原理

推导过程直接读：《统计学习方法》或者码农场的总结
过程十分繁琐并且涉及大量公式，这里也不重复。这里主要把中心思想列出来。

线性可分学习算法

输入：线性可分训练数据集 $T=(x_1,y_1),(x_2,y_2),\cdots(x_n,y_n)$ ，其中，$x_i \subseteq R^n$，$y_i \subseteq [-1,+1], i=1,2\cdots N$;
原优化问题是：

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2$$
$$\text{s.t.} \quad y_i(w\cdot x_i+b)-1\geqslant 0, \quad i=1,2,\cdots,N$$

加入拉格朗日乘子和一系列推导后，变成下面：

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(x_i\cdot x_j)-\sum_{i=1}^{N}\alpha_i$$
$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i y_i=0$$
$$\alpha_i\geqslant 0, \quad i=1,2,\cdots,N$$

又考虑到现实中没有完全的线性可分的数据，所以加入软间隔后原始优化问题变为：

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2+C\sum_{i=1}^{N}\xi_i$$
$$\text{s.t.} \quad y_i(w\cdot x_i+b)\geqslant 1-\xi_i, \quad i=1,2,\cdots,N$$
$$\xi_i\geqslant 0, \quad i=1,2,\cdots,N$$

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(x_i\cdot x_j)-\sum_{i=1}^{N}\alpha_i$$
$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i y_i=0$$
$$0\leqslant\alpha_i\leqslant C, \quad i=1,2,\cdots,N$$

C 值大时对误分类的情况惩罚越大，是同时使得间隔尽量大和误分类尽量少的系数。
求得最优解 $\alpha=(\alpha_1,\alpha_2\cdots\alpha_N)^T$ ，即可求：

$$w^* = \sum_{i=1}^{N} \alpha_i^* y_i x_i$$

$$b^* = y_j - \sum_{i=1}^{N} \alpha_i^* y_i (x_i \cdot x_j)$$

最终可以求得分离超平面：

$$w^* \cdot x + b^* = 0$$

$$f(x) = sign(w^* \cdot x + b^*)$$

非线性支持向量机

非线性支持向量机只是上面的一种扩展，原始优化函数实际差不多：

$$\min_{\alpha} \quad \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{N} \alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$0 \leqslant \alpha_i \leqslant C, \quad i = 1, 2, \cdots, N$$

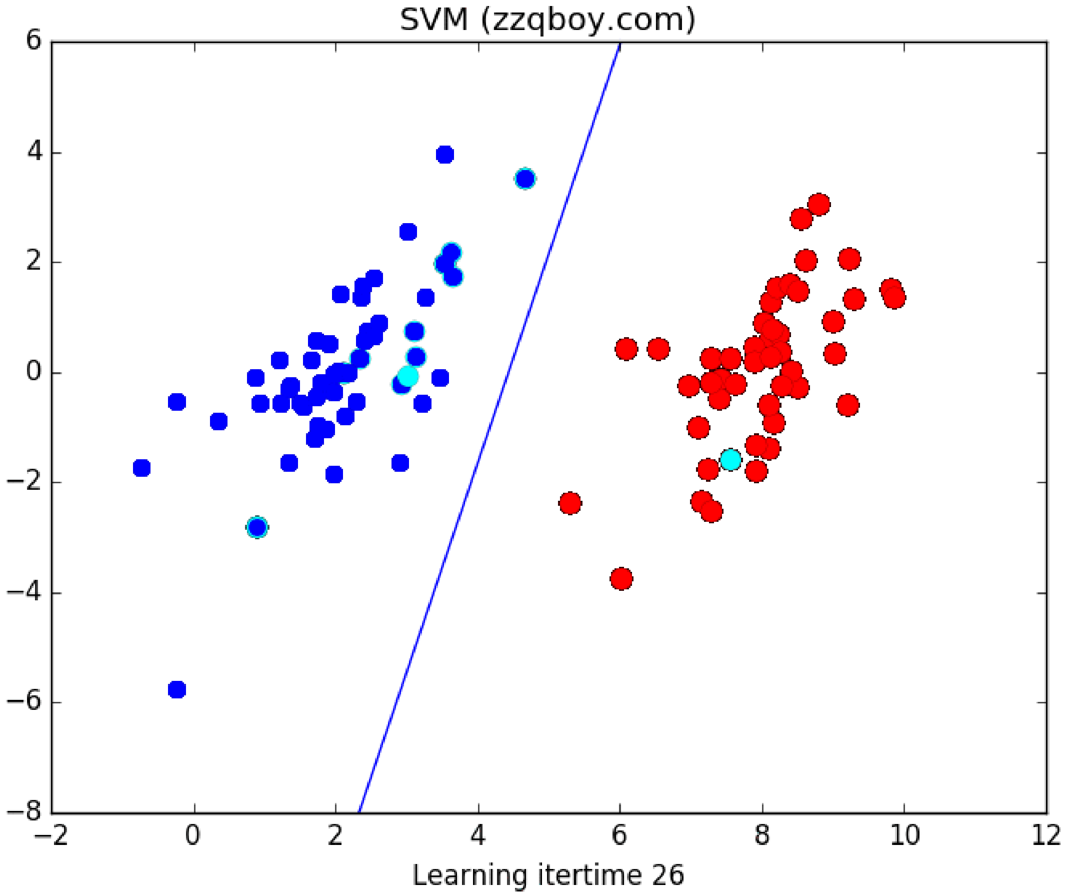
可以看出，线性支持机不过是$K(x_i, x_j) = x_i \cdot x_j$的时候而已。
其他推导过程就不说了，软间隔最大化和SMO算法（求解算法）可以在一开始的教程细读，否则这里又变繁琐了。
另外说一下α的意义，个人理解当$0 < \alpha < C$时，点正好在间隔边界；当$\alpha = 0$（C）时，点的分类位置在间隔边界不偏向超平面的一侧（偏向超平面的一侧）；

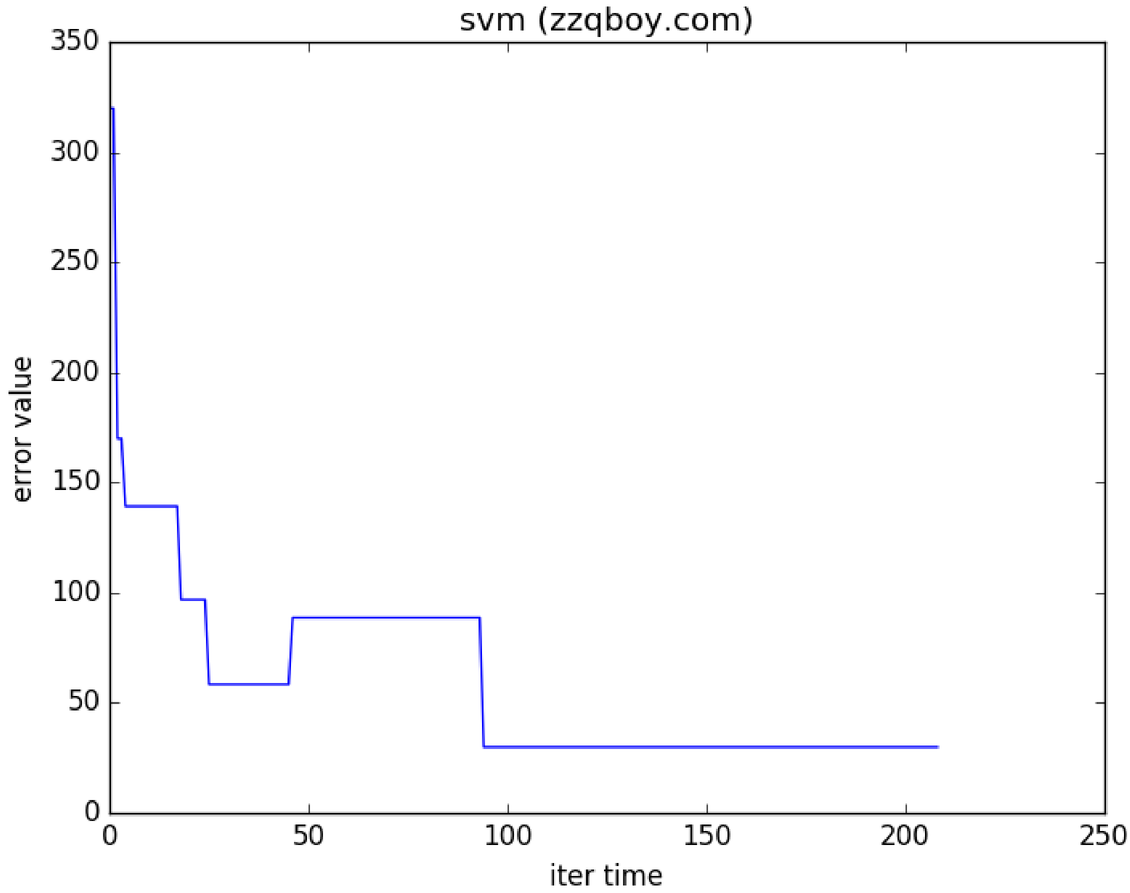
可视化

既然是原创文章，就得有些自己的东西。码农场说过：

> 比起枯燥的公式，我更喜欢写Python实现，再通过Matplotlib这个强大的作图库可视化出来

其实我也觉得这样做很直观，下面是我在《机器学习实战》的代码上修改，再结合 码农场的教程 画出的图。


SVM (zzqboy.com)
Learning itertime 26
6
4
2
0
−2
−4
−6
−8
−2
0
2
4
6
8
10
12


svm (zzqboy.com)
error value
iter time
350
300
250
200
150
100
50
0
0
50
100
150
200
250

代码

画图的代码如下，全部代码

```python
def draw_svm_learning():
    ### 开始拟合
    dataArr, labelArr = loadDataSet('testSet.txt')
    b, alphas, choose_all_x, all_alpha, all_b = smoP(dataArr, labelArr

    ## 找到使得直线方程改变的两个点 alpha1 alpha2
    change_index = [index for index, i in enumerate(choose_all_x) if l
    change_x, change_alpha, change_b = [], [], []
    for i in change_index:
        change_x.append(choose_all_x[i])
        change_alpha.append(all_alpha[i])
        change_b.append(all_b[i])
    print len(choose_all_x)
    print len(change_x)

    ### 下面开始画图
    fig = plt.figure(121)
    ax = plt.axes(xlim=(-2, 12), ylim=(-8, 6))
    line, = ax.plot([], [])

    # 标签为-1的点
    xcord0 = [dataArr[i][0] for i in range(len(labelArr)) if labelArr[
    ycord0 = [dataArr[i][1] for i in range(len(labelArr)) if labelArr[
    # 标签为1的点
    xcord1 = [dataArr[i][0] for i in range(len(labelArr)) if labelArr[
    ycord1 = [dataArr[i][1] for i in range(len(labelArr)) if labelArr[

    # 更新每次的超平面方程
    def animate(time):
        # time 代表迭代次数，也是帧数
        label = u'Learning itertime {0}'.format(time)
        ax.set_xlabel(label)
        a, b = change_alpha[time], change_b[time]
        w = calcWs(a, dataArr, labelArr)
        choose_x = change_x[time]
        # 画出每次优化的两个点和其他数据点
        for i in range(len(labelArr)):
            xPt = dataArr[i][0]
            yPt = dataArr[i][1]
            label = labelArr[i]
            if i in choose_x:
                continue
            if (label == -1):
                ax.scatter(xPt, yPt, marker='o', s=60, linewidths=0.01
            else:
                ax.scatter(xcord1, ycord1, marker='o', s=90, c='red',l
        for i in choose_x:
            ax.scatter(dataArr[i][0], dataArr[i][1], marker='o', s=90,
        # 画出直线
        w0 = w[0][0]
        w1 = w[1][0]
        b = float(b)
        x = arange(-2.0, 12.0, 0.1)
```

```
            y = (-w0 * x - b) / w1
            line.set_data(x, y)
            plt.title(u'SVM (zzqboy.com)')
            return line, ax

    anim = animation.FuncAnimation(fig, animate, frames=len(change_x),
    # plt.show()
    anim.save('svm.gif', fps=2, writer='imagemagick')
```

这里讲一下，这部分的代码

```
change_index = [index for index, i in enumerate(choose_all_x) if len(i
    change_x, change_alpha, change_b = [], [], []
    for i in change_index:
        change_x.append(choose_all_x[i])
        change_alpha.append(all_alpha[i])
        change_b.append(all_b[i])
    print len(choose_all_x)
    print len(change_x)
```

这里的结果是209和29，由于随机性，所以你的运行结果不一定是这个。

因为迭代次数是比较多的，我不可能全部保存在一个gif里，所以我只画出实际上起作用的时刻，而这样的时刻很少，可以看到大概只有$\frac{1}{10}$。这也在一定程度上解释了“炼金术”这个词。

有兴趣的可以修改一下，可以看到有很大一部分是没有找到第二个优化的α