Notes for *Machine Learning: A Probabilistic Perspective* by Kevin P. Murphy

Kevin O'Connor

Chapter 13: *Sparse Linear Models*

Goal: select sets of relevant variables simultaneously.

13.2: *Bayesian Variable Selection*

Suppose we have latent variables, $\gamma_j = \mathbb{1}(\text{feature j relevant})$. Then compute posterior of $\gamma = (\gamma_1, ..., \gamma_D)$,

$$p(\gamma|\mathcal{D}) = \frac{\exp\{-f(\gamma)\}}{\sum_{\gamma'} \exp\{-f(\gamma)\}} \qquad \text{where} \quad f(\gamma) = -\Big(\log p(\mathcal{D}|\gamma) + \log p(\gamma)\Big)$$

Two ways of estimating γ:

1. MAP: $\hat{\gamma} = \text{argmax}\, p(\gamma|\mathcal{D}) = \text{argmin}\, f(\gamma)$. However this is often not representative of full posterior mass.
2. Median model: $\hat{\gamma} = \{j : p(\gamma_j = 1|\mathcal{D}) > 0.5\}$.

13.2.1: *Spike and Slab Model*

Let π_0 be the probability that a feature is relevant. Then define the hyper-prior on γ,

$$p(\gamma) = \prod_{j=1}^{D} \text{Ber}(\gamma_j|\pi_0) = \pi_0^{\|\gamma\|_0}(1-\pi_0)^{D-\|\gamma\|_0}$$

Define the prior on w_j

$$p(w_j|\sigma^2, \gamma_j) = \begin{cases} \delta_0(w_j) & \gamma_j = 0 \\ \mathcal{N}(w_j|0, \sigma^2\sigma_w^2) & \gamma_j = 1 \end{cases}$$

Then assuming Gaussianity, we can write the likelihood in terms of only features with $\gamma_j = 1$ (denoted by the subscript, γ).

$$p(\mathcal{D}|\gamma) = \iint \mathcal{N}(y|X_\gamma w_\gamma, \sigma^2 I_N)\mathcal{N}(w_\gamma|0_\gamma, \sigma^2\sigma_w^2 I_\gamma)p(\sigma^2)dw_\gamma d\sigma^2$$

If the marginal likelihood is intractable, can approximate with BIC.

$$\log p(\mathcal{D}|\gamma) \approx \log p(y|X, \hat{w}_\gamma, \sigma^2) - \frac{\|\gamma\|_0}{2}\log N$$

$$\log p(\gamma|\mathcal{D}) \approx \log p(y|X, \hat{w}_\gamma, \sigma^2) - \frac{\|\gamma\|_0}{2}\log N - \lambda\|\gamma\|_0 + \text{const}$$

where $\hat{w}_\gamma$ is ML or MAP estimate based on X_γ and $\lambda = \log((1-\pi_0)/\pi_0)$ (controls the sparsity).

13.2.2: *Bernoulli-Gaussian Model*

Also called **binary mask model**.

$$y_i|x_i, w, \gamma, \sigma^2 \sim \mathcal{N}\left(\sum_j \gamma_j w_j x_{ij}, \sigma^2\right)$$

$$\gamma_j \sim \text{Ber}(\pi_0)$$

$$w_j \sim \mathcal{N}(0, \sigma_w^2)$$

Difference between this and spike and slab:

1. Don't integrate out irrelevant coefficients.
2. $\gamma_j \to y \leftarrow w_j$ vs. $\gamma_j \to w_j \to y$.

13.3: l_1 *Regularization*

Using discrete priors like $\gamma_j \in \{0, 1\}$ can cause computation problems with finding the posterior mode. So we replace such priors with continuous ones that encourage sparsity. For example, Laplace prior

$$p(w|\lambda) \propto \prod_{j=1}^{D} \exp\{-\lambda|w_j|\}$$

$$\text{PNLL}(w) = \text{NLL}(w) + \lambda\|w\|_1$$

LASSO solution for regression vs. Ridge solution.

$$\hat{w}_k^{\text{LASSO}} = \text{sign}(\hat{w}_k^{\text{OLS}})\left(|\hat{w}_k^{\text{OLS}}| - \frac{\lambda}{2}\right)_+ \qquad \hat{w}_k^{\text{RIDGE}} = \frac{\hat{w}_k^{\text{OLS}}}{1+\lambda}$$

13.3.4: *Regularization Path*

As we increase λ, $\hat{w}(\lambda)$ gets sparser. Can visualize this by plotting $\hat{w}_j(\lambda)$ vs λ, called **regularization path**. Note that active set of non-zero coefficients only changes at some finite number of critical points. Gives rise to the LARS algorithm which is roughly as efficient as OLS.

13.3.5: *Model Selection*

We call a method which can recover the true model when $N \to \infty$, **model selection consistent**. We usually choose λ for the LASSO via cross validation to maximize predictive optimality. But this does not result in model selection consistency. Can make l_1 regularization more robust to small perturbations of the data by using Bayesian l_1 regularization or LASSO on bootstrapped data.

13.4: l_1 *Regularization: Algorithms*

13.4.1: *Coordinate Descent*

Idea: optimize coordinates one-by-one.

$$w_j^* = \text{argmin}_z\, f(w + ze_j) - f(w)$$

Can either cycle through each coordinate or select each at random.

13.5: l_1 *Regularization: Extensions*

13.5.1: *Group LASSO*

Idea: Encourage sparsity of groups of coefficients.

$$J(w) = \text{NLL}(w) + \sum_{g=1}^{G} \lambda_g \|w_g\|_2 \qquad \text{where} \quad \|w_g\|_2 = \sqrt{\sum_{j \in g} w_j^2}$$

Note that this differs from Ridge regression as we use $\|w_g\|_2$ rather than $\|w_g\|_2^2$.

13.5.2: *Fused LASSO*

Idea: want neighboring coefficients to be close to one another.

$$J(w, \lambda_1, \lambda_2) = \sum_{i=1}^{N} (y_i - w_i)^2 + \lambda_1 \sum_{i=1}^{N} |w_i| + \lambda_2 \sum_{i=1}^{N-1} |w_{i+1} - w_i|$$

Can also generalize this beyond simple chain neighbor structures.

13.5.3: *Elastic Net*

Idea: Combine Ridge and LASSO.

$$J(w, \lambda_1, \lambda_2) = \|y - Xw\|^2 + \lambda_2 \|w\|_2^2 + \lambda_1 \|w\|_1$$

Chapter 14: *Kernels*

14.1: *Introduction*

For some complex objects, not clear how to model them without assuming some generative model on their features. But if we have a notion of similarity between objects, we can use kernel methods.

14.2: *Kernel Functions*

Kernel is some function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Typically symmetric and non-negative.

14.2.1: *RBF Kernels*

A **radial basis function** (RBF) is one that only depends on $\|x - x'\|$. Gives rise to the RBF kernel,

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

Squared-exponential or Gaussian kernel,

$$\kappa(x, x') = \exp\left\{ -\frac{1}{2}(x - x')^T \Sigma^{-1} (x - x') \right\}$$

14.2.2: *Kernels for Comparing Documents*

Cosine similarity,

$$\kappa(x_i, x_{i'}) = \frac{x_i^T x_{i'}}{\|x_i\|_2 \|x_{i'}\|_2}$$

Could also transform with TF-IDF then get cosine similarity.

14.2.3: *Mercer (Positive Definite) Kernels*

Gram matrix, K, has $K_{ij} = \kappa(x_i, x_j)$. If K positive definite for any choice of inputs, called a **Mercer kernel**. Mercer kernels are nice because we can use Mercer's theorem which uses $K = U^T \Lambda U = (\Lambda^{1/2} U)^T (U \Lambda^{1/2})$ to write $\kappa(x, x') = \phi(x)^T \phi(x')$. Thus there is some function ϕ such that κ is an inner-product in the embedded space.

14.2.6: *String Kernels*

$$\kappa(x, x') = \sum_{s \in \mathcal{A}^*} w_s \phi_s(x) \phi_s(x')$$

where $\phi_s(x)$ is number of times s occurs as substring of x, $w_s \geqslant 0$, and $\mathcal{A}^*$ is set of all strings from alphabet $\mathcal{A}$.

14.3: *Using Kernels inside GLMs*

14.3.1: *Kernel Machines*

A **kernel machine** is a GLM where input vector has the form

$$\phi(x) = (\kappa(x, \mu_1), \ldots, \kappa(x, \mu_K))$$

for some K centroids $\mu_1, \ldots, \mu_K$.

14.4: *Kernel Trick*

Kernel trick: Instead of defining a feature vector in terms of kernels as above, just work with original inputs and replace inner-products with kernels.

14.4.1: *Kernelized Nearest-Neighbors*

Observe that

$$\|x_i - x_j\|_2^2 = \langle x_i, x_i \rangle + \langle x_j, x_j \rangle - 2 \langle x_i, x_j \rangle$$

So we can replace all of these inner products with kernels.

14.4.4: *Kernel PCA*

Find principal component projections in lower dimensional non-linear embedding.

14.5: *SVMs*

Consider l_2-regularized empirical risk function,

$$J(w, \lambda) = \sum_{i=1}^{N} L(y_i, \hat{y}_i) + \lambda \|w\|^2$$

Idea: promote sparsity via L rather than the penalty term.

14.5.1: *SVMs for Regression*

(Vapnik) Epsilon insensitive loss function

$$L_\epsilon(y, \hat{y}) = \begin{cases} 0 & |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon & \text{otherwise} \end{cases}$$

So it doesn't penalize you if you are within ϵ of the target. Can formulate constrained optimization problem with this loss as

$$J = C \sum_{i=1}^{N} (\xi_i^+ - \xi_i^-) + \frac{1}{2}\|w\|^2$$

where ξ_i^+, ξ_i^- are slack variables such that

$$y_i \leqslant f(x_i) + \epsilon + \xi_i^+ \qquad y_i \geqslant f(x_i) - \epsilon - \xi_i^-$$

and $\hat{y} = f(x_i) = w^T x_i + w_0$, constrained to $\xi_i^+ \geqslant 0$, $\xi_i^- \geqslant 0$. Gives standard quadratic program with $2N + D + 1$ variables, yielding solution,

$$\hat{w} = \sum_i \alpha_i x_i \qquad \alpha_i \geqslant 0$$

with sparse α. Then can kernelize as

$$\hat{y}(x) = \hat{w}_0 + \sum_i \alpha_i \kappa(x_i, x)$$

14.5.2: *SVMs for Classification*

Hinge loss: let $y \in \{0, 1\}$ then define

$$L_{hinge}(y, f(x)) = (1 - yf(x))_+$$

Then write as constrained optimization problem,

$$\min_{w, w_0, \xi} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{N} \xi_i \text{ s.t. } \xi_i \geqslant 0,\ y_i(x_i^T w + w_0) \geqslant 1 - \xi_i,\ i = 1, ..., N$$

Again we get a solution of the form

$$\hat{w} = \sum_i \alpha_i x_i$$

and thus after kernelizing,

$$\hat{y}(x) = \text{sign}\left(\hat{w}_0 + \sum_{i=1}^{N} \alpha_i \kappa(x_i, x)\right)$$

14.5.2.2: *The Large Margin Principle*

Can be shown that SVM objective is maximizing distance of decision boundary from support vectors.

14.7: *Kernels for Building Generative Models*

Smoothing kernel, $\kappa : \mathcal{X} \to \mathbb{R}$ satisfies

$$\int \kappa(x)dx = 1, \quad \int x\kappa(x)dx = 0, \quad \int x^2\kappa(x)dx > 0$$

14.7.2: *Kernel Density Estimator (KDE)*

Also known as **Parzen window estimator**. Use kernel with bandwidth h to estimate density as

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \kappa_h(x - x_i)$$

14.7.4: *Kernel Regression*

Goal: compute $f(x) = \mathbb{E}[y|x]$. Can use KDE to approximate joint density,

$$p(x, y) \approx \frac{1}{N} \sum_{i=1}^{N} \kappa_h(x - x_i)\kappa_h(y - y_i)$$

Then

$$f(x) = \frac{\sum_{i=1}^{N} \kappa_h(x - x_i)y_i}{\sum_{i=1}^{N} \kappa_h(x - x_i)}$$

Also called **Nadarya-Watson model**.