Coursera 機器學習技法

January 2, 2016

1 Week 1

1.1 SVM 解决的问题

动机：提高线性模型对随机噪声的抵抗力，减少 overfit 的可能——选择 margin 最大的分类面。

找最大 margin 即找同时满足两个条件的分类面：

1. 正确分类所有的点
2. 所有数据点中离分类面最近的点到分类面的距离最大

设 x', x'' 在平面 $\mathbf{w}^T\mathbf{x}'$ 上，则 $\mathbf{w}^T\mathbf{x}' = -b$，$\mathbf{w}^T\mathbf{x}'' = -b$ 故 $\mathbf{w}^T\mathbf{x}' - \mathbf{x}'' = 0$，即 $\mathbf{w}$ 为法线。所以求任意一点 $\mathbf{x}$ 到平面距离即求 $\mathbf{x} - \mathbf{x}'$ 在法线上的投影：

$$d = \frac{1}{\|\mathbf{w}\|}|\mathbf{w}^T\mathbf{x} + b|$$

而且要分类正确，所以 $y_n(\mathbf{w}^T\mathbf{x}_n + b) > 0$，上式简化为

$$dist = \frac{1}{\|\mathbf{w}\|}y_n(\mathbf{w}^T\mathbf{x}_n + b)$$

问题转化为

$$\begin{aligned} \max_{b,\mathbf{w}} \min_{n=1,\dots,N} \quad & \frac{1}{\|\mathbf{w}\|}y_n(\mathbf{w}^T\mathbf{x}_n + b) \\ \text{subject to} \quad & y_n(\mathbf{w}^T\mathbf{x}_n + b) > 0 \quad \text{for all } n \end{aligned} \tag{1}$$

注意到，超平面方程 $\mathbf{w}^T\mathbf{x} + b = 0$ 两边同乘一个倍数后保持不变，所以上式中可以通过 scaling 来使得 $\min_{n=1,\dots,N} y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$，所以问题变为

$$\begin{aligned} \max_{b,\mathbf{w}} \quad & \frac{1}{\|\mathbf{w}\|} \\ \text{subject to} \quad & \min_{n=1,\dots,N} y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1 \end{aligned} \tag{2}$$

然后把约束条件放宽为对所有 $n = 1, \dots, N$ 满足 $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$。这样做不会改变原问题的最优解，因为若最优解 $b, \mathbf{w}$ 使得所有 $n = 1, \dots, N$ 都有 $y_n(\mathbf{w}^T\mathbf{x}_n + b) = t > 1$，则可将 $b, \mathbf{w}$ scale 为原来的 $\frac{1}{t}$ 让它们满足原约束，并得到更优解 $\frac{b}{t}, \frac{\mathbf{w}}{t}$，这就与 $b, \mathbf{w}$ 是最优解的假设矛盾。所以最优解只可能在原约束下取到。

问题再次转变为

$$\begin{aligned} \min_{b,\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{subject to} \quad & y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \text{for all } n \end{aligned} \tag{3}$$

1.2 线性 SVM 和 QP

以上线性 SVM 的最优化问题即 quadratic programming 问题。一般形式为

$$\begin{aligned} \min_{\mathbf{u}\in\mathbb{R}^L} \quad & \frac{1}{2}\mathbf{u}^T\mathbf{Q}\mathbf{u} + \mathbf{p}^T\mathbf{u} \\ \text{subject to} \quad & \mathbf{a}_m^T\mathbf{u} \geq c_m \quad (m = 1, \ldots, M) \end{aligned} \tag{4}$$

当 $\mathbf{Q}$ 正定时 QP 问题为凸。QP 可以直接用凸优化的库来计算。通常需要改写为矩阵形式:

$$\begin{aligned} \min_{\mathbf{u}\in\mathbb{R}^L} \quad & \frac{1}{2}\mathbf{u}^T\mathbf{Q}\mathbf{u} + \mathbf{p}^T\mathbf{u} \\ \text{subject to} \quad & \mathbf{A}\mathbf{u} \geq \mathbf{c} \end{aligned} \tag{5}$$

然后扔给 QP solver 算算算……凸优化还是要学习的……不然不知道原理啊心慌慌。

对线性 hard-margin SVM 问题，QP 的这些参数分别为

$$\begin{aligned} \mathbf{Q} &= \begin{bmatrix} 0 & \mathbf{0}_d^T \\ \mathbf{0}_d & \mathbf{I}_d \end{bmatrix} \\ \mathbf{A} &= \begin{bmatrix} y_1 & y_1\mathbf{x}_1^T \\ \vdots & \vdots \\ y_N & y_N\mathbf{x}_N^T \end{bmatrix} \\ \mathbf{p} &= \mathbf{0}_{d+1} \\ \mathbf{c} &= \mathbf{1}_N \end{aligned} \tag{6}$$

此时留给 QP 的 $\mathbf{Q}$ 矩阵维度为 $d+1$，与数据维度相关。

1.3 SVM 的优势

总的来讲，SVM 相当于在保证 $E_{in} = 0$ 的情况下优化一个惩罚项，而且较大的 margin 能够有效地限制模型的复杂度，同时令交叉验证时得到的 E_{CV} 也会受到限制。

LFD 书上给出了 margin 为 ρ 的 SVM 的 VC 维

$$d_{VC}(\rho) \leq \left\lceil \frac{R^2}{\rho^2} \right\rceil + 1$$

其中 R 为距原点最远的点到原点的距离。

证明看 LFD 书和课后练习。大致思路是分别讨论 SVM 能 shatter 的点数 N 的奇偶性，然后根据每个点到分类面的距离至少为 ρ 得到不等式并求和，右边用 Cauchy-Schwarz 不等式放大，再用概率证明存在，得到右边的上界即可。

做 leave-one-out 交叉验证时，$E_{CV} = \frac{1}{N}\sum_{n=1}^{N} e_n$，如果去掉的点不在 margin 边界上，则肯定分类正确，所以 $e_n = 0$，而 margin 边界上的点有 $e_n \leq 1$，所以 SVM 的

$$E_{CV} = \frac{1}{N}\sum_{n=1}^{N} e_n \leq \frac{\#\text{ support vectors}}{N}$$

这个上界不依赖数据维度，使得 SVM 可以配合特征变换，在高维空间得到非线性的分类边界，同时限制复杂度，降低普通线性模型加特征变换可能产生的过拟合问题。

特征变换之后的 SVM 问题为

$$\begin{aligned} \min_{\tilde{b},\tilde{\mathbf{w}}} \quad & \frac{1}{2}\tilde{\mathbf{w}}^T\tilde{\mathbf{w}} \\ \text{subject to} \quad & y_n(\tilde{\mathbf{w}}^T\mathbf{z}_n + \tilde{b}) \geq 1 \quad \text{for all } n \end{aligned} \tag{7}$$

1.4 Dual SVM 的动机

特征变换后的 SVM QP 问题有 $\tilde{d}+1$ 个变量: $\tilde{b}$ 和 $\tilde{\mathbf{w}}$ 和 N 个约束，依赖于训练数据的维度和变换后空间的维度。当 $\tilde{d}$ 太大时难以计算。所以，理想的情形是，变换之后的计算不依赖于 $\tilde{d}$。根据凸优化，可通过解原问题的 Lagrange dual 来获得原问题最优解。

1.5 QP Lagrange Dual

假设待解问题为

$$\begin{aligned} \min_{\mathbf{u}\in\mathbb{R}^L} \quad & \frac{1}{2}\mathbf{u}^T\mathbf{Q}\mathbf{u} + \mathbf{p}^T\mathbf{u} \\ \text{subject to} \quad & \mathbf{a}^T\mathbf{u} \geq c \end{aligned} \tag{8}$$

相关联的优化问题为

$$\min_{\mathbf{u}\in\mathbb{R}^L} \frac{1}{2}\mathbf{u}^T\mathbf{Q}\mathbf{u} + \mathbf{p}^T\mathbf{u} + \max_{\alpha\geq 0}\alpha(c - \mathbf{a}^T\mathbf{u}) \tag{9}$$

LFD exercise 8.9 证明了此问题和原问题的最优解相同，都能满足原问题的约束。直观来看嘛，上面第二个式子构造出来一个新函数，里面嵌入原问题的约束。最优解肯定要满足约束，所以后面那个 max 一坨肯定是 0，也就是说，构造出来的新函数让约束得到满足，同时不改变原函数的取值。

将上式改写为 Lagrange 函数

$$\mathcal{L}(\mathbf{u},\alpha) = \frac{1}{2}\mathbf{u}^T\mathbf{Q}\mathbf{u} + \mathbf{p}^T\mathbf{u} + \alpha(c - \mathbf{a}^T\mathbf{u})$$

则要优化的问题为

$$\min_{\mathbf{u}}\max_{\alpha\geq 0}\mathcal{L}(\mathbf{u},\alpha)$$

此问题的 strong dual 为

$$\max_{\alpha \geq 0} \min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \alpha)$$

交换最大最小之后，就可以在没有约束的情况下求解内层，然后再解外层。

多个约束存在时，要将每个约束都用 Lagrange 乘数加入 Lagrange 函数，即所谓的 Karush-Kühn-Tucker（KKT）定理。求得的最优解满足 KKT 条件：

设待优化问题为

$$\begin{aligned} \min_{\mathbf{u} \in \mathbb{R}^L} \quad & \frac{1}{2}\mathbf{u}^T \mathbf{Q} \mathbf{u} + \mathbf{p}^T \mathbf{u} \\ \text{subject to} \quad & \mathbf{a}_m^T \mathbf{u} \geq c_m \quad (m = 1, \ldots, M) \end{aligned} \tag{10}$$

定义 Lagrange 函数

$$\mathcal{L}(\mathbf{u}, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{u}^T \mathbf{Q} \mathbf{u} + \mathbf{p}^T \mathbf{u} + \sum_{m=1}^{M} \alpha_m (c_m - \mathbf{a}_m^T \mathbf{u})$$

则 $\mathbf{u}^*$ 是原问题最优解 iff $\mathbf{u}^*, \alpha^*$ 是 dual

$$\max_{\boldsymbol{\alpha} \geq \mathbf{0}} \min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \boldsymbol{\alpha})$$

的最优解。而且 dual 的最优解满足

1. Primal 和 dual 的约束：$\mathbf{a}_m^T \mathbf{u}^* \geq c_m$ 和 $\alpha_m \geq 0$。其中 $m = 1, \ldots, M$
2. Complementary slackness：$\alpha_m^* (\mathbf{a}_m^T \mathbf{u}^* - c_m) = 0$
3. Stationarity w.r.t. $\mathbf{u}$：$\nabla_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \boldsymbol{\alpha})|_{\mathbf{u}=\mathbf{u}^*, \boldsymbol{\alpha}=\boldsymbol{\alpha}^*} = 0$

1.6 Dual SVM

根据 SVM 的优化目标和约束构造出对应的 dual 即可。

根据 SVM 的问题

$$\begin{aligned} \min_{b, \mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^T \mathbf{w} \\ \text{subject to} \quad & y_n (\mathbf{w}^T \mathbf{x}_n + b) \geq 1 \quad \text{for all } n \end{aligned} \tag{11}$$

得到 Lagrange 函数

$$\begin{aligned} \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}) &= \frac{1}{2}\mathbf{w}^T \mathbf{w} + \sum_{n=1}^{N} \alpha_n (1 - y_n (\mathbf{w}^T \mathbf{x}_n + b)) \\ &= \frac{1}{2}\mathbf{w}^T \mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{w}^T \mathbf{x}_n - b \sum_{n=1}^{N} \alpha_n y_n + \sum_{n=1}^{N} \alpha_n \end{aligned} \tag{12}$$

先解内层没有约束的最小化问题：

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial b} &= -\sum_{n=1}^{N} \alpha_n y_n \\ \frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n \end{aligned} \tag{13}$$

得到内层最优需要满足的条件

$$\begin{aligned}\sum_{n=1}^{N}\alpha_n y_n &= 0\\ \mathbf{w} &= \sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n\end{aligned} \tag{14}$$

这里得到了一个关于 α 的限制，因为如果 α 不满足此条件，$-b\sum_{n=1}^{N}\alpha_n y_n$ 可以通过取 b 的值将 $\mathcal{L}$ 变为 $-\infty$。

将内层问题得到的 $\mathbf{w}$ 回带入原问题，得

$$\begin{aligned}&\frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{n=1}^{N}\alpha_n y_n \mathbf{w}^T\mathbf{x}_n\\ =&\frac{1}{2}\sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n^T \sum_{m=1}^{N}\alpha_m y_m \mathbf{x}_m - \sum_{n=1}^{N}\alpha_n y_n \sum_{m=1}^{N}\alpha_m y_m \mathbf{x}_m^T\mathbf{x}_n\\ =&\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T\mathbf{x}_m - \sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T\mathbf{x}_m\\ =&-\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T\mathbf{x}_m\end{aligned} \tag{15}$$

最终 Lagrange 函数变为

$$\mathcal{L}(\alpha) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T\mathbf{x}_m + \sum_{n=1}^{N}\alpha_n$$

取个负号，目标问题变为

$$\begin{aligned}\min_{\alpha\in\mathbb{R}^N}\quad & \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T\mathbf{x}_m - \sum_{n=1}^{N}\alpha_n\\ \text{subject to}\quad & \sum_{n=1}^{N} y_n\alpha_n = 0\\ & \alpha_n \geq 0 (n = 1,\dots,N)\end{aligned} \tag{16}$$

现在终于编程了一般的 QP 问题，直接扔给 QP solver 即可，参数为：

$$\begin{aligned}\mathbf{Q}_D &= \begin{bmatrix} y_1y_1\mathbf{x}_1^T\mathbf{x}_1 & \cdots & y_1y_N\mathbf{x}_1^T\mathbf{x}_N\\ y_2y_1\mathbf{x}_2^T\mathbf{x}_1 & \cdots & y_2y_N\mathbf{x}_2^T\mathbf{x}_N\\ \vdots & \vdots & \vdots\\ y_Ny_1\mathbf{x}_N^T\mathbf{x}_1 & \cdots & y_Ny_N\mathbf{x}_N^T\mathbf{x}_N\end{bmatrix}\\ \mathbf{A}_D &= \begin{bmatrix}\mathbf{y}^T\\ -\mathbf{y}^T\\ \mathbf{I}_{N\times N}\end{bmatrix}\\ \mathbf{p} &= -\mathbf{1}_N\\ \mathbf{c} &= \mathbf{0}_{N+2}\end{aligned} \tag{17}$$

这里的矩阵 $\mathbf{Q}_D$ 一般非零元素很少，所以数据量 N 较大时需要耗费大量内存来存储中间结果，而且计算也比较吃力。同时，虽然 $\mathbf{Q}_D$ 的维度与 $\tilde{d}$ 无关，但其中每个元素要算内积，特征转换之后算内积的维度更大，所以其实还是与 $\tilde{d}$ 有关系。

在此问题下求解外层最优 α，然后代入 $\mathbf{w}$ 即可求得最优。

假设训练数据正负例都有（否则不用分类了），那么最优解中至少有一个 $\alpha_s > 0$。由 KKT 条件中的 complementary slackness 可知，α 和约束在最优解时必有一个为 0，所以这个 $\alpha_s > 0$ 必须满足

$$y_n(\mathbf{w}^T\mathbf{x}_s + b) = 1$$

由此即可解出 b

$$b^* = y_s - \sum_{\alpha_n^*>0} y_n\alpha_n^*\mathbf{x}_n^T\mathbf{x}_s$$

这里可知，只有满足 $\alpha_n > 0$ 才会为分类面做贡献。它们对应的数据点即成为支持向量。

将 $\mathbf{w}$ 和 b 代入分类面的方程，即可得到最终输出的 hypothesis 为

$$g(\mathbf{x}) = sign\left(\sum_{\alpha_n^*>0} y_n\alpha_n^*\mathbf{x}_n^T\mathbf{x} + b^*\right)$$

注意，支持向量只是最终决定分类面的 candidate，分类面 margin 边界上有些点可能对应的 $\alpha = 0$，因此并未对分类面做贡献。于是之前的交叉验证 error 可以进一步被只满足 $\alpha > 0$ 的支持向量限制。实际中支持向量的数量可能并不多，所以 dual 问题为非线性的特征转换提供了便利：既能利用特征变换做出复杂的分类边界，又能保证模型复杂度受到限制。

2 Week 2

2.1 Kernel Trick

Kernel 的用途是将前面 dual SVM 中矩阵 $\mathbf{Q}_D$ 的内积计算和非线性变换加以结合，不用显式地做特征变换，同时直接计算内积，使计算彻底与 $\tilde{d}$ 无关。一般形式为：

$$K_\Phi(\mathbf{x}, \mathbf{x}') \equiv \Phi(\mathbf{x})^T\Phi(\mathbf{x}')$$

2.2 多项式 Kernel

以 second-order polynomial kernel 为例推导一下：

$$\Phi_2(\mathbf{x})^T\Phi_2(\mathbf{x}') = 1 + \sum_{i=1}^{d} x_ix_i' + \sum_{i=1}^{d}\sum_{j=1}^{d} x_ix_jx_i'x_j'$$

最后一项

$$\sum_{i=1}^{d}\sum_{j=1}^{d} x_ix_jx_i'x_j' = \sum_{i=1}^{d} x_ix_i' \times \sum_{j=1}^{d} x_jx_j' = (\mathbf{x}^T\mathbf{x}')^2$$

所以

$$K(\mathbf{x}^T, \mathbf{x}') = 1 + (\mathbf{x}^T\mathbf{x}') + (\mathbf{x}^T\mathbf{x}')^2$$

可见这里直接在训练数据的空间中算内积，就能直接得出变换后空间的内积，不用做变换了。前面导出的 SVM 的公式中，直接用 kernel 换掉内积即得带 kernel 简化的 SVM。除 SVM 之外的其他学习算法需要计算内积的地方都可以从 kernel 获益。

Kernel 做的事情实际上是将特征变换到一个特定的空间中，然后利用数学性质来简化内积运算。

在上面 2 阶多项式 kernel 的基础上，可以进一步配方得到平方形式的 kernel:

$$\Phi(\mathbf{x}) = \left(\zeta, \sqrt{2\gamma\zeta}x_1, \sqrt{2\gamma\zeta}x_2, \ldots, \sqrt{2\gamma\zeta}x_d, \gamma x_1 x_1, \gamma x_2 x_2, \ldots, \gamma x_d x_d\right)$$

于是

$$K(\mathbf{x}^T, \mathbf{x}') = (\zeta + \gamma\mathbf{x}^T\mathbf{x}')^2$$

调整参数 ζ, γ 可以改变变换后的几何特性，从而实现不同的内积运算，影响 SVM 的距离计算，由此来达到不同复杂度的分类边界和 margin。

同样的还可以得出 degree-Q polynomial kernel:

$$K(\mathbf{x}^T, \mathbf{x}') = (\zeta + \gamma\mathbf{x}^T\mathbf{x}')^Q$$

2.3 Gaussian-RBF Kernel

另一个常用的 kernel 是 Gaussian-RBF kernel:

$$K(\mathbf{x}^T, \mathbf{x}') = \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2\right), \quad \gamma > 0$$

这个东西是哪两个变换后向量的内积呢？以一维为例:

$$\begin{aligned} K(x, x') &= \exp\left(-\|x - x'\|^2\right) \\ &= \exp\left(-x^2\right)\exp\left(2xx'\right)\exp\left(-(x')^2\right) \\ &= \exp\left(-x^2\right)\left(\sum_{k=0}^{\infty}\frac{2^k(x)^k(x')^k}{k!}\right)\exp\left(-(x')^2\right) \end{aligned} \tag{18}$$

最后一步用了原点的 Taylor 展开。把这个拆开就得到特征变换:

$$\Phi(x) = \exp(-x^2)\left(1, \sqrt{\frac{2^1}{1!}}x, \sqrt{\frac{2^2}{2!}}x, \ldots\right)$$

这玩意儿直接变到了无限维，增加了模型的表达能力，但仍然能算出变换后的内积。

Gaussian-RBF kernel 用的是 Gaussian 函数，这个玩意儿实际上就是正态分布的那种钟形曲线。最后 SVM 得到的 g 里面，用这个换掉内积，得到的实际上是一堆以支持向量为中心的钟形曲线的线性组合。参数 γ 控制钟的宽度，γ 越大越窄（变为一个个尖峰）。

2.4 Kernel 的选择

先尝试 linear kernel，即多项式 kernel 中 $\gamma = 1, \zeta = 0, Q = 1$，也就是恒等变换，直接在原空间中算内积。Linear kernel 的好处是因为没有复杂的变换，所以可以在得到 $\mathbf{w}$ 之后直接看出不同特征的权重分配结果，有助于数据分析。但坏处显然是模型表达能力不够，无法做出复杂的分类边界。

多项式 kernel 表达能力强了一些，但太高次多项式 kernel 计算是会有数值计算的误差问题：要么数值太大表示不了/存不下，要么数值太小几乎就是 0，所以多项式 kernel 一般用 $Q \leq 10$ 的次数，同时要慎重选择才 ζ, γ。

Gaussian-RBF kernel 表达能力强，但无法像 linear kernel 那样提供对分类结果的解释。解释说变换到无限维再算内积有点邪乎。

除了这些 kernel 也有别的选择，甚至可以自己构造新 kernel。但必须满足

$$K = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_N) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & K(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ K(\mathbf{x}_N, \mathbf{x}_1) & K(\mathbf{x}_N, \mathbf{x}_2) & \cdots & K(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

这个矩阵必须是半正定的。这个条件是 kernel 合法性的充要条件，称为 Mercer's condition。

2.5 Soft-margin SVM

特征变换 + kernel 仍然有过拟合的可能。有时候数据中存在一些 outliers，直接上复杂的 kernel 很容易过拟合。所以 soft-margin SVM 的目的是要让 SVM 对 outlier 的容忍度，允许一些数据进入 margin 甚至允许一些分类错误，从而提高对过拟合的抵抗力。

对每个点 $(\mathbf{x}_n, y_n)$，定义 margin violation $\xi_n \geq 0$，要求

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n$$

也就是说 ξ_n 描述了一个点突破 margin 的程度。为了衡量整个 SVM 对突破 margin 的容忍性，将所有 ξ_n 求和作为惩罚项加入目标函数。于是 SVM 的问题变为

$$\begin{aligned} \max_{b,\mathbf{w},\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N} \xi_n \\ \text{subject to} \quad & y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n \quad \text{for } n = 1, 2, \ldots, N \\ & \xi_n \geq 0 \quad \text{for } n = 1, 2, \ldots, N \end{aligned} \tag{19}$$

惩罚项的权重 C 表示了我们在允许突破 margin 和要求 margin 最大之间的权衡。C 越大越不允许越界，也就越接近 hard-margin SVM。C 越小越能容忍越界，但关心 margin 大。

此时的 Lagrange 函数为

$$\mathcal{L}(b, \mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\alpha_n(1 - \xi_n - y_n(\mathbf{w}^T\mathbf{x}_n + b)) - \sum_{n=1}^{N}\beta_n\xi_n$$

对 ξ_n 求 $\partial\mathcal{L}/\partial\xi_n = 0$ 即得 $C - \alpha_n - \beta_n = 0$，所以可以把 $\beta_n = C - \alpha_n$ 代回原式子，消掉 β_n，即

$$\begin{aligned}\mathcal{L}(b,\mathbf{w},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta}) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\alpha_n(1-\xi_n - y_n(\mathbf{w}^T\mathbf{x}_n+b)) - \sum_{n=1}^{N}(C-\alpha_n)\xi_n \\ &= \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{n=1}^{N}\alpha_n(1-y_n(\mathbf{w}^T\mathbf{x}_n+b))\end{aligned} \tag{20}$$

现在这个问题与 hard-margin SVM 一样了，只是每个 α_n 多了一个上界 C:

$$0 \le \alpha_n \le C$$

所以可以直接解出 $\boldsymbol{\alpha}$，然后得到 $\mathbf{w}$。但求 b 有所不同。根据 KKT 条件中的 complementary slackness，在 soft-margin SVM 中要满足

$$\begin{aligned}\alpha_n^*\left(y_n(\mathbf{w}^T\mathbf{x}_n+b)) - 1 + \xi_n^*\right) &= 0 \\ \beta_n^*\xi_n^* = (C-\alpha_n^*)\xi_n^* &= 0\end{aligned} \tag{21}$$

分三种情况讨论下

1. $\alpha_n^* = 0$，对应的点不是支持向量

2. $0 < \alpha_n^* < C$，根据上面第二个式子得到 $\xi_n^* = 0$，代入第一个得到 $y_n(\mathbf{w}^T\mathbf{x}_n+b)) = 1$，对应的点是支持向量，称为 free support vector，按 hard-margin SVM 同样的方法解出 b

3. $\alpha_n^* = C$，称为 bound support vector，此时 $\xi_n^* \ge 0$，对应的点有可能违反 margin 边界。此时 $y_n(\mathbf{w}^T\mathbf{x}_n+b)) = 1-\xi_n^* \le 1$，得到的是 b 的取值范围，随便选一个

2.6 模型选择

引入 kernel 之后涉及到调参的问题，那么如何选择模型？Validation!

例如对 soft-margin SVM，与参数相关的 $E_{CV}(C,\gamma)$ 很难直接优化，所以可以做一系列的 C,γ 参数组合，然后根据交叉验证的结果来选 E_{CV} 最小的参数组合。

另外，因为 SVM 的 leave-one-out 交叉验证 error 上界受支持向量数的限制，所以可以通过取不同的参数组合，根据支持向量数来排除一些复杂的模型。但由于只是上界，所以一般只是作为 E_{CV} 计算代价太高时的 safety check 的方法。

3 Week 3

这一周的主要内容就是建立 SVM 和其他线性模型的联系，用 SVM 来做其他事情。SVM 是 QP，可以转为 dual 再解，也可以用 kernel，所以我们希望把这些特性也扩展到其他线性模型中。

3.1 Soft-margin SVM 与 regularization 的联系

把 ξ_n 看做 $\xi_n = max(1 - y_n(\mathbf{w}^T\mathbf{z}_n + b), 0)$，即点 $\mathbf{z}_n$ 超越分类边界的程度。如果没有超越，根据 SVM 的条件有 $y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1$，此时 $\xi_n = 0$。这样改写之后，soft-margin SVM 问题变为

$$\min_{b,\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N} max\{1 - y_n(\mathbf{w}^T\mathbf{z}_n + b, 0\}$$

后面那一坨相当于一个 error function，而前一项则可看做 L2 regularization (weight decay)。

为啥不直接解这个 augmented error 的最小化问题？因为 max 形式不好，有不可微点，而且此问题非 QP，无法转为 dual，无法使用 kernel。

注意这里的常数 C 越大，则 error 占的比例越大，所以是越倾向于减少错误而放宽惩罚项，对应于普通 L2 regularization 问题 $\frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + E_{in}$ 中比较小的 λ。

3.2 SVM 的 error function

令 $s = \mathbf{w}^T\mathbf{z}_n + b$, 从前面的式子可以得出 soft-margin SVM 使用的 error function 为

$$\widehat{err_{SVM}}(s, y) = max(1 - ys, 0)$$

也叫 hinge error measure，它是 0-1 error $err_{0/1} = \mathbf{1}\{ys \leq 0\}$ 的上界，而且当 $ys \geq 1$ 时，和 logistic 回归用的 scaled cross entropy $err_{SCE} = \log_2(1 + e^{-ys})$ 比较接近。但在 $ys < 0$ 时，hinge error 与 SCE 一样相对于 0/1 error 都是比较松的上界。基本上可以把 soft-margin SVM 看做带 L2 regularization 的 logistic 回归。

3.3 用 SVM 做 soft binary classification

采用一种两阶段学习的方法，即，先通过 SVM 训练出 $\mathbf{w}_{SVM}$ 和 b_{SVM}，然后对这个分类平面求出的分数值进行 scaling 和 shifting，再传入 sigmoid 函数得到概率，即

$$g(\mathbf{x}) = \theta\left(A(\mathbf{w}_{SVM}^T\Phi(\mathbf{x}) + b_{SVM}) + B\right)$$

问题转化为

$$\min_{A,B} \frac{1}{N}\sum_{n=1}^{N} log_2\left(1 + \exp\left(-y_n\left(A(\mathbf{w}_{SVM}^T\Phi(\mathbf{x}) + b_{SVM}) + B\right)\right)\right)$$

解的步骤为

1. 执行 SVM 得到 $\mathbf{w}_{SVM}$ 和 b_{SVM}
2. 将数据变换到 $\mathbf{z}'_n = \mathbf{w}_{SVM}^T\Phi(\mathbf{x_n}) + b_{SVM}$
3. 执行 logistic regression 得到 A, B
4. 返回 g

第三步直接梯度下降或随机梯度下降即可。

3.4 直接在 $\mathcal{Z}$ 空间解 logistic 回归

既然 kernel SVM 可以直接在变换后的空间中解问题，而前面说到 SVM 和 logistic 回归又有联系，那么能否用 kernel 直接在变换后的空间中解 logistic 回归呢?

Representer Theorem：任意 L2-regularized linear model

$$\min_{\mathbf{w}} \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + \frac{1}{N}\sum_{n=1}^{N} err(y_n, \mathbf{w}^T\mathbf{z}_n)$$

的最优解 $\mathbf{w}_*$ 都可以被表示为数据的线性组合，即 $\mathbf{w}_* = \sum_{n=1}^{N} \beta_n \mathbf{z}_n$

证明：$\mathbf{w}_*$ 能被表示为数据的线性组合即相当于它在 $span\{\mathbf{z}_n\}$。假设它不能被表示为线性组合，则可以将其分别正交投影到 $span\{\mathbf{z}_n\}$ 和与它正交的空间，即

$$\mathbf{w}_* = \mathbf{w}_{\parallel} + \mathbf{w}_{\perp}$$

所以，error 项

$$err(y_n, \mathbf{w}_*^T\mathbf{z}_n) = err(y_n, (\mathbf{w}_{\parallel} + \mathbf{w}_{\perp})^T\mathbf{z}_n)$$

惩罚项

$$\mathbf{w}_*^T\mathbf{w}_* = \mathbf{w}_{\parallel}^T\mathbf{w}_{\parallel} + 2\mathbf{w}_{\parallel}^T\mathbf{w}_{\perp} + \mathbf{w}_{\perp}^T\mathbf{w}_{\perp} > \mathbf{w}_{\parallel}^T\mathbf{w}_{\parallel}$$

表明 $\mathbf{w}_{\parallel}^T\mathbf{w}_{\parallel}$ 更小，与 $\mathbf{w}_*$ 是最优解矛盾。

所以，根据 Representer 定理，可以直接将 $\mathbf{w}_* = \sum_{n=1}^{N} \beta_n \mathbf{z}_n$ 代入 L2-regularized logistic 回归问题

$$\min_{\mathbf{w}} \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + \frac{1}{N}\sum_{n=1}^{N} log_2(1 + \exp(-y_n\mathbf{w}^T\mathbf{z}_n))$$

得到 kernel logistic regression（KLR)：

$$\min_{\boldsymbol{\beta}} \frac{\lambda}{N}\sum_{n=1}^{N}\sum_{n=1}^{M}\beta_n\beta_m K(\mathbf{x}_n\mathbf{x}_m) + \frac{1}{N}\sum_{n=1}^{N} log_2\left(1 + \exp\left(-y_n\sum_{m=1}^{N}\beta_m K(\mathbf{x}_m, \mathbf{x}_n)\right)\right)$$

此时只有一个变量，直接 GD、SGD 求解即可。

上面这个式子也可以把前一项看做惩罚项，后一项中的 kernel 则相当于把数据 $\mathbf{x}_n$ 变换到了 $(K(\mathbf{x}_1, \mathbf{x}_n), \ldots, K(\mathbf{x}_N, \mathbf{x}_n))$，并且用 $\boldsymbol{\beta}$ 而非 $\mathbf{w}$ 来做权重。

与 SVM 不同的是，直接解这个东西，解出来 β 很多都不是 0，所以计算和存储的消耗还是要考虑进去的。

3.5 Kernel Ridge Regression

既然 logistic 回归可以用 kernel，那么一般线性回归呢?

采用跟上面类似的方法，根据 Representer 定理，将数据点的线性组合和平方误差函数代入

$$\min_{\mathbf{w}} \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + \frac{1}{N}\sum_{n=1}^{N}(y_n - \mathbf{w}^T\mathbf{z}_n)^2$$

得到

$$\min_{\boldsymbol{\beta}} \frac{\lambda}{N}\sum_{n=1}^{N}\sum_{n=1}^{M}\beta_n\beta_m K(\mathbf{x}_n\mathbf{x}_m) + \frac{1}{N}\sum_{n=1}^{N}\left(y_n - \sum_{m=1}^{N}\beta_m K(\mathbf{x}_m,\mathbf{x}_n)\right)$$

写成矩阵形式即

$$E_{aug}(\boldsymbol{\beta}) = \frac{\lambda}{N}\boldsymbol{\beta}^T\mathbf{K}\boldsymbol{\beta} + \frac{1}{N}(\boldsymbol{\beta}^T\mathbf{K}^T\mathbf{K}\boldsymbol{\beta} - 2\boldsymbol{\beta}^T\mathbf{K}^T\mathbf{y} + \mathbf{y}^T\mathbf{y})$$

求梯度

$$\nabla E_{aug}(\boldsymbol{\beta}) = \frac{2}{N}\mathbf{K}^T((\lambda\mathbf{I} + \mathbf{K})\boldsymbol{\beta} - \mathbf{y})$$

直接解得

$$\boldsymbol{\beta} = (\lambda\mathbf{I} + \mathbf{K})^{-1}\mathbf{y}$$

这个计算量与数据量相关，而普通的回归问题与数据维度相关。同时，解出来 β 也有很多不为 0。所以这里需要权衡，普通回归 $N \gg d$ 时比较好用，N 一大，kernel 回归效率就低了。但同时普通回归的表达能力不如 kernel 回归。

注意到，这里实际上是包含了转换前数据中的常数项 bias。或者可以从普通的 ridge regression 推导出 kernel ridge regression。普通回归即

$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}, \quad \boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

根据

$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{X}^T = \mathbf{X}^T\mathbf{X}\mathbf{X}^T + \lambda\mathbf{X}^T = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})$$

左边乘 $(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}$，右边乘 $(\mathbf{X}\mathbf{X}^\mathbf{T} + \lambda\mathbf{I})^{-1}\mathbf{y}$，得到

$$\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{y} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} = \boldsymbol{\beta}$$

令 $\boldsymbol{\alpha} = (\boldsymbol{X}\boldsymbol{X}^T + \lambda\mathbf{I})^{-1}\mathbf{y}$，则

$$\boldsymbol{\beta} = \mathbf{X}^T\boldsymbol{\alpha}$$

对于一个要预测的数据点 $\mathbf{x}$，有

$$\hat{y} = \boldsymbol{\beta}^T\mathbf{x} = \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{X}\mathbf{x}$$

注意到，这里的 $\mathbf{X}\mathbf{X}^T$ 即训练数据中所有的点之间的内积，而 $\mathbf{X}\mathbf{x}$ 则是每个训练数据与待预测数据的内积。定义前者为矩阵 $\mathbf{K}$，后者为向量 $\mathbf{k}$，则上式可改写为

$$\hat{y} = \boldsymbol{\beta}^T \mathbf{x} = \mathbf{y}^T (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{k}$$

这里都是基于 ridge regression 推出来的，所以都是默认将训练数据增加一个分量 1，对应到权重的常数项 bias。如果做了特征变换，则相当于保持补充的分量 1 不变，其他维度输入给转换函数 Φ，然后这里的内积就可以换成 kernel。

3.6 用 kernel ridge regression 做分类

此方法即 least-square SVM（LSSVM），和普通 SVM 得到的分类边界差不多，但支持向量更多，得到的非零结果也更多。

3.7 Support Vector Regression

我觉得这个方法才是真正结合了 SVM 思想的回归。思路时在回归出来的函数上下加上 margin，如果数据点在 margin 内则认为没有回归误差，否则计算误差。与 soft-margin SVM 一样，相当于提高了模型的容忍度。

这里用到的 error 函数为

$$err(y, s) = max(0, |s - y| - \epsilon)$$

其中 ϵ 为 margin 宽度，如果模型计算出的分值 s 与正确值 y 的差距小于 margin 宽度则 error 为 0，否则取这个差值作为 error。

注意这个 tube error 实际上和平方误差在 $|s - y|$ 较小时是很接近的，而越往两边走，平方误差增长越快。所以 tube error 更少受到 outliers 的影响，因为平方误差在这种情况下通常值比较大，模型会更多地去做修正，从而容易对 outlier 产生过拟合。

仿照 soft-margin SVM，可以构造出 tube regression 的问题：

$$\min_{b,\mathbf{w}} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{n=1}^{N} \max(0, |\mathbf{w}^T \mathbf{z}_n + b - y_n| - \epsilon)$$

然后替换掉上面那一坨 max，把这个东西转换为 QP，史称 support vector regression（SVR）。

$$\begin{aligned} \min_{b,\mathbf{w},\boldsymbol{\xi}} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{n=1}^{N} \xi_n \\ \text{subject to} \quad & |\mathbf{w}^T \mathbf{z}_n + b - y_n| \geq \epsilon + \xi_n \\ & \xi_n \geq 0 \\ & n = 1, 2, \ldots, N \end{aligned} \tag{22}$$

现在的问题是如何确定 ξ_n？ξ_n 实际上是在衡量预测分数与实际值之差是否有超过 margin，或者超过了多少。所以上面的约束条件规定的是每个点 $\mathbf{z}_n$ 在 ξ_n 的容忍度之下要满足的范围。根据绝对值里面那一坨的符号，可以把容忍度 ξ_n 分为在回归线上面和下面的容忍度：$\xi_n^{\vee}, \xi_n^{\wedge}$，所以可以将上面的问题改写为：

$$\begin{aligned}\min_{b,\mathbf{w},\boldsymbol{\xi}^\vee,\boldsymbol{\xi}^\wedge} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}(\xi_n^\vee + \xi_n^\wedge) \\ \text{subject to} \quad & -\epsilon - \xi_n^\vee \le y_n - \mathbf{w}^T\mathbf{z}_n - b \le \epsilon + \xi_n^\wedge \\ & \xi_n^\vee \ge 0, \xi_n^\wedge \ge 0 \\ & n = 1, 2, \ldots, N\end{aligned} \tag{23}$$

此问题总共有 $\tilde{d}+1+2N$ 个变量，$2N+2N$ 个约束。

将此问题转化为 dual，分别设置 $\xi_n^\vee$ 和 $\xi_n^\wedge$ 的 lagrange multiplier 为 $\alpha_n^\vee$ 和 $\alpha_n^\wedge$，得到 dual

$$\begin{aligned}\min_{\boldsymbol{\xi}^\vee,\boldsymbol{\xi}^\wedge} \quad & \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(\alpha_n^\wedge - \alpha_n^\vee)(\alpha_m^\wedge - \alpha_m^\vee)K(\mathbf{x}_n, \mathbf{x}_m) \\ & + \sum_{n=1}^{N}((\epsilon - y_n)\alpha_n^\wedge + (\epsilon + y_n)\alpha_n^\vee) \\ \text{subject to} \quad & \sum_{n=1}^{N}(\alpha_n^\wedge - \alpha_n^\vee) = 0 \\ & 0 \le \alpha_n^\wedge \le C, 0 \le \alpha_n^\vee \le C \\ & n = 1, 2, \ldots, N\end{aligned} \tag{24}$$

SVR 的解是否稀疏呢？根据前面可以推出

$$\mathbf{w} = \sum_{n=1}^{N}(\alpha_n^\wedge - \alpha_n^\vee)\mathbf{z}_n$$

根据 KKT 之 complementary slackness

$$\begin{aligned}\alpha_n^\wedge(\epsilon + \xi_n^\wedge - y_n + \mathbf{w}^T\mathbf{z}_n + b) = 0 \\ \alpha_n^\vee(\epsilon + \xi_n^\vee + y_n - \mathbf{w}^T\mathbf{z}_n - b) = 0\end{aligned} \tag{25}$$

当 $|\mathbf{w}^T\mathbf{z}_n + b - y_n| < \epsilon$ 时，误差可以被忽略，所以 $\xi_n^\wedge = \xi_n^\vee = 0$，而且 $(\epsilon+\xi_n^\wedge - y_n + \mathbf{w}^T\mathbf{z}_n + b) \ne 0$，$(\epsilon+\xi_n^\vee + y_n - \mathbf{w}^T\mathbf{z}_n - b) \ne 0$，所以 $\alpha_n^\wedge = \alpha_n^\vee = 0$，对应的 $\mathbf{z}_n$ 系数为 0，非支持向量，不贡献给 $\mathbf{w}$。

所以 SVR 的解是稀疏的。

3.8 模型比较

算法	问题形式	解法	Error Measure
PLA/Pocket	min	iteration	$err_{0/1}$
SVR	min+reg	QP on primal	err_{TUBE}
Soft-margin SVM	min+reg.	QP on primal	$\widehat{err_{SVM}}$
Ridge Regression	min+reg.	analytical solution	err_{SQR}
Logistic Regression	min+reg.	GD/SGD	err_{SCE}
SVM	min	QP on dual	
Dual SVR	min	QP on dual	
probabilistic SVM	min	SVM then logistic	
Kernel Ridge Reg.	min+reg.	kernel + analytical	
Kernel Log. Reg.	min+reg.	kernel + GD/SGD	

4 Week 4

4.1 Aggregation

Aggregation 的思路是，手里有一堆（通常比较弱的）hypothesis $g_1, g_2, \ldots, g_T$ 时，通过投票的方法得到一个综合所有 g 的更强的 G。

常用的投票方法

1. 根据 validation error 选择最优 g：$G(\mathbf{x}) = g_k(\mathbf{x}), k = \underset{t\in 1,2,\ldots,T}{\operatorname{argmin}} E_{val}(g_t^-)$

2. 均匀混合：$G(\mathbf{x}) = sign(\sum_{t=1}^{T} g_t(\mathbf{x}))$

3. 非均匀混合：$G(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})), \alpha_t \geq 0$。这个方法包含前两种，第一种相当于取 $\alpha_t = \mathbf{1}\{E_{val}(g_t^-)\text{最小}\}$，第二种相当于取 $\alpha_t = 1$

4. 按条件混合：$G(\mathbf{x}) = sign(\sum_{t=1}^{T} q_t(\mathbf{x}) g_t(\mathbf{x})), q_t(\mathbf{x}) \geq 0$。显然取 $q_t(\mathbf{x}) = \alpha_t$ 就可以包含第三种了

两个例子：通过 decision stump 获得比较复杂的分类边界（类似于特征变换）；通过 PLA 得到的（随机的）分类线得到一个平均的分类线（类似于 SVM 的最大 margin regularization）。

4.2 Uniform Blending

如前所述，每人一票：

$$G(\mathbf{x}) = sign(\sum_{t=1}^{T} g_t(\mathbf{x}))$$

多类问题中，投票改成哪个类别被投得多就选哪个：

$$G(\mathbf{x}) = \underset{1\leq k\leq K}{\operatorname{argmax}} \sum_{t=1}^{T} \mathbf{1}\{g_t(\mathbf{x}) = k\}$$

回归问题中，改为计算每个 g 输出值的平均：

$$G(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{x})$$

根据回归问题可以推一个上界。假设上面取平均的操作为 avg，目标函数为 f，则对于某个 $\mathbf{x}_n$（略去不写了），有

$$\begin{aligned} avg((g_t - f^2)) &= avg(g_t^2 - 2g_t f + f^2) \\ &= avg(g_t^2) - 2Gf + f^2 \\ &= avg(g_t^2) - G^2 + (G-f)^2 \\ &= avg(g_t^2) - 2G^2 + G^2 + (G-f)^2 \\ &= avg(g_t^2 - 2g_t G + G^2) + (G-f)^2 \\ &= avg((g_t - G)^2) + (G-f)^2 \end{aligned} \tag{26}$$

将上式两边对所有 $\mathbf{x}_n$ 求和/积分，就得到上面左边即随便选一个 g_t 与 f 的误差 $E_{out}(g_t)$，右边第二项即 G 的误差 $E_{out}(G)$，右边第一项即 $avg(\mathbb{E}((g_t - G)^2)$，所以

$$avg(E_{out}(g_t)) \geq E_{out}(G)$$

假设我们通过这样的方式来学习到最终的 hypothesis:

1. 从某分布 i.i.d 得到一组大小为 N 的数据

2. 通过算法 $\mathcal{A}$ 得到 g_t

3. 一共进行 T 轮这样的步骤，最后求所有 g_t 的平均在 $T \to \infty$ 时的极限 $\bar{g}$

最后按上面的式子可得

$$E_{out}(g_t) = avg(\mathbb{E}((g_t - \bar{g})^2)) + E_{out}(\bar{g})$$

上式说明: 算法 $\mathcal{A}$ 的表现等于右边第一项: variance, 加上右边第二项: bias。其中 bias 是 consensus $\bar{g}$ 的表现。

Uniform blending 实际上就是在减小 variance 来获得更稳定的表现。

4.3 Linear Blending

按权重投票:

$$G(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})), \alpha_t \geq 0$$

问题变为了找出使 E_in（一般当然要 validation 了，用 E_{val}）最小的 $\boldsymbol{\alpha}$。

对于线性回归，即

$$\min_{\alpha_t \geq 0} \frac{1}{N}\sum_{n=1}^{N}\left(y_n - \sum_{t=1}^{T}\alpha_t g_t(\mathbf{x})\right)^2$$

这东西就是一个线性模型，相当于是用每个 g_t 给每个 $\mathbf{x}_n$ 做特征转换为 $g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_T(\mathbf{x})$，再用 α_t 来加权，唯一的区别是这里权值 α_t 都有约束。但这里的约束通常可以忽略掉，因为对于分类来说，如果 $\alpha_t < 0$，你给它加个绝对值，把符号搞到后面，就变成 $-g_t(\mathbf{x})$，相当于预测结果取了个反，得到的分类器还是一样的。(但是对于其他问题呢?)

4.4 Any Blending

Any Blending 即把 linear blending 中的线性模型换成任意模型。注意如果做了交叉验证，训练时用的相当于 $g_1^-(\mathbf{x}), g_2^-(\mathbf{x}), \ldots, g_T^-(\mathbf{x})$，最后返回最终结果时，要将数据变为 $g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_T(\mathbf{x})$。

Any blending 可以实现 conditional blending，即先判断 $\mathbf{x}$ 是否符合某种条件，然后再用 g 预测。

4.5 Bagging: Bootstrap Aggregation

之前的各种 blending 手段前提都是手上已经有了一堆的 g，那么能不能一边得到参与投票的新 g，一边做 aggregate?

要得到新的 g，就要通过各种方式在现有 g 的基础上产生差异，如

1. 来自不同模型的 g
2. 同一模型的不同参数，如 GD
3. 算法本身的随机性，如 PLA
4. 数据的随机性，如交叉验证

或者，根据前面推导的那个上界，在同一个算法的基础上通过不同的数据来得到新的 g，但又必须保证和原训练数据来自同一分布。于是可以用 bootstrap 的方法，从原训练数据中采样获得新的数据——有放回地从 $\mathcal{D}$ 抽取 N' 个数据。

Bagging 特别适合于对数据随机性敏感的 g。

4.6 Adaptive Boosting (AdaBoost)

AdaBoost 的目的是让若干较弱的分类器 g 组合成一个更强的分类器 G。

从 bootstrap 出发，每一轮做过抽样之后，每组数据 $\mathbf{x}$ 相当于有了一个权值，相当于 $\mathbf{x}$ 在抽样出来的数据集中占有多大比例。而我们要最小化的误差函数则相应地算入了 $\mathbf{x}$ 的权值。

回忆教小孩认苹果的例子，在 bagging 过程中，通过人为地设置权重，可以起到类似于老师那种强调犯错的例子、弱化正确例子的作用，这样做的理由是增加 g 的差异性，让每个 g 专注于某个方面的“专长”。

具体到算法中，如果第 t 轮的 g_t 在第 $t+1$ 轮的权重分配 $u_n^{(t+1)}$ 下表现很糟糕，那么显然第 $t+1$ 轮基本不会得到 g_t 或者和它类似的 hypothesis，从而让 g_{t+1} 产生了和 g_t 的差异性。

要让 g_t 表现糟糕，相当于让它的表现与随机乱分差不多，即在第 $t+1$ 轮有

$$\frac{\sum_{n=1}^{N} u_n^{(t+1)} \mathbf{1}\{y_n \neq g_t(\mathbf{x}_n)\}}{\sum_{n=1}^{N} u_n^{(t+1)}} = \frac{1}{2}$$

那么如何调整呢？简单推算即可得到，从第 t 轮到 $t+1$ 轮，只需要

- 分错的数据权值调整为正比于 $1-\epsilon_t$
- 分对的数据权值调整为正比于 ϵ_t

其中 ϵ_t 为 g_t 在第 t 轮中的错误率。

这两个 scaling 操作可以统一为一个因子 $e_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$，每轮更新样本权重时，分错的样本乘以 e_t ，分对的除以 e_t。这个因子的物理意义是，如果 $\epsilon_t \leq \frac{1}{2}$，则 $e_t \geq 1$。也就是说，如果 g_t 的错误率低于随机乱分，即表现优于随机乱分，则将犯错的样本权重增大，分对的样本权重减小，从而使后面的 g 更多地聚焦在分错的数据上。

最后 aggregation 的过程可以像 bagging 一样用各种线性非线性来搞，也可以用 AdaBoost 的方法来搞，即根据各个弱分类器的表现来赋予它们权重，取 $\alpha_t = \ln e_t = \ln \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$，在每轮迭代中直接计算出来，最后返回 $G(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}))$

初始时，设置所有 $u^{(1)} = \frac{1}{N}$。

AdaBoost 能很快地将 E_{in} 搞到很小。

5 Week 5

5.1 Decision Tree & CART

Decision tree、AdaBoost、bagging 都是在一开始没有各种 g 的时候进行学习的方法。Decision tree 对应于 blending 中的 conditional aggregation，即按条件地做预测。

决策树中每个叶节点相当于一个 g，从根节点到叶节点的每条路径都对应这个 g 要符合的条件，即，从根是否存在一条到这个叶节点的路径

$$G(\mathbf{x}) = \sum_{c=1}^{C} \mathbf{1}\{\mathbf{x} \text{ is on path } t\} \cdot g_t(\mathbf{x})$$

或者可以用递归的角度来看，在每个节点先通过一个分支函数 $b(\mathbf{x})$ 来决定应该走哪个分支，然后再执行对应分支下的子树的决策函数，即

$$G(\mathbf{x}) = \sum_{c=1}^{C} \mathbf{1}\{b(\mathbf{x}) = c\} \cdot G_c(\mathbf{x})$$

递归的思路比较适合程序实现。

这里主要介绍了 CART，一种比较由代表性的决策树算法：每个 g 都返回常数，构建二叉树。

1. 如果不用分支了，返回使 E_{in} 最小的常数

2. 基于 decision stump 学习节点分支条件 $b(\mathbf{x}) = \underset{h}{\operatorname{argmin}} \sum_{c=1}^{2} |\mathcal{D}_c| \times impurity(\mathcal{D}_c)$，其中 $\mathcal{D}_c$ 表示用 decision stump h 分割出来的一部分数据

3. 将数据分为两部分 $\mathcal{D}_c = \{(\mathbf{x}_n, y_n) : b(\mathbf{x}_n) = c\}$

4. 递归构建子树 $G_c \leftarrow \texttt{DecisionTree}(\mathcal{D}_c)$

5. 递归回来返回 $G(\mathbf{x}) = \sum_{c=1}^{2} \mathbf{1}\{b(\mathbf{x} = c)\} G_c(\mathbf{x})$

多类问题时直接修改一下就好。

上面每一个节点都要最小化 impurity，说白了就是要让每一个 decision stump 切出来的两半边，错分数量最小。

Impurity 的计算，回归常用

$$impurity(\mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} (y_n - y^*)^2$$

而分类常用 Gini index

$$1 - \sum_{k=1}^{K} \left(\frac{\sum_{n=1}^{N} \mathbf{1}\{y_n = k\}}{N} \right)^2$$

也就是计算 1 减去每一类分对的数据比例的平方和。

5.2 Pruning

根据叶节点数量 $\Omega(G)$ 调整惩罚力度

$$\underset{G}{\operatorname{argmin}} E_{in}(G) + \lambda \Omega(G)$$

通常训练时先得到一棵不剪枝的树 $G^{(0)}$，然后通过合并叶节点的方式依次得到少 1 片叶子 $G^{(1)}$、2 片叶子 $G^{(2)}$……的树，再通过上面的公式计算，得到一个能最小化 regularized error 的 G。λ 可通过交叉验证来选择。

5.3 Decision Tree 总结

训练中用到的 decision stump 可以换成 decision subset 来处理非数值类型的特征。

训练节点分支时，缺失的特征可以通过 surrogate branch 来解决，即在训练中找出与某个特征具有相似判断标准的特征，然后遇到缺失特征可以由备用的这些特征来替代。

决策树模型简单，很符合人类做决策的过程，而且有比较直观的可解释性（对比一下 kerne SVM）。

从可视化的角度来看，2D 决策树做出的分类边界中可能存在只划分部分点的分界线（子树），而 AdaBoost 中的所有分类线都是横穿整个平面。

5.4 Random Forest

随机森林结合 bagging 和决策树，即在 bagging 过程中，随机有放回重新抽样 N' 组数据用于训练决策树，最后用 uniform blending 组合所有决策树。

注意到 bagging 的每一轮迭代中，重新抽样和决策树的训练都是独立的，所以随机森林可以方便地将每一棵决策树的训练并行化，而且 bagging 也有利于减少单棵决策树过拟合的风险。

除了 bagging 本身的训练数据随机化，还可以引入特征随机化：先抽出 N' 组数据，然后对其中的每个 $\mathbf{x}$ 做一个特征变换 $\Phi(\mathbf{x}) = \mathbf{Px}$，变换矩阵 $\mathbf{P}$ 相当于选择了原 $\mathbf{x}$ 中的部分特征进行加权组合，将数据转换到一个（通常是）低维空间中。这两次随机化（bagging + random-combination）使每棵树的训练产生了更多了差异性。

5.5 Out-of-bag Data

Bagging 抽样没有抽中的数据称为 out-of-bag data，某组数据在 N 次抽样中都没有被抽到的概率为

$$\lim_{N\to\infty}\left(1-\frac{1}{N}\right)^N = \frac{1}{\left(1+\frac{1}{N-1}\right)^N} = \frac{1}{e}$$

所以平均每棵树的训练中都有约 $\frac{1}{e}\cdot N$ 组数据没有被抽中。这些未抽中数据可以用来对某个 G^- 做 validation。对每一组 $(\mathbf{x}_n, y_n)$，找出没有用这组数据训练出的所有决策树 g_t，得到 $G_n^- = average(g_t)$，然后用 $(\mathbf{x}_n, y_n)$ 来 validate G_n^-。最后将所有的 error 平均，即

$$E_{OOB}(G) = \frac{1}{N}\sum_{n=1}^{N} error(y_n, G_n^-(\mathbf{x}_n))$$

这个特性允许我们不用像交叉验证那样做重复的训练，即可得到最终 G 的评估结果。

5.6 特征选择

特征选择可以降低数据维度，简化训练过程，也可以去除一些有噪数据，减小过拟合风险，并增加模型的可解释性。决策树自带特征选择哦。

两种特征选择的方式：

线性模型　先用全部特征训练一个线性模型，再根据每个特征的权重 $|w_i|$ 选择权重较大的特征用于训练其他模型

Permutation test　将训练数据中的某个特征 x_i 随机重排，再用重排后的数据训练出新模型，比较前后两次训练结果的表现差异。差异越大说明此特征越重要。在随机森林中，可以通过 out-of-bag data 来避免重复训练。方法是在计算 out-of-bag error E_{OOB} 时，将 out-of-bag data $\mathbf{x}_n$ 中的特征 x_i 随机换为某个其他 out-of-bag data $\mathbf{x}_m$ 的 x_i。

5.7 Random Forest 总结

随机森林可以通过多棵决策树做出平滑且 margin 较大的边界。基本上树越多越好。训练的时候最好检查一下树的数量是不是够多，是否让随机性达到了较为稳定的状态。

6 Week 6

6.1 AdaBoosted Decision Tree

步骤：

1. 第 $t = 1, 2, \ldots, T$ 轮迭代
2. 用 $\mathbf{u}^{(t)}$ 对训练数据加权
3. 根据权重训练决策树 g_t 为 $\texttt{DecisionTree}(\mathcal{D}, \mathbf{u}^{(t)})$
4. 计算 g_t 的投票权重 α_t
5. 返回 α 加权投票的 G

由于 AdaBoost 中训练每棵决策树时要最小化 $E_{in}^{\mathbf{u}}(h) = \dfrac{1}{N}\sum_{n=1}^{N} u_n \cdot error(y_n, h(\mathbf{x}_n))$，我们希望将决策树当做黑盒，而不要修改其中的 error function。所以改用调整数据的方式，将数据权重 $\mathbf{u}$ 融合进去。说白了就是根据 $\mathbf{u}$ 对数据抽样，抽出大小为 N' 的 $\tilde{\mathcal{D}}_t$。

每棵决策树的投票权重 $\alpha_t = \ln\sqrt{\dfrac{1-\epsilon_t}{\epsilon_t}}$，这个 weighted error rate ϵ_t 如果在决策树完全长成的情况下会是 0，这棵树对应的 α_t 算出来会是 $+\infty$，所以需要剪枝，即用较弱的决策树。

如果用前面说到的限制节点数的方法来剪枝，剪到最极端的情形，只剩 1 个点，就等于在决策树中只需要学习分支函数，也就是一个 decision stump。所以在二元分类情况下，AdaBoosted Decision Stump 是 AdaBoosted Decision Tree 的特例。

6.2 AdaBoost as Functional Gradient Descent

下面主要推导如何将 AdaBoost 作为 functional gradient descent。

首先将 AdaBoost 中的样本权重更新过程统一为

$$u_n^{(t+1)} = u_n^{(t)} \exp(-y_n \alpha g_t(\mathbf{x}_n))$$

做完 T 轮迭代后

$$u_n^{(T+1)} = u_n^{(1)} \cdot \prod_{t=1}^{T} \exp(-y_n \alpha_t g_t(\mathbf{x}_n)) = \frac{1}{N} \exp\left(-y_n \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)\right)$$

而最终返回的 G 为

$$G(\mathbf{x}) = sign\left(\sum_{t=1}^{T}\alpha_t g_t(\mathbf{x})\right)$$

中间的求和即对 $\mathbf{x}$ 算出的分数。

如果把 $g_t(\mathbf{x}_n)$ 看做特征转换，α_t 做权重，联系到 SVM，$y_n \cdot score$ 相当于计算 margin，我们肯定想让 margin 尽可能大，那就要让 $\exp(-y_n \cdot score)$ 尽可能小，也就是让上面迭代完之后的 $u_n^{(T+1)}$ 尽可能小。而事实上 AdaBoost 会逐步减小 $\sum_{n=1}^{N} u_n^{(t)}$。所以将上面 $u_n^{(T+1)}$ 的计算公式代入，就得到

$$\sum_{n=1}^{N} u_n^{(T+1)} = \frac{1}{N}\sum_{n=1}^{N}\exp\left(-y_n\sum_{t=1}^{T}\alpha_t g_t(\mathbf{x}_n)\right)$$

令 $s = \sum_{t=1}^{T}\alpha_t g_t(\mathbf{x}_n)$，则上面这个式子可看做一个 error function

$$\widehat{err}_{ADA}(s, y) = \exp(-ys)$$

称为 exponential error measure，这个东西也是 0/1 错误的上界。

那么 AdaBoost 如何通过迭代来逐步最小化这个东西？联系梯度下降的迭代，AdaBoost 相当于是找到一个 functional 的梯度方向 $h(\mathbf{x}_n)$

$$\begin{aligned}
\widehat{E}_{ADA} &= \frac{1}{N}\sum_{n=1}^{N}\exp\left(-y_n\left(\sum_{k=1}^{t-1}\alpha_k g_k(\mathbf{x}_n) + \eta h(\mathbf{x}_n)\right)\right) \\
&= \sum_{n=1}^{N} u_n^{(t)}\exp(-y_n\eta h(\mathbf{x}_n)) \\
&\approx \sum_{n=1}^{N} u_n^{(t)}(1 - y_n\eta h(\mathbf{x}_n)) \\
&= \sum_{n=1}^{N} u_n^{(t)} - \eta\sum_{n=1}^{N} u_n^{(t)} y_n h(\mathbf{x}_n)
\end{aligned} \tag{27}$$

来最小化 $\sum_{n=1}^{N} u_n^{(t)}(-y_n h(\mathbf{x}_n))$。

然后分情况讨论

$$\begin{aligned}
\sum_{n=1}^{N} u_n^{(t)}(-y_n h(\mathbf{x}_n)) &= \sum_{n=1}^{N} u_n^{(t)}\begin{cases}-1 & \text{if } y_n = h(\mathbf{x}_n)\\ +1 & \text{if } y_n \neq h(\mathbf{x}_n)\end{cases} \\
&= -\sum_{n=1}^{N} u_n^{(t)} + \sum_{n=1}^{N} u_n^{(t)}\begin{cases}0 & \text{if } y_n = h(\mathbf{x}_n)\\ 2 & \text{if } y_n \neq h(\mathbf{x}_n)\end{cases} \\
&= -\sum_{n=1}^{N} u_n^{(t)} + 2N E_{in}^{\mathbf{u}^{(t)}}(h)
\end{aligned} \tag{28}$$

最后一个等号是因为那个 0、2 的 case 实际算的就是 weighted error。显然，能够最小化 weighted error 的就是 AdaBoost 的基础算法。所以 AdaBoost 就是在用基础算法找到这个最佳的梯度方向 $g_t = h$。

找到 g_t 后，$\widehat{E}_{ADA}$ 中要优化的只有后面一项 $\sum_{n=1}^{N} u_n^{(t)}(-y_n g_t(\mathbf{x}_n))$。然后对于下降的步长 η，可以用最激进的方式直接找到下降最多的值，称为 steepest descent。根据 $\exp(\dots)$ 一项的符号可以分情况讨论：

$$\begin{cases} u_n^{(t)} \exp(-\eta) & \text{if } y_n = g_t(\mathbf{x}_n) \\ u_n^{(t)} \exp(+\eta) & \text{if } y_n \neq g_t(\mathbf{x}_n) \end{cases}$$

要求最好的 η 直接求导即可，计算得到 $\eta_t = \ln\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \alpha_t$。所以 AdaBoost 做的实际上就是一个 functional 的 steepest descent。

6.3 Gradient Boosting & Gradient Boosted Regression

在上面的过程中，把 error function 改为任意函数，即从

$$\min_{\eta}\min_{h}\frac{1}{N}\sum_{n=1}^{N}\exp\left(-y_n\left(\sum_{k=1}^{t-1}\alpha_k g_k(\mathbf{x}_n)+\eta h(\mathbf{x}_n)\right)\right)$$

变为

$$\min_{\eta}\min_{h}\frac{1}{N}\sum_{n=1}^{N} error\left(\sum_{k=1}^{t-1}\alpha_k g_k(\mathbf{x}_n)+\eta h(\mathbf{x}_n), y_n\right)$$

就得到 gradient boosting。
特别地，如果用平方错误来做回归，则上面的最小化相当于

$$\min_{\eta}\min_{h}\frac{1}{N}\sum_{n=1}^{N} error\left(s_n+\eta h(\mathbf{x}_n), y_n\right)$$

在 s_n 处泰勒展开，里面根据 h 最小化的部分即

$$\min_{h}\frac{1}{N}\sum_{n=1}^{N} error(s_n, y_n)+\frac{1}{N}\sum_{n=1}^{N}\eta h(\mathbf{x}_n)\frac{\partial error(s, y_n)}{\partial s}|_{s=s_n}$$

前面一部分都不影响优化操作，最后求和部分即

$$\sum_{n=1}^{N} h(\mathbf{x}_n)\cdot 2(s_n - y_n)$$

整个最小，显然要对 $h(\mathbf{x}_n)$ 加以限制，否则直接取 $h(\mathbf{x}_n) = -\infty\cdot(s_n - y_n)$ 就行了。类似于 regularization，引入 h 的约束，上式变为最小化

$$\sum_{n=1}^{N}\left(2\cdot h(\mathbf{x}_n)(s_n - y_n)+(h(\mathbf{x}_n))^2\right)$$

配方得

$$\sum_{n=1}^{N}\left(constant + (h(\mathbf{x}_n) - (y_n - s_n))^2\right)$$

其他省略的部分都是与优化操作无关的常量，而后面需要找的最好的 $h(\mathbf{x}_n)$，就相当于对 residual 数据 $\{\mathbf{x}_n, y_n - s_n\}$ 做回归，相当于每轮迭代逐步靠近标准答案 y_n。所以 gradient boosting 中对 residual 做回归即得 g_t。

找到 g_t 后，要找最优的 η:

$$\min_{\eta} \frac{1}{N}\sum_{n=1}^{N} error\left(s_n + \eta g_t(\mathbf{x}_n), y_n\right)$$

在平方错误下，即

$$\min_{\eta} \frac{1}{N}\sum_{n=1}^{N} \left(s_n + \eta g_t(\mathbf{x}_n) - y_n\right)^2 = \frac{1}{N}\sum_{n=1}^{N}((y_n - s_n) - \eta g_t(\mathbf{x}_n))^2$$

找最优 η 相当于对用 g_t 变换过的 $\mathbf{x}$ 和 residual 做单变量的线性回归，最优解为

$$\frac{\sum_{n=1}^{N} g_t(\mathbf{x}_n)(y_n - s_n)}{\sum_{n=1}^{N}(g_t(\mathbf{x}_n))^2}$$

6.4 Gradient Boosted Decision Tree

将上面 gradient boosting 结合决策树，就得到了 GBDT:

1. 初始化所有 score $s_1 = s_2 = \cdots = s_N = 0$
2. 对每轮迭代 $t = 1, 2, \ldots, T$
3. 用决策树对 $\{\mathbf{x}_n, y_n - s_n\}$ 做回归，得到 g_t
4. 对 $\{g_t(\mathbf{x}_n), y_n - s_n\}$ 做单变量回归，得到 α_t
5. 更新 $s_n \leftarrow s_n + \alpha_t g_t(\mathbf{x}_n)$
6. 返回 $G(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})$

6.5 Aggregation 的总结

Blending		Aggregation + Learning	
uniform	投票	Bagging	bootstrap + 投票
		Random Forest	随机 bagging + DT
non-uniform	线性	AdaBoost	调样本权重 + steepest descent
		GradientBoost	对 residual 回归 + steepest descent
		GBDT	GradientBoost + 弱 DT
conditional	非线性	Decision Tree	划分数据 + 分支函数实现条件

6.6 Neural Network

基本思路是组合多个线性模型。以二元分类为例，假设有两组 g，分别对应 $\mathbf{w}_1$ 和 $\mathbf{w}_2$，如果给这俩的输出加上 α_1 和 α_2 的权值，则通过设置这些权值，可以实现布尔运算，从而形成非线性的分类边界。

常见的网络结构是，从输入 $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ 开始，把所有特征依次和 $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_j$ 相乘得到 j 个输出，然后输入经过 sigmoid 函数（常用 tanh）得到下一层的输入，依次经过所有 hidden layer 后，用最后一个 hidden layer 的 $\mathbf{w}$ 组合得到输出。

符号 $w_{ij}^{(l)}$ 表示第 l 层、第 j 个节点对应的 $\mathbf{w}$ 的第 i 个分量。第 l 层的节点总数为 $d^{(l)}$。第 l 层、第 j 个节点的输出为 $x_j^{(l)}$。

神经网络每一层都把前一层输出的 $\mathbf{x}$ 分别用一堆 $\mathbf{w}$ 相乘，然后搞一下 sigmoid 函数，如果这个内积越大，说明 $\mathbf{x}$ 中这些特征与 $\mathbf{w}$ 这种分配方式越接近，相当于用各层的 $\mathbf{w}$ 对原始的输入特征进行变换，各层的 $\mathbf{w}$ 负责提取它对应的一些隐含特征。网络的学习也就是要找到各层的 $\mathbf{w}$。

6.7 NN 训练 & 反向传播

最后一层就是一个普通的线性模型（写出来就是一个求和式），以平方错误为例，可以用随机梯度下降来优化 error 函数

$$e_n = (y_n - net(\mathbf{x}_n))^2$$

其中 $net(\cdot)$ 看做各层权重 $w_{ij}^{(l)}$ 的函数。

要求梯度就要对每个 $w_{ij}^{(l)}$ 求导嘛，但因为网络由多层，需要用 chain rule，通过一系列中间变量来求。

最后一层（输出层）：

$$e_n = (y_n - net(\mathbf{x}_n))^2 = (y_n - s_1^{(L)})^2 = \left(y_n - \sum_{i=0}^{d^{(L-1)}} w_{i1}^{(L)} x_i^{(L-1)} \right)^2$$

求之

$$\frac{\partial e_n}{\partial w_{i1}^{(L)}} = \frac{\partial e_n}{\partial s_1^{(L)}} \cdot \frac{\partial s_1^{(L)}}{\partial w_{i1}^{(L)}} = -2(y_n - s_1^{(L)}) \cdot x_i^{(L-1)}$$

令

$$\delta_1^{(L)} = -2(y_n - s_1^{(L)})$$

即 $\delta_j^{(l)}$ 就是 e_n 对第 l 层第 j 个节点 sigmoid 函数的输入 $s_j^{(l)}$ 求导的结果。其他层可以递推，比如求导求到第 l 层

$$\frac{\partial e_n}{\partial w_{ij}^{(l)}} = \frac{\partial e_n}{\partial s_j^{(l)}} \cdot \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} \cdot x_i^{(l-1)}$$

现在要求 $\delta_j^{(l)}$，这个东西就是要找 $s_j^{(l)}$ 到 e_n 的关系：

$$s_j^{(l)} \to x_j^{(l)} \to (s_1^{(l+1)}, \dots, s_k^{(l+1)}, \dots) \to \dots$$

所以

$$\delta_j^{(l)} = \frac{\partial e_n}{\partial s_j^{(l)}} = \sum_{k=1}^{d^{(l+1)}} \frac{\partial e_n}{\partial s_k^{(l+1)}} \frac{\partial s_k^{(l+1)}}{\partial x_j^{(l)}} \frac{\partial x_j^{(l)}}{\partial s_j^{(l)}} = \sum_k \delta_k^{(l+1)} w_{jk}^{(l+1)} (\tanh(s_j^{(l)}))'$$

所以 δ 可以从后往前反向计算。
反向传播计算过程：

1. 初始化各层 $w_{ij}^{(l)}$

2. 第 $t = 0, 1, \dots, T$ 轮迭代

3. 随机选择 $n \in \{1, 2, \dots, N\}$

4. 由 $\mathbf{x}^{(0)} = \mathbf{x}_n$ 正向计算出每层的输出 $x_i^{(l)}$

5. 由 $\mathbf{x}^{(0)} = \mathbf{x}_n$ 反向计算出每层 $\delta_j^{(l)}$

6. 更新所有 $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$

最后返回 $g_{NN}(\mathbf{x}) = \left(\dots \tanh \left(\sum_j w_{jk}^{(2)} \cdot \tanh \left(\sum_i w_{ij}^{(1)} x_i \right) \right) \right)$

实践上，一般每次随机多取几个点，然后并行执行前向和反向的计算，最后把各次计算出的 $x_i^{(l-1)} \delta_j^{(l)}$ 求平均后用于最后的更新中。此法称为 mini-batch。

6.8 NN 优化和 regularization

前面说的用 SGD 训练出各个 w，但这个最优解可能只是局部最优，因为整个网络的 error function 一般都不是凸函数。实践上的做法是在初始化时设置较小的 w，或者尝试随机取 w。

NN 的 VC 维大小与神经元数量和 w 数量有关，增加层数可能搞得很复杂很强大，但要小心 overfit。

Regularization 主要用普通的 weight-decay regularizer

$$\sum (w_{ij})^2$$

或 weight-elimination regularizer

$$\sum \frac{(w_{ij}^{(l)})^2}{1 + (w_{ij}^{(l)})^2}$$

用这个东西的目的是使较大的 w 减小，小的 w 直接变为 0，让最后的 w 比较稀疏。

另一种 regularization 的方法是及时停止梯度下降的迭代过程。迭代次数越少可以看做是在 w 变化的一个较小范围内做尝试，及早停止直观上来看，有助于限制复杂度。

用 validation 选择不同的迭代次数。

7 Week 7

7.1 Deep Learning Intro

思想主要是用多层的网络，逐层提取特征，从比较 raw 的特征中逐步萃取出高级特征然后做判断。对于 raw feature 较多的领域如语音、视觉有帮助。

主要难点

1. 设计网络结构。可能需要特定领域的知识
2. 控制模型复杂度。设计 regularization 方法
3. 优化目标函数。非 convex，容易陷入 local optimum，应合理设初值
4. 加速计算。一般都往 GPU 搞

7.2 Pre-train & Autoencoder

Pre-train 即一种设初值的方法，通过简单的 2 层网络来学习初值。

初值也就是权重 $w_{ij}^{(l)}$ 的初值，权重做的就是特征转换，为有助于后面学习，需要尽可能地保持原始输入的信息，并根据每层的网络结构，将前层输入转换为后层的输入，形成 $d - \tilde{d} - d$ 网络（通常 $\tilde{d} < d$ 即做压缩）目标是让最后的输出 $g_i(\mathbf{x}) \approx x_i$，即所谓 autoencoder，前后两层权重分别称 encoding/decoding weights。Autoencoder 做的即是学出原始数据的一种呈现方式。

Autoencoder 的训练所用 error function 为

$$\sum_{i=1}^{d}(g_i(\mathbf{x})-x_i)^2$$

训练数据即 $\{(\mathbf{x}_n, y_n=\mathbf{x}_n)\}$，通常当做 unsupervised。

简单的 regularization 方法是令网络前后层 $w_{ij}^{(1)}=w_{ji}^{(2)}$。

一般每两层之间学一个 autoencoder。

作为一种降低噪音影响的 regularization 方法，常在训练 autoencoder 时在 target 中加入人为的噪音，即训练数据变为 $\{(\tilde{\mathbf{x}}_n, y_1=\mathbf{x}_n)\}$，其中 $\tilde{\mathbf{x}}_n=\mathbf{x}_n+noise$。此法相当于提示 autoencoder 要将噪音剔除，找出数据的真相。

7.3 Autoencoder & PCA

考虑在 autoencoder 中做几个改变

1. 不考虑 x_0 输入

2. 前后权重相等 $w_{ij}^{(1)}=w_{ji}^{(2)}=w_{ij}$

3. $\tilde{d}<d$

则 autoencoder 输出的第 k 个分量为

$$h_k(\mathbf{x})=\sum_{j=1}^{\tilde{d}}w_{jk}^{(2)}\left(\sum_{i=1}^{d}w_{ij}^{(1)}x_i\right)$$

根据以上条件可以改写为

$$h_k(\mathbf{x})=\sum_{j=1}^{\tilde{d}}w_{kj}\left(\sum_{i=1}^{d}w_{ij}x_i\right)$$

令矩阵 $\mathbf{W}=(w_{ij})_{d\times\tilde{d}}$，autoencoder 的输出向量为

$$\tilde{\mathbf{x}}=h(\mathbf{x})=\mathbf{W}\mathbf{W}^T\mathbf{x}$$

训练 autoencoder 要优化的目标函数

$$E_{in}(h)=E_{in}(\mathbf{W})=\frac{1}{N}\sum_{n=1}^{N}\|\mathbf{x}_n-\mathbf{W}\mathbf{W}^T\mathbf{x}_n\|^2$$

因为 $\mathbf{W}\mathbf{W}^T$ 对称，所以可以做特征值分解为 $\mathbf{V}\mathbf{D}\mathbf{V}^T$，且它最多有 $\tilde{d}$ 个非零特征值（因为特征值最多 rank 个，而 $\mathbf{W}\mathbf{W}^T$ 半正定且和 $\mathbf{W}$ 等 rank）。

所以上面的 E_{in} 可以写为

$$\min_{\mathbf{V}}\min_{\mathbf{D}}\frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{V}\mathbf{I}\mathbf{V}^T\mathbf{x}_n-\mathbf{V}\mathbf{D}\mathbf{V}^T\mathbf{x}_n\right\|$$

根据正交对角化的几何意义，$\mathbf{VDV}^T$ 作用在一个向量上，相当于将它做坐标变换，然后扔掉至少 $d-\tilde{d}$ 个分量，再把其他分量做 scaling，最后把坐标转换回去。

同时，$\mathbf{V}$ 正交，保长度，所以不影响优化，内层的优化相当于

$$\min_{\mathbf{D}} \sum \|(\mathbf{I}-\mathbf{D})\cdot\ldots\|^2$$

显然要让它最大，也就是让 $\mathbf{I}-\mathbf{D}$ 中有尽可能多的 0 元素，也就是说要

$$\min_{\mathbf{V}} \sum_{n=1}^{N} \left\| \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{I}_{d-\tilde{d}} \end{bmatrix} \mathbf{V}^T \mathbf{x}_n \right\|^2$$

也就是

$$\max_{\mathbf{V}} \sum_{n=1}^{N} \left\| \begin{bmatrix} \mathbf{I}_{\tilde{d}} & 0 \\ 0 & 0 \end{bmatrix} \mathbf{V}^T \mathbf{x}_n \right\|^2$$

如果 $\tilde{d}=1$，那就只有 $\mathbf{V}^T$ 的第一行 $\mathbf{v}$ 会起作用，相当于

$$\max_{\mathbf{v}} \sum_{n=1}^{N} \mathbf{v}^T \mathbf{x}_n \mathbf{x}_n^T \mathbf{v}$$

约束为 $\mathbf{v}^T\mathbf{v}=1$，因为 $\mathbf{V}$ 里面都是规范正交化的向量。

这个东西根据拉格朗日乘数法，可知最优的 $\mathbf{v}$ 应满足

$$\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T \mathbf{v} = \lambda \mathbf{v}$$

也就是说最优解必须是 $\mathbf{X}^T\mathbf{X}$ 的特征向量。这个东西代入原来要优化的式子可得最大值为 λ，所以我们要找的最优的 $\mathbf{v}$ 就等于 $\mathbf{X}^T\mathbf{X}$ 的最大特征值对应的特征向量。

同理，对一般的 $\tilde{d}$，可以得到类似结论，即最优解必须是 $\mathbf{X}^T\mathbf{X}$ 的最大的几个特征值对应的特征向量。

直观上来看，autoencoder 要做的是找到与 $\mathbf{x}_n$ 最“match”的权重。

这个东西基本就是 PCA。也可以在做 PCA 之前先把每个数据变为

$$\mathbf{x}_n \leftarrow \mathbf{x}_n - \bar{\mathbf{x}}$$

即将均值归为 0，然后计算 $\mathbf{X}^T\mathbf{X}$ 最大的 $\tilde{d}$ 个特征向量，最后返回 encode 后的特征 $\Phi(\mathbf{x})=\mathbf{W}(\mathbf{x}-\bar{\mathbf{x}})$。

Autoencoder 也可以看做是降维，找到尽量保持原数据信息的表示方式，并降低 d 到 $\tilde{d}$。

7.4 Radial Basis Function Network

RBF 就是一种距离衡量指标，RBF 网络就是用一堆的中心来对数据进行类似投票的动作，然后把所有结果线性组合起来，即

$$h(\mathbf{x}) = output\left(\sum_{m=1}^{M} \beta_m RBF(\mathbf{x}, \boldsymbol{\mu}_m) + b\right)$$

其中 $\boldsymbol{\mu}_m$ 即各个 RBF 的中心，β_m 为投票权重。

在 Gaussian RBF + SVM 模型中，M 即支持向量数，$\boldsymbol{\mu}_m$ 为支持向量，β_m 即 SVM dual 的 $\alpha_m y_m$。

RBF 网络学习任务即找到 $\boldsymbol{\mu}_m$ 和 β_m。

7.5 RBF Network Training & k-Means

若中心数 $M = N$，称 full RBF network，每个中心值 $\boldsymbol{\mu}_m = \mathbf{x}_m$。此举之思想即认为每个 $\mathbf{x}_m$ 都会影响与其相似的 $\mathbf{x}$，例如对于二元分类，每个 $\mathbf{x}_m$ 按 $\mathbf{x}$ 与之距离为其投 y_m。

因为 RBF 是距离指标，所以可以简化为只考虑与 $\mathbf{x}$ 最近的一个或几个 $\mathbf{x}_m$，称为 k-nearest neighbor （KNN)。此法与 full RBF network 一样，都是在训练时偷懒，测试时要花费较高的计算代价。

RBF network 做的事情相当于对每个 $\mathbf{x}_n$ 做特征变换为

$$\mathbf{z}_n = (RBF(\mathbf{x}_n, \mathbf{x}_1), RBF(\mathbf{x}_n, \mathbf{x}_2), \dots, RBF(\mathbf{x}_n, \mathbf{x}_N))$$

然后把这些 $\mathbf{z}_n$ 列为 $N \times N$ 对称矩阵 $\mathbf{Z}$，类似于 kernel 里的 matrix。

对 Gaussian RBF，只要每个 $\mathbf{x}_n$ 都不同，得到的矩阵 $\mathbf{Z}$ 便可逆。

可以计算，full RBF network + 线性回归会得到 $E_{in} = 0$，通常需要 regularization。

注意 full RBF network + regularization 和 kernel ridge regression + regression 得到的结果不同，因为 full RBF network 是在原空间做 regularization，而后者是在 kernel 的空间中做 regularization。

另一种 regularization 的方法是限制中心数量 $M \ll N$，那么学习任务就是要找到合适的代表来作为中心。注意若 $\mathbf{x}_1 \approx \mathbf{x}_2$，那么他们可以用同一个中心来代表。所以找中心的过程即 clustering，将所有数据分为 M 个集合，每个集合有自己的中心 $\boldsymbol{\mu}_m$，使得整体 clustering 的 error 最小：

$$\min_{\{S_1,\dots,S_M;\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_M\}} E_{in}(S_1,\dots,S_M;\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_M) = \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}\mathbf{1}\{\mathbf{x}_n \in S_m\}\|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

这个优化很难直接做，所以先考虑当 $\boldsymbol{\mu}_m$ 都固定时，要做分组的动作，只需要找到距离每个 $\mathbf{x}_n$ 最近的 $\boldsymbol{\mu}_m$ 即可。

而当集合固定时，要找最优的中心，可以对上式所有 $\boldsymbol{\mu}_m$ 求导得

$$\nabla E_{in} = -2\sum_{n=1}^{N}\mathbf{1}\{\mathbf{x}_n \in S_m\}(\mathbf{x}_n - \boldsymbol{\mu}_m) = -2\left(\left(\sum_{\mathbf{x}_n \in S_m}\mathbf{x}_n - |S_m|\boldsymbol{\mu}_m\right)\right)$$

显然这个最优解应该是取每个 S_m 的均值。

于是根据上面这种交替优化的步骤得到 k-means:

1. 初始化 $\boldsymbol{\mu}_k$ 为 k 个随机选择的 $\mathbf{x}_n$

2. 重复交替优化步骤：先按 $\boldsymbol{\mu}$ 分组，再求各组均值更新 $\boldsymbol{\mu}$

3. 不断迭代直到收敛

收敛性是可以保证的：因为优化过程不断减小 E_{in}，而其下限为 0。k-means 对中心的初始化比较敏感。

将 k-means 结合 RBF network 就得到 RBF network 的训练过程：

1. 用 k-means 找到 M 个中心

2. 对数据做转换 $\Phi(\mathbf{x}) = (RBF(\mathbf{x}, \boldsymbol{\mu}_1), RBF(\mathbf{x}, \boldsymbol{\mu}_2), \ldots, RBF(\mathbf{x}, \boldsymbol{\mu}_M))$

3. 在 $\{(\Phi(\mathbf{x}_n), y_n)\}$ 上用线性模型找到投票权重 β

4. 返回最终 hypothesis

8 Week 8

8.1 Matrix Factorization Intro

问题：电影推荐，数据是 abstract feature，即非具有特定含义的数值。总共 N 个用户，M 部电影，数据记录了某用户 n 为某电影 m 的评分 r_{nm}。对第 m 部电影而言，与之相关的所有用户评分数据为

$$\mathcal{D}_m = \{(\tilde{\mathbf{x}_n} = (n), y_n = r_{nm})\}$$

现在把这种 categorical feature 改写为 binary encoded vector，即每种 category 都在自己对应的位置上有 1，否则为 0，上面的数据变为

$$\mathcal{D}_m = \{(\mathbf{x}_n = [0, 0, \ldots, 1, \ldots, 0]^T, y_n = r_{nm})\}$$

再把每个用户的所有评分都聚合起来

$$\mathcal{D}_m = \{(\mathbf{x}_n = [0, 0, \ldots, 1, \ldots, 0]^T, \mathbf{y}_n = [r_{n1}, \ldots, r_{nM}]^T)\}$$

注意其中有些 r_nm 是未知值，因为此用户可能没看过某些电影。

现在的任务时，从这种抽象特征中提取可用的特征。采用类似 autoencoder 的两层网络来实现，网络中不含有每层的 x_0 输入，也不用 sigmoid 函数。

将第一层的所有权重记为矩阵 $\mathbf{V}_{\tilde{d} \times N}$，第二层权重记为矩阵 $\mathbf{W}_{\tilde{d} \times M}$，所以整个网络的输出为

$$h(\mathbf{x}) = \mathbf{W}^T \mathbf{V} \mathbf{x}$$

因为输入 $\mathbf{x}_n$ 只有第 n 位为 1，其他都为 0，所以 $\mathbf{Vx}$ 相当于只提取了矩阵 $\mathbf{V}$ 的第 n 列，即

$$h(\mathbf{x}_n) = \mathbf{W}^T \mathbf{v}_n$$

进一步，把 $\mathbf{Vx}$ 看做特征转换，对于第 m 部电影，只有 $\mathbf{W}^T$ 的第 m 行参与计算，所以对于第 m 部电影

$$h_m(\mathbf{x}) = \mathbf{w}_m^T \Phi(x)$$

也就是对每个电影都是一个线性模型。

对全部数据而言，根据平方错误，有

$$E_{in}(\mathbf{w}_m, \mathbf{v}_n) = \frac{1}{\sum_{m=1}^{M} |\mathcal{D}_m|} \sum_{n,m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2$$

现在把每个用户的所有评分写成矩阵 $\mathbf{R}_{N\times M}$，则我们的任务是让

$$\mathbf{w}_m^T \mathbf{v}_n \approx r_{nm}$$

即

$$\mathbf{R} \approx \mathbf{V}^T \mathbf{W}$$

也就是说，要找到一个矩阵分解。

8.2 Matrix Factorization Learning

根据 loss

$$\min_{\mathbf{W},\mathbf{V}} E_{in}(\mathbf{w}_m, \mathbf{v}_n) \propto \sum_{n,m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2 = \sum_{m=1}^{M} \left(\sum_{(\mathbf{x}_n, r_{nm}) \in \mathcal{D}_m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n) \right)^2$$

此式用类似于 k-means 的方法做交替优化，因为固定 $\mathbf{v}_n$ 时，每个 $\mathbf{w}_m$ 只在对应的项出现在 sum 中才会影响求导。而内积是可以交换的，所以 $\mathbf{w}_m$ 和 $\mathbf{v}_n$ 的位置是同等的，得到交替优化的算法：

1. 随机初始化 $\mathbf{w}_m$, $\mathbf{v}_n$

2. 优化 $\mathbf{w}_m$，对 $\{(\mathbf{v}_n, r_{nm})\}$ 做线性回归

3. 优化 $\mathbf{v}_n$，对 $\{(\mathbf{w}_m, r_{nm})\}$ 做线性回归

4. 直到收敛

此法史称 alternating least squares。

8.3 Learn by SGD

也可以用 SGD 来优化，因为内层求梯度之后外面还有个求和，符合 SGD 的模式。Error function

$$error(n, m, r_{nm}) = (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2$$

梯度

$$\nabla_{\mathbf{v}_n} error = -2(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)\mathbf{w}_m$$
$$\nabla_{\mathbf{w}_m} error = -2(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)\mathbf{v}_n$$

优化算法：

1. 随机初始化 $\mathbf{w}_m$, $\mathbf{v}_n$

2. 每一轮 $t = 0, 1, \ldots, T$

3. 随机选择 r_{nm}

4. 计算 $\tilde{r}_{nm} = r_{nm} - \mathbf{w}_m^T \mathbf{v}_n$

5. SGD 更新 $\mathbf{v}_n^{new} \leftarrow \mathbf{v}_n^{old} + \eta \cdot \tilde{r}_{nm} \mathbf{w}_m^{old}$, $\mathbf{w}_m^{new} \leftarrow \mathbf{w}_m^{old} + \eta \cdot \tilde{r}_{nm} \mathbf{v}_n^{old}$