实验环境

PyCharm-162.1237.1

TextMate ， python 2,7

一． 实验题目

1. 完成一个 bp 神经网络算法，对给定的数据进行分类

2. 完成一个 rbf 算法，对给定的数据进行分类

3. 通过调用自带的 svm 方法对给定的数据进行分类

4. 可以让把数据分为训练集和测试集测试结果准确性

5. 对不同算法进行分析

二． 实验数据

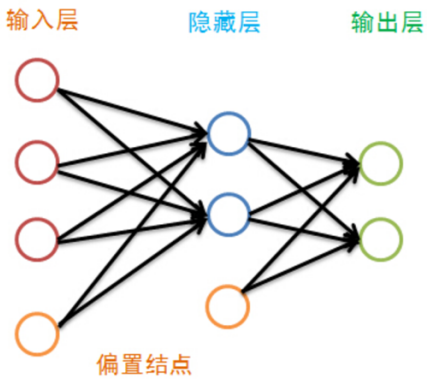
数据是位于“代码”文件夹下的 Data-Ass2.mat，是 3000*3 的数据，前两维是数据特征，后一维是数据类别

三． 实验过程

【一】BP 神经网络

（1） bp 神经网络解释

神经网络就是模拟人的大脑的神经单元的工作方式，但进行了很大的简化，神经网络由很多神经网络层构成，而每一层又由许多单元组成，第一层叫输入层，最后一层叫输出层，中间的各层叫隐藏层，在 BP 神经网络中，只有相邻的神经层的各个单元之间有联系，除了输出层外，每一层都有一个偏置结点(每个输入都是加权求和)



参照 PPT，输出层的第 j 个神经元的误差公式如下，其中 dj(n)是数据的第三维，也就是期望的输出，y 是实际输出：

$$e_j(n) = d_j(n) - y_j(n)$$

对于第 n 个样本的总共的误差公式如下:

$$\nabla \mathrm{J} = \sum_{y \epsilon \Upsilon} \frac{a^T y - b}{||y||^2} y$$

最后是我们的整体的平均错误率，公式如下

$$\mathrm{E(n)} = \frac{1}{2} \sum_{J \in C} e_j^2(n)$$

除此之外当隐藏层和输出层的时候还要选择一个激活函数
BP 利用一种称为激活函数来描述层与层输出之间的关系，从而模拟各层神经元之间的交互反应。激活函数必须满足处处可导的条件。我选择了比较常用的是一种激活函数，如下：

$$y_j(\mathrm{n}) = \varphi(\mathrm{x}) = \frac{1}{1 + \exp(-x)}$$

（注：由于这个激活函数属于（0，1），所以代码中先把所有分类为-1 的置为 0）
而其中我们的 x 就是我们的输入层的输入乘权重的累加求和，还有我们 隐含层的输出乘对应权重的累加求和。然后由于是用了一个梯度下降的思想，所以要求我们的激活函数的梯度。对应的结果如下:

$$\varphi'(x) = y_j(\mathrm{n})[1 - y_j(\mathrm{n})]$$

对于输出层的反馈函数如下：

$$\delta_j(\mathrm{n}) = (d_j(n) - y_j(n))\varphi'\big(v_j(n)\big)$$

对于隐含层反馈函数如下：

$$\delta_j(\mathrm{n}) = \varphi'\left(v_j(n)\right) \sum_k \big(d_k(n) - y_k(n)\big)\, \varphi'\big(v_k(n)\big) w_{kj}(n)$$

其中 j 表示我们对应的当前的隐含层神经元，而 k 为和 j 有关联的所有输出层神经元，而$wkj(n)$为当前 j 与 k 之间的权重。
通过不断地反馈我们需要不断地跟新 w,如下是跟新 w 的公式：

$$w_{ji}(n+1) = w_{ji}(n) + \eta \delta_j(n) y_i(n)$$

其中$yi\ (n)$表示我们当前神经元的当前输入向量的值。η为相对应的学习率。
通过不断地更新 w 以及不断地迭代，最后错误率会下降到很低，

由于给的数据比较好代码中我设定错误率到 0 停止

（2） 实验结果

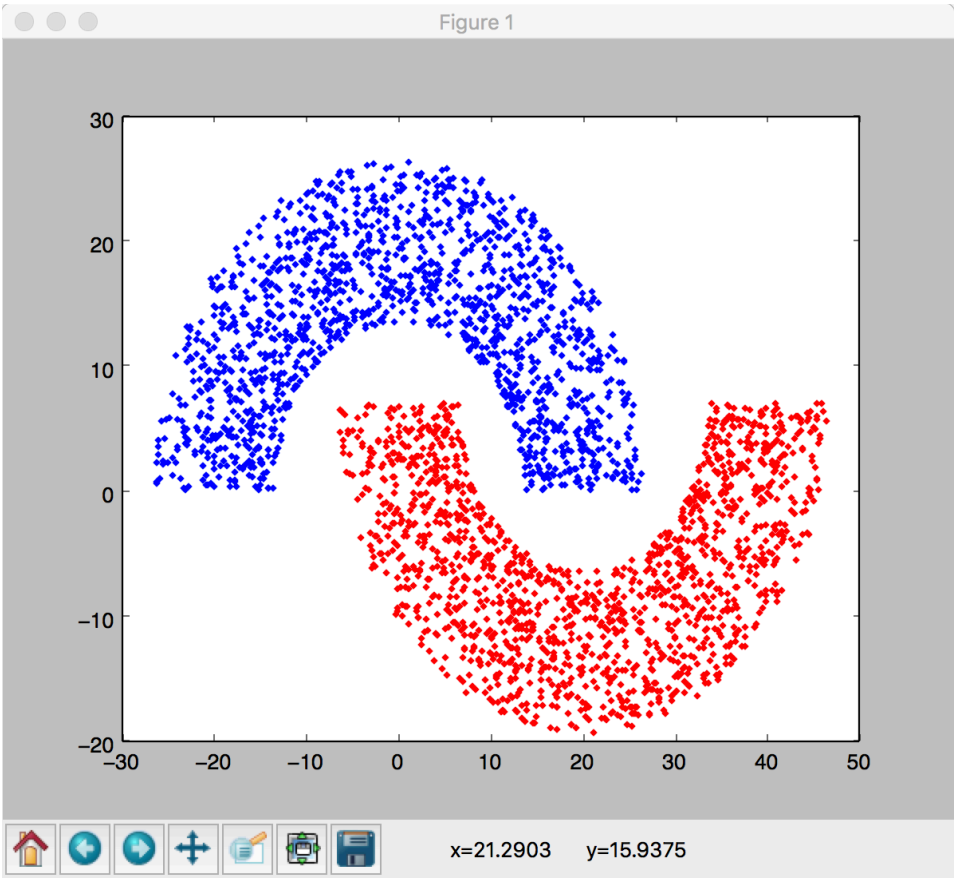


图 1 3000 点既是训练集也是测试集

从图一我们可以看出，这两类很好地分出来了

```
w1:
[ 34.51179796  -1.27059038   1.35158958]
w2:
[ 10.14587991  -0.77822768  -0.745351  ]
w3:
[ 7.19385729  0.60892868 -0.47703519]
w4:
[-10.21230526  14.44911964  -9.81861346]
number of iterations:
831
```

图 2 图一情况下的 w 值以及迭代次数

我们可以看到它迭代了 831 次

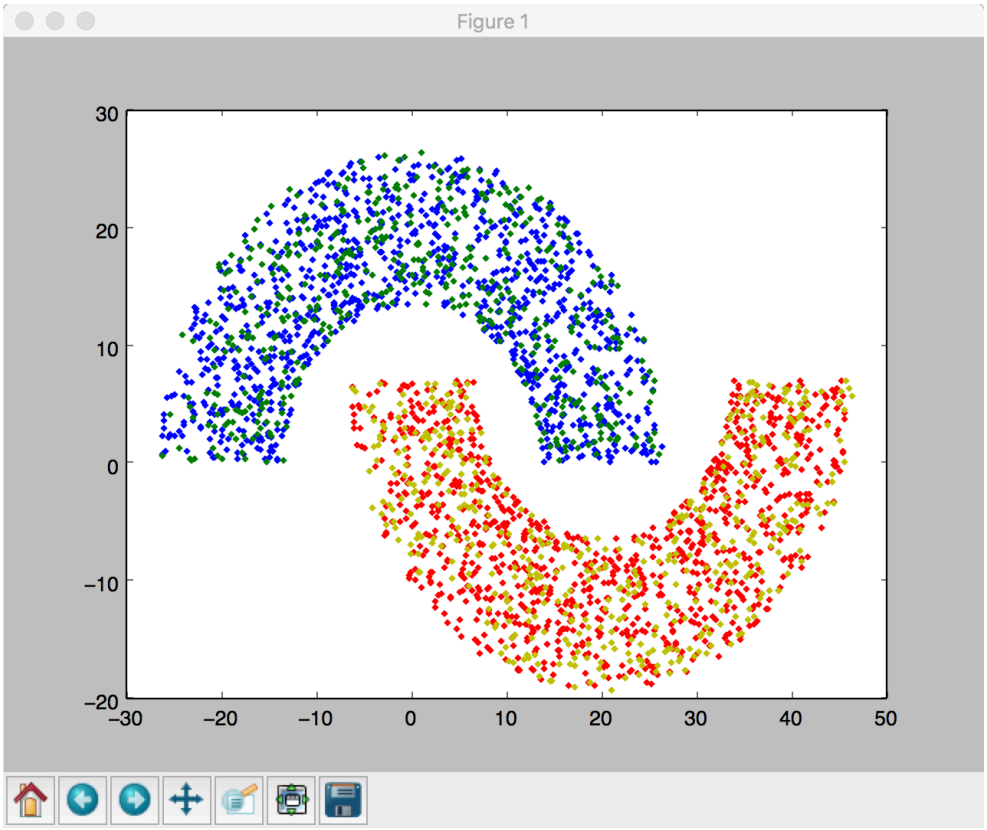


图 3 2000 是训练集，1000 个点是测试集

可以看到即使测试集不是训练街本身，正确率也是 100%

```
w1:
[ 27.92549768  -1.0531215    1.17405973]
w2:
[ 10.61112296  -0.86264577  -0.69311381]
w3:
[ 8.89588956  0.71443232 -0.60365544]
w4:
[ -8.42900504  12.47560125  -8.33454545]
number of iterations:
1080
```

图 4 图三情况下的 w 值以及迭代次数

可以看到，这里迭代了 1080 次

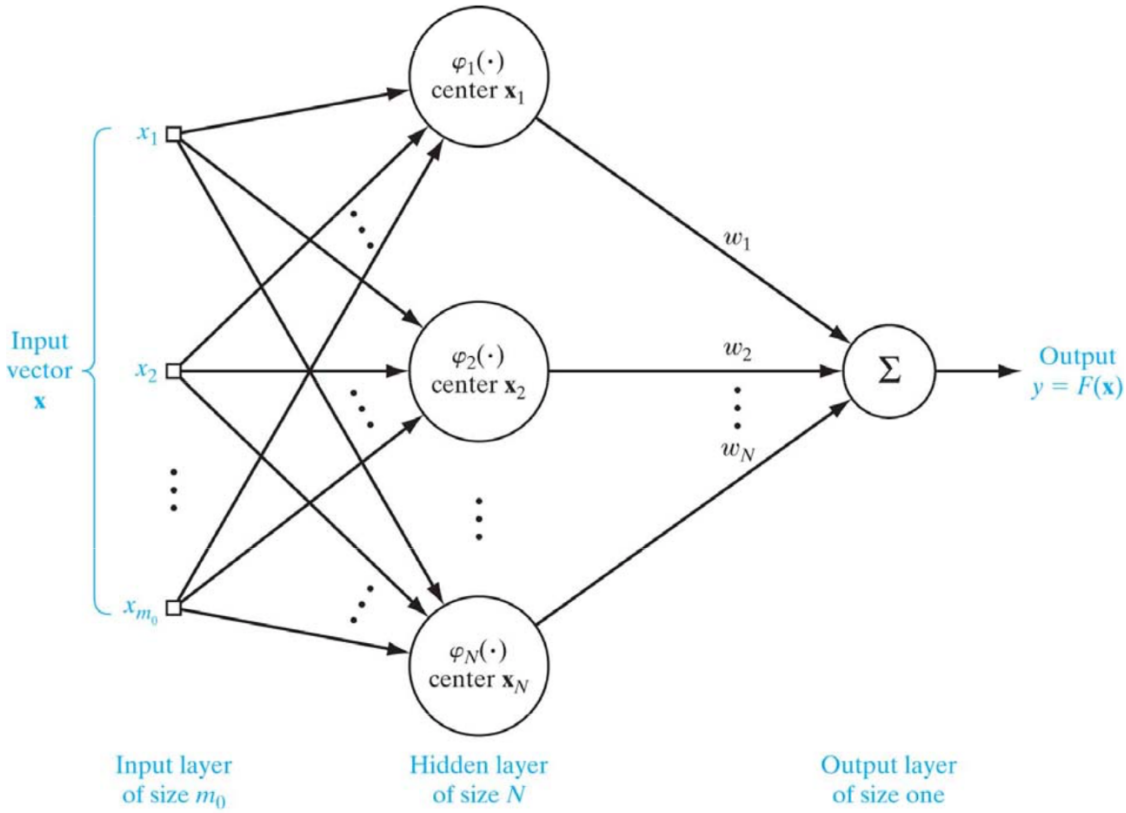
【二】RBF 网络学习

（1） 和 BP 比较

BP 是一种全局逼近网络，它有一个或多个可调参数（权值或阈值）对任何一个输出都有影响。由于对于每次输入，网络上的每一个权值都要调整，从而导致全局逼近网络的学习速度很慢。

RBF 网络是一种局部逼近网络，该算法对于输入空间的某个局部区域只有少数几个连接权值影响输出，所以比较快

网络如下图



我们的径向基函数实际上就是Φw = d，其中Φ是一个和我们训练集大小有关的一个矩阵。例如我们训练集大小为 3000，那么Φ就是一个 3000x3000 的矩阵，其中Φij = Φ(||xi − xj ||)，如果我们的Φ是一个径向基函数，那么我们的Φ矩阵就一定是可逆的，所以我们就可以用如下公式求出我们的对应的权重:

$$\mathbf{w} = \varphi^{-1}d$$

然后选取一个径向基函数：

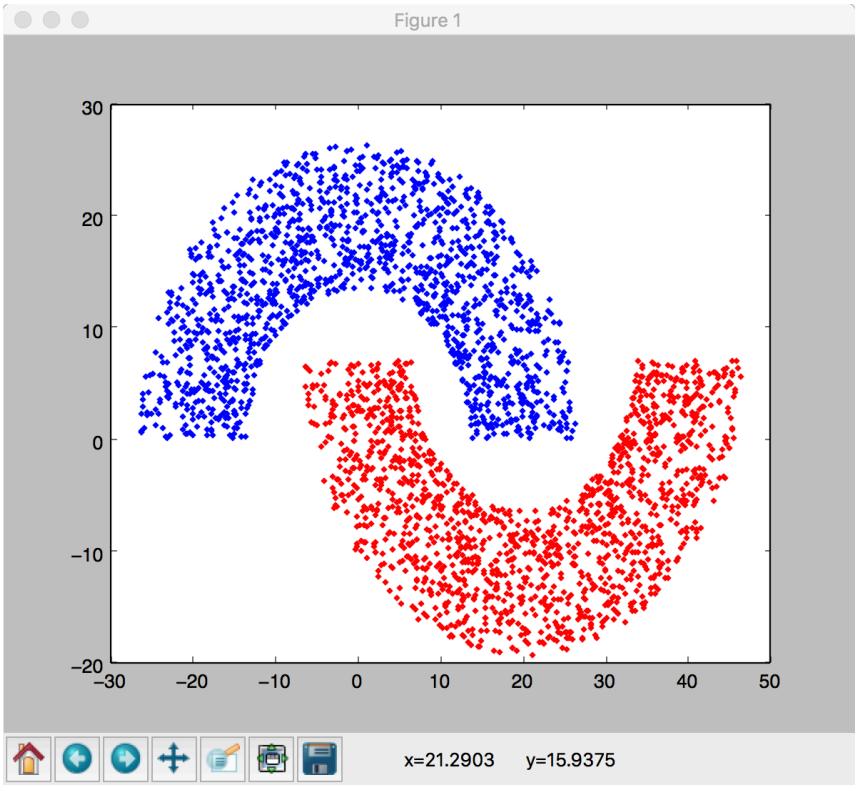
$$\varphi(\mathrm{r}) = \exp(-\frac{r^2}{2\sigma^2})$$

基于为径向基函数的差值函数为：

$$F(x)=\sum_{p=1}^{P} w_p \varphi_p(\|X-X^p\|)=w_1\varphi_1(\|X-X^1\|)+w_2\varphi_2(\|X-X^2\|)+\cdots\cdots+w_p\varphi_p(\|X-X^p\|)$$

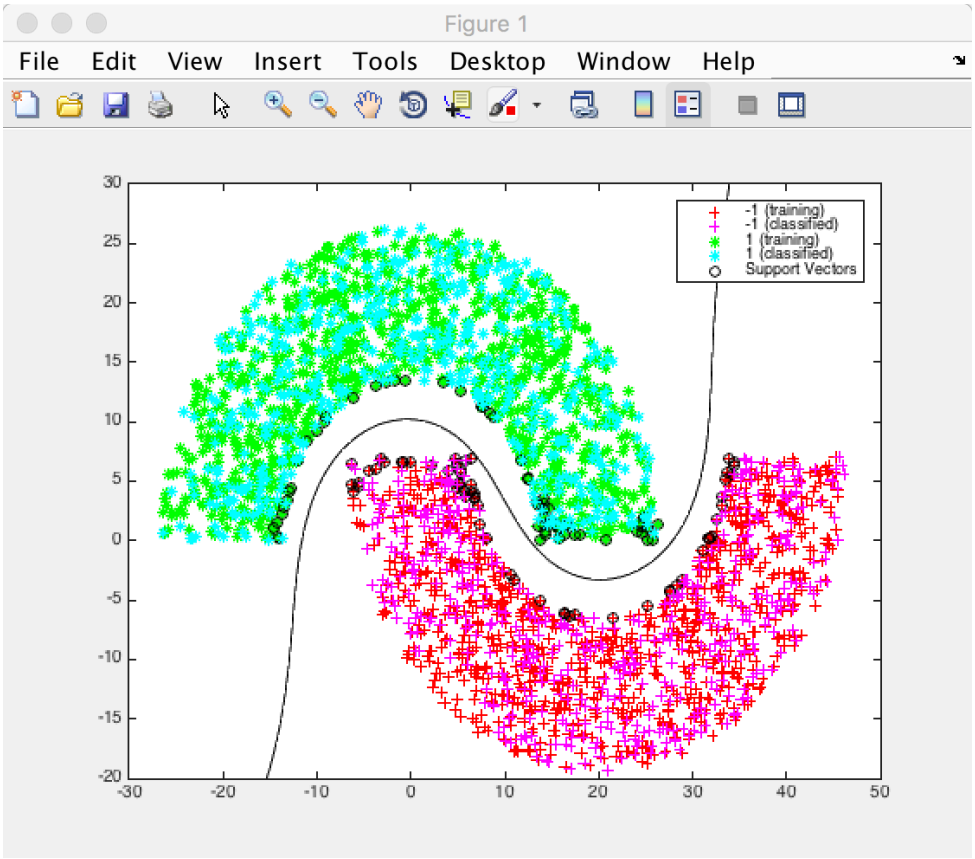
输入 X 是个 m 维的向量，样本容量为 N，N>m.

（2） 结果



【三】SVM

由于 python 用 svm 似乎还要下载一些工具包，所以还是用 matlab 实现了 ，调用方法，感觉分类速度比前两种快了好多，直接看结果：



图六. SVM 分类图

四． 收获与感悟

a） 经过这次实验首先我更加熟悉了 python 的使用

b） 我了解了 bp 神经网络算法的原理和使用

c） 我了解了 rbf 算法的原理和使用

五． 代码

位于“代码”文件夹下

六． 参考博客

http://www.cnblogs.com/fengfenggirl/p/bp_network.html

http://blog.sina.com.cn/s/blog_88f0497e0102v79c.html

http://www.cnblogs.com/wentingtu/archive/2012/06/05/2536425.html

http://www.cnblogs.com/zhangchaoyang/articles/2591663.html