$

$$f(x) = +b$$

其中，y_i是样本的真实类型。样本A的函数间隔比样本C的函数间隔小。如果样本C预测错了，其函数间隔就会是负数。函数间隔为1的样本集称为支持向量（support vectors）。仅这些样本都足以定义决策边界；预测测试集样本的类型时，其他的样本可以不用。函数间隔有关的概念是空间间隔（geometric margin），或称为分离支持向量的带空间的最大宽度。空间间隔等于标准化的函数间隔。有必要用w标准化函数间隔，因为间隔在训练时是不确定的。当w是一个单位向量时，空间间隔等于函数间隔。现在我们可以把最佳决策边界定义成最大空间间隔。可以通过下面的约束条件求解最大空间间隔时的模型参数：

$$min \frac 1 n

subject to: y_i(+b) \ge 1$$

支持向量机的一个有用的属性是这个优化问题是凸包形，其唯一的局部最小值也是全局最小值。这里省略证明过程，前面的优化问题可以用对偶型调整核函数写出如下形式：

$$W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$subjectto: \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$subjectto: \alpha_i \geq 0$$

求最大化空间间隔的参数的约束是：所有的阳性样本的函数间隔最小为1，所有的阳性样本的函数间隔最多为-1，是一个二次规划问题（quadratic programming problem）。这类问题通常的解法是用序列极小优化（Sequential Minimal Optimization，SMO）算法。SMO算法把优化问题分解成一系列最小化子问题，可以通过解析方法解决。

scikit-learn文字识别

下面我们用SVM来解决分类问题。近几年，SVM已经成功解决了文字识别问题。就是给一张图片，分类器需要预测出上面的文字。文字识别是OCR（optical character-recognition，光学识别）系统的一部分。当用原始的像素强度矩阵表示特征向量时，很小的图片也会产生很高维的向量。如果类型是线性不可分，必须被映射到高维空间时，特征空间的维度会变得更庞大。SVM可以有效的处理这些问题。首先，我们用scikit-learn 训练SVM识别手写数字。然后，我们再用SVM识别照片上的字母。

识别手写数字

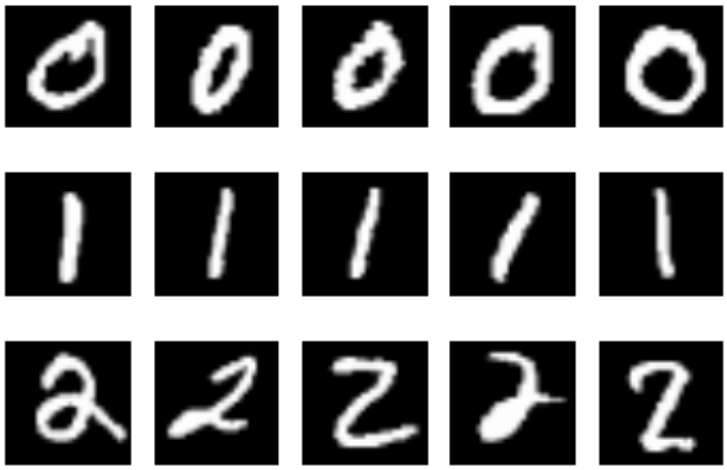
MNIST手写数字数据库（Mixed National Institute of Standards and Technology database）包含70000张手写数字图片。这些数字是通过美国国家统计局的员工和美国高校的学生收集的。每张图片都是28x28的灰度图。让我们取一些图看看：

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_mldata
import matplotlib.cm as cm

digits = fetch_mldata('MNIST original', data_home='data/mnist').data

counter = 1
for i in range(1, 4):
    for j in range(1, 6):
        plt.subplot(3, 5, counter)
        plt.imshow(digits[(i - 1) * 8000 + j].reshape((28, 28)), cmap=cm.
Greys_r)
        plt.axis('off')
        counter += 1
plt.show()
```



首先，我们加载数据。scikit-learn提供了fetch_mldata函数下载数据，然后读入对象。然后，我们创建subplot分别显示0，1，2的5个样本。

MNIST数据集可以分成60000张图片的训练集和10000张图片的测试集。这个数据集经常用于估计机器学习模型的效果；训练模型前总需要一点预处理，所以这个数据集很受欢迎。让我们用scikit-learn建一个分类器来预测图片的数字。

首先，导入需要用的模块：

In [1]:

```
from sklearn.datasets import fetch_mldata
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import scale
from sklearn.cross_validation import train_test_split
from sklearn.svm import SVC
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report
```

然后用fetch_mldata函数导入数据。再放大特征值，然后把图像调整到原点为中心的位置。再用

交叉检验分割数据集：

In [2]:

```
data = fetch_mldata('MNIST original', data_home='data\mnist')
X, y = data.data, data.target
X = X/255.0*2 - 1
X = scale(X)
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

之后，我们实例化一个SVC类的对象，就是支持向量分类器。这个对象的API和scikit-learn的其他估计器差不多；分类器用fit函数训练，然后用predict函数预测结果。如果你看过SVC的文档内容，你会发现它的参数与其他估计器要多一个。SVC最有意思的超参数是kernel，C和gamma。kernel是核函数类型。scikit-learn提供了线性，多项式，S形曲线，和RBF（径向基函数）核。C是控制正则化的参数，类似逻辑回归里的λ超参数。gamma是多项式，S形曲线，和RBF核的相关系数。超参数的设置很难恰到好处，所以我们用网格搜索来计算：

In [ ]:

```
# clf = SVC(kernel='rbf', C=2.8, gamma=.0073)
pipeline = Pipeline([
    ('clf', SVC(kernel='rbf', gamma=0.01, C=100))
])
parameters = {
    'clf__gamma': (0.01, 0.03, 0.1, 0.3, 1),
    'clf__C': (0.1, 0.3, 1, 3, 10, 30),
}
# TODO is refit true by default?
grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1, verbose=1, sc
oring='accuracy', refit=True)
grid_search.fit(X_train, y_train)
print('最佳效果：%0.3f' % grid_search.best_score_)
print('最优参数集：')
best_arameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print('\t%s: %r' % (param_name, best_parameters[param_name]))
predictios = grid_search.predict(X_test)
print(classification_report(y_test, predictions))
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
[Parallel(n_jobs=2)]: Done 1 jobs | elapsed: 7.7min
[Parallel(n_jobs=2)]: Done 50 jobs | elapsed: 201.2min
[Parallel(n_jobs=2)]: Done 88 out of 90 | elapsed: 304.8min
remaining: 6.9min
[Parallel(n_jobs=2)]: Done 90 out of 90 | elapsed: 309.2min finish
ed
Best score: 0.966
Best parameters set:
    clf__C: 3
    clf__gamma: 0.01
        precision recall f1-score support
```

```
          0.0 0.98 0.99 0.99 1758
          1.0 0.98 0.99 0.98 1968
          2.0 0.95 0.97 0.96 1727
          3.0 0.97 0.95 0.96 1803
          4.0 0.97 0.98 0.97 1714
          5.0 0.96 0.96 0.96 1535
          6.0 0.98 0.98 0.98 1758
          7.0 0.97 0.96 0.97 1840
          8.0 0.95 0.96 0.96 1668
          9.0 0.96 0.95 0.96 1729
avg / total 0.97 0.97 0.97 17500
```

模型预测的最佳综合指标是0.97，增大训练样本量可以继续改善效果，不过对硬件的要求依然很高，在我的电脑上i5 cpu 16G RAM上跑了309分钟。

自然图片文字识别

现在，我们再做一个更具挑战性的任务。我们将从自然图片中识别英文字母。Chars74K数据集(http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k)是T. E. de Campos，B. R. Babu和M. Varma为《Character Recognition in Natural Images》提供的74000张图片，包括0到9的数字和26个英文字母。下面是小写字母z的三个示例图片：

这个数据集包含不同的图片类型。本例使用从印度的Bangalore拍摄的街景里抽取的7705张文字图片。与MNIST数据集不同，Chars74K数据集里面的这些图片中的文字具有不同的字体，颜色和变化。数据下载完成后，我们将用`English/Img/GoodImg/Bmp/`里面的图片。首先我们导入模块

In [1]:

```
import os
import numpy as np
import mahotas as mh
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.cross_validation import train_test_split
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report
```

然后我们用mahotas库将图片转换成同样大小和颜色。与MNIST数据集不同，Chars74K数据集里面的图片大小不同，我们将图片转换成一边为30px的图片。

In [ ]:

```
X = []
y = []
for path, subdirs, files in os.walk('data/English/Img/GoodImg/Bmp/'):
    for filename in files:
        f = os.path.join(path, filename)
        target = filename[3:filename.index('-')]
        img = mh.imread(f, as_grey=True)
        if img.shape[0] <= 30 or img.shape[1] <= 30:
            continue
        img_resized = mh.imresize(img, (30, 30))
        if img_resized.shape != (30, 30):
            img_resized = mh.imresize(img_resized, (30, 30))
        X.append(img_resized.reshape((900, 1)))
        y.append(target)
```

最后我们把图片转换成Numpy数组，建立SVC模型，训练并预测：

In [ ]:

```
X = np.array(X)
X = X.reshape(X.shape[:2])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.1)
pipeline = Pipeline([
    ('clf', SVC(kernel='rbf', gamma=0.01, C=100))
])
parameters = {
    'clf__gamma': (0.01, 0.03, 0.1, 0.3, 1),
    'clf__C': (0.1, 0.3, 1, 3, 10, 30),
}
grid_search = GridSearchCV(pipeline, parameters, n_jobs=3, verbose=1, sco
ring='accuracy')
grid_search.fit(X_train, y_train)
print('最佳效果：%0.3f' % grid_search.best_score_)
print('最优参数集：')
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print('\t%s: %r' % (param_name, best_parameters[param_name]))
predictions = grid_search.predict(X_test)
print(classification_report(y_test, predictions))
```

这个任务比识别MNIST数据集里面的手写数据需要耗费更多的计算能力。字母的外形变化很大，因为这些字母都是从照片里提取的，不是扫描件。另外，Chars74K数据集里每个类型的训练样本数量比MNIST数据集更少。分类器的性能可以通过增加训练数据，用另外的图片预处理方法，或者用更复杂的特征表述等手段来改善。

总结

本章，我们介绍了支持向量机——一种可以弥补感知器不足的强大模型。感知器可以有效的处理线性可分问题模型，但是如果不把特征空间扩展到更高的维度，它不能表达更复杂的决策边界。但是，这样的扩展会导致计算与综合相关的问题。支持向量机用核函数修正第一个问题，可以避免特征映射的复杂计算，通过决策边界与最近样本的间隔最大化修正第二个问题。下一章，我们将介绍人工神经网络模型，和支持向量机一样，可以弥补感知器的不足。