CSE 446 – Machine Learning

Taylor Blau

Maximum Likelihood Estimates

Given some model class parameterized by θ and some data $\mathcal{D}$, it is often desirable to find the parameter(s) θ with *maximum likelihood* given $\mathcal{D}$.

$$\begin{aligned}\hat{\theta}_{\text{MLE}} &= \arg\max_\theta \Pr(\mathcal{D} \mid \theta) \\ &= \arg\max_\theta \log \Pr(\mathcal{D} \mid \theta)\end{aligned}$$

For binomial random variables, recall that $f = \theta^k(1-\theta)^{n-k}$. Hoeffding's inequality states that:

$$\Pr(|\hat{\theta}_{\text{MLE}} - \theta^*| \geq \epsilon) \leq 2e^{-2n\epsilon^2}$$

Note that not all maximum likelihood estimates are unbiased; in particular the MLE for the variance of a Gaussian random variable has non-zero bias.

Linear Regression

A generic *linear regression model* is as follows. For $X \in \mathbb{R}^{n\times d}$, with true labels $y \in \mathbb{R}$ (where $y_i = w^T x_i + \epsilon^i$, with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$). We find a w such that the error is minimal.

$$\begin{aligned}\text{RSS}(w) &= \sum_{i=1}^n (y_i - x_i^T w)^2 \\ &= (y - Xw)^T(y - Xw) = ||y - Xw||_2^2\end{aligned}$$

$$\nabla_w ||y - Xw||_2^2 = -2X(y - Xw) = 0$$

Leading to the normal solution $X^T X w = X^T y$. If we instead suppose an intercept b (s.t., $y_i = x_i^T w + b + \epsilon_i$), then predict:

$$(X^T - \mu^T)\hat{w} + b, \quad \mu = \overline{x}, \widehat{b} = \overline{y}$$

Bias/variance trade-off

Define the error on the test set $\mathcal{T}$ to be:

$$\epsilon_{\text{test}} = \frac{1}{|\mathcal{T}|} \sum_{(x,y)\in\mathcal{T}} L(y_i, f_{\hat{w}}(x_i))$$

Error comes from three sources: (1) noise, (2) bias, and (3) variance. Low-complexity models have high bias (and low variance), and vice-versa.

$$\begin{aligned}&\mathbb{E}_{\mathcal{T},y|x_t}[(y - f_{\hat{w}}(x_t))^2] \\ =&\mathbb{E}_{\mathcal{T},y|x_t}[(\underbrace{(y - f_{w^*}(x_t))}_{C} + \underbrace{(f_{w^*}(x_t) - f_{\hat{w}}(x_t))}_{D})^2]\end{aligned}$$

In particular, we have:

- High-complexity functions have low bias and high variance.
- Low-complexity functions have high bias and low variance.

We also have that for fixed model complexity:

- Training sets containing one sample has no training error (i.e., pick a function through your single point) but high true error.
- As we add more points, training error goes up (i.e., bias is increasing while variance decreases) since your function (at some point) can no longer "pass through" all the points.
- This asymptotically approaches the value $\sigma + \text{bias}^2$, since var $\to 0$ as the training set grows to encompass all values.

In summary we have that:

$$\text{bias}(x) = f_{w^*}(x) - \mathbb{E}_{\text{train}}[f_{\hat{w}}(x)]$$

or how much we deviate from the "true" f. Likewise,

$$\text{var}(x) = \mathbb{E}_{\text{train}}[(f_{\hat{w}}(x) - \mathbb{E}_{\text{train}}[f_{\hat{w}}(x)])^2]$$

or how much the function varies over different training sets.

ℓ_2-regularization

Recall the non-regularized least-squares objective:

$$\begin{aligned}\hat{w}_{\text{LS}} &= \arg\min_w \sum_{i=1}^n (y_i - x_i^T w)^2 \\ &= \arg\min_w (y - Xw)^T(y - Xw)\end{aligned}$$

When $(y - Xw)^{-1}$ exists, $\hat{w} = (X^T X)^{-1} X^T y$. But, if $d > n$, we have some flat directions and thus $X^T X$ is not full-rank and is non-invertible. Then, we "mix" with the ℓ_2-norm, and obtain the ridge regression objective:

$$\hat{w}_{\text{ridge}} = \arg\min_w \left(\sum_{i=1}^n (y_i - x_i^T w)^2\right) + \lambda ||w||_2^2$$

With:

$$\hat{w}_{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y$$

Note that:

$$\lim_{\lambda\to 0} \hat{w}_{\text{ridge}} = \hat{w}_{\text{LS}}, \quad \lim_{\lambda\to\infty} \hat{w}_{\text{ridge}} = 0$$

One can see that:

- Setting λ close to 0 attains better training error, but does not generalize well.
- Setting λ large, we have both high training and testing error.

Note that λ is a hyper-parameter. We can choose an optimal λ based on k-fold cross-validation. Partition $\mathcal{T} = \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_k$, and let:

$$\text{error}_{\mathcal{T}_i} = \frac{1}{|\mathcal{T}_i|} \sum_{(x_j,y_j)\in\mathcal{T}_I} (y_j - f_{\mathcal{T}\setminus\mathcal{T}_i}(x_j))^2$$

And the k-fold error as:

$$\text{error}_{k\text{-fold}} = \frac{1}{k}\sum_{i=1}^k \text{error}_{\mathcal{T}_i}$$

Typically k-fold CV is preferred to LOOCV, since the former is tractable and the later is often not.

ℓ_1-regularization, LASSO

Note that for d large, we would prefer *sparse* solutions. These are efficient to compute, and often lead to more interpretable solutions. ℓ_2-thresholding can increase weight on correlated features when one of the correlates is removed. Instead, we prefer solutions with low ℓ_1-norm.

$$\hat{w}_{\text{LASSO}} = \arg\min_w \sum_{i=1}^n (y_i - x_i^T w)^2 + \lambda ||w||_1$$

We can optimize this naïvely using coordinate descent. Until convergence, cycle through coordinates j, doing:

$$\hat{w}_j = \begin{cases}(c_j + \lambda)/a_j & \text{if } c_j < -\lambda \\ 0 & \text{if } |c_j| \leq \lambda \\ (c_j - \lambda)/a_j & \text{if } c_j > \lambda\end{cases}$$

Where:

$$a_j = \sum_{i=1}^n x_{i,j}^2, \quad c_j = 2\sum_{i=1}^n \left(y_i - \sum_{k\neq j} x_{i,k} w_k\right) x_{i,j}$$

LASSO leads to solutions that have:

- Coordinates that are *exactly* zero-valued.
- Solutions that have greater sparsity than low ℓ_p-norm solutions ($p > 1$).
- Solutions with higher *interpretability*.

Logistic Regression

Suppose now instead we want to learn a function $f : \mathbb{R}^d \to \mathcal{K}$ where each $k \in \mathcal{K}$ is some *feature class*. Define the logistic loss of f to be:

$$\ell(f(x), y) = \mathbf{1}\{f(x) \neq y\}$$

Compute the expected loss as:

$$\mathbb{E}_{XY}[\mathbf{1}\{f(X) \neq Y\}] = 1 - \Pr(Y = f(x) \mid X = x)$$

Observe the Bayes-optimal classifier is:

$$f(x) = \arg\max_y \Pr(Y = y \mid X = x)$$

Define the Sigmoid function as:

$$\sigma(z) = \frac{1}{1+\exp(-z)} = \frac{\exp(z)}{1+\exp(z)}$$

Then, for binary $\mathcal{K}$, we have:

$$\Pr(Y = 1 \mid X = x, w) = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$

$$\Pr(Y = 0 \mid X = x, w) = \frac{1}{1+\exp(w^T x)}$$

So our decision rule becomes:

$$\frac{\Pr(Y = 1 \mid x, w)}{\Pr(Y = 0 \mid x, w)} = \exp(w^T x) \underset{Y=0}{\overset{Y=1}{\gtrless}} 1$$

Switching our classes to $\{-1, 1\}$, we have:

$$\hat{w}_{\text{MLE}} = \arg\min_w \sum_{i=1}^{n} \log(1 + \exp(-y_i x_i^T w)$$

Regularizing, we have:

$$\hat{w}_{\text{MLE}} = \arg\min_{w,b} \sum_{i=1}^{n} \log(1+\exp(-y_i(x_i^T w+b)) + \lambda||w||_2^2$$

Note that for x such that $w^T x + b = 0$, we guess that x has equal probability of either class (i.e., that $\Pr(Y = \pm 1 \mid x, w) = 1/2$).

Be sure that the normal $\hat{w}$ "points towards" the positive class points.

Gradient Descent, Stochastic Methods

Note that we can locate the minimum of a convex function using gradient descent with an appropriate step size η as follows:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w \left(\frac{1}{n} \sum_{i=1}^{n} \ell_i(w) \right) \Bigg|_{w=w^{(t)}}$$

This process can take time proportional to the size of $\mathcal{T}$, N. We can instead approximate the empirical risk by some batch I_t, and obtain *stochastic gradient descent*:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w \, \ell_{I_t}(w)|_{w=w^{(t)}}$$

In practice, stochastic gradient descent with a reasonable batch size can attain a convergence rate higher than gradient descent. With lower batch sizes, the path is less smooth.

Perceptron

The Perceptron algorithm learns a classification boundary, and updates for each $x \in \mathcal{X}$ iteratively as follows:

$$w_{t+1} := w_t + \mathbf{1}\{\text{sign}(w_t^T x) \neq y\} y_t x_t$$

If $\mathcal{X}$ is linearly separable with margin γ, then due to Block-Novikoff, we have that the algorithm makes an upper-bound of mistakes at R^2/γ^2.

Support Vector Machines (SVMs)

Our goal is to find a maximally separating hyper-plane between some set of labeled points.

$$\min_w ||w||_2^2, \text{ s.t., } y_i(x_i^T w + b) \geq 1$$

With slack:

$$\min_{w,\xi_i} ||w||_2^2 + C \sum_{i \in \mathcal{X}} \xi_i, \text{ s.t., } y_i(x_i^T w + b) \geq 1 - \xi_i$$

where $\xi_i = \max\{0, 1 - y_i(w^T x_i + b)\}$, and:

$$\partial_w \ell((x, y), w) = \mathbf{1}\{y(w^T x + b) \leq 1\}(-yx)$$

Kernel Methods

$$\begin{aligned} \widehat{w} &= \arg\min_w ||y - \Phi^T w||_2^2 + \lambda||w||_2^2 \\ &= \arg\min_\alpha ||y - K\alpha||_2^2 + \lambda \alpha^T K \alpha \end{aligned}$$

$$\alpha = (K + \lambda I)^{-1} y \Longrightarrow \widehat{w}^T \phi(x) = \widehat{\alpha}^T \Phi \phi(x)$$

Theorem 1 (Mercer). *K is a valid kernel evaluation matrix iff K is symmetric and positive semi-definite for any pointset where $K_{ij} = K(x_i, x_j)$.*

Bootstrap

To compute a confidence interval over some statistic $\widehat{\theta} = t(\mathcal{D})$, for $b \in [B]$, take D^{*b} to be sampled from $\mathcal{D}$ with replacement.

k-means

k-means is a cluster finding strategy which issues hard assignments to points in the pointset:

1. Randomly guess k cluster centers.
2. Create a partition $\pi : \mathcal{X} \to [k]$, and recompute the cluster center μ_k to be the median of x such that $\pi(x) = k$.
3. Repeat until convergence.

Principal Component Analysis (PCA)

Maximizing the variance in a k-dimensional subspace is identical to selecting the first k rows of Q^T, where $X = QDQ^T$, and $D = \text{diag}(\lambda_1, \ldots, \lambda_d)$, with $\lambda_1 \geq \cdots \lambda_d \geq 0$.

Singular Value Decomposition (SVD)

One can rewrite a matrix X as $X = USV^T$, where U is a matrix with columns of the left-singular vectors, and V^T a matrix with the rows of right-singular vectors. Observe that:

$$X^T X = (USV^T)^T (USV^T) = VS^2 V^T$$

Where XX^T and X^T are the column- and row-wise covariance matrices, respectively. Applications include: (1) de-noising, (2) compression, and (3) collaborative filtering.

Expectation Maximization (EM)

A limitation of k-means is that it gives hard assignments, and is not shape-invariant. Define the EM model for a multivariate Gaussian mixture as $\Theta = \{\pi_i, \mu_i, \Sigma_i\}_{i=1}^N$. Where z_k. Note that $p(x_i| \mid z_{ik} = 1) = \mathcal{N}(\mu_i, \Sigma_i)$, and $p(x_i) = \sum_{j \in [k]} \pi_k p(x_i \mid z_{ij} = 1)$. Thus, in the E-step:

$$\mathbb{E}[z_{ik}] = \Pr(z_{ik} = 1 \mid \Theta) = \frac{\pi_k \mathcal{N}(x_i \mid \mu_i, \Sigma_i)}{\sum_{j \in [k]} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$$

and in the M-step, we fix the z_{ik}'s and maximize the expected log-likelihood function.

Neural Networks

Neural networks combine layers of nodes, each with an associated set of connection weights and activation functions.

- NNs are *arbitrary function approximators* because they can create bounded rectangles. Likewise, they can simulate logical circuits.
- Convolutions extract local information, and can be stacked to create convolved tensors.
- Pooling reduces dimensionality, max-pool, avg-pool, etc.

Neural networks often have a large number of *hyper-parameters* and so we must conduct experiments in order to obtain optimal performance.

- *Grid search* try all combinations of hyper-parameter settings.
- *Random search* try combinations of randomly drawn hyper-parameters from an interval set. Often yields better performance, since more total possible outcomes, especially in sensitive networks.

Backprop can be computed automatically with an automatic-differentiator.