Lecture 7, Feb 6, 2024

Dual Representations of GLMs

- Consider the loss function: $\mathcal{L}(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{\phi w}\|_2^2 = \sum_{i=1}^{N} (\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) - y^{(i)})^2 + \lambda \boldsymbol{w}^T \boldsymbol{w}$
  - Setting $\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0 \implies 2 \sum_{i=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) - y^{(i)} a \right) \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) + 2\lambda \boldsymbol{w} = 0$
  - Then $\boldsymbol{w} = -\dfrac{1}{\lambda} \sum_{i=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) - y^{(i)} a \right) \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) = \sum_{i=1}^{N} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) = \boldsymbol{\Phi}^T \boldsymbol{\alpha}$
    * $\alpha_i = -\dfrac{1}{\lambda} (\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) - y^{(i)})$ are the *dual variables* while $\boldsymbol{w}$ are the *primal variables*
- Substitute $\boldsymbol{w} = \boldsymbol{\Phi}^T \boldsymbol{\alpha}$ into the loss function: $\mathcal{L}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T \boldsymbol{\Phi} \boldsymbol{\Phi}^T \boldsymbol{\Phi} \boldsymbol{\Phi}^T \boldsymbol{\alpha} - 2\boldsymbol{\alpha}^T \boldsymbol{\Phi} \boldsymbol{\Phi}^T \boldsymbol{y} + \boldsymbol{y}^T \boldsymbol{y} + \lambda \boldsymbol{\alpha}^T \boldsymbol{\Phi} \boldsymbol{\Phi}^T \boldsymbol{\alpha} = \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{K} \boldsymbol{\alpha} - 2\boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{y} + \boldsymbol{y}^T \boldsymbol{y} + \lambda \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha}$
  - $\boldsymbol{K} = \boldsymbol{\Phi} \boldsymbol{\Phi}^T \in \mathbb{R}^{N \times N}$ is the *Gram matrix*, which is real and symmetric
  - The (i, j)th entry of $\boldsymbol{K}$ is given by $K_{ij} = \boldsymbol{\phi}(\boldsymbol{x}^{(i)})^T \boldsymbol{\phi}(\boldsymbol{x}^{(j)}) = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$
  - $k \colon \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is the *kernel*
- Using the loss in terms of $\boldsymbol{K}$, we take $\nabla_{\boldsymbol{\alpha}} J = 0$ leads to $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{1})^{-1} \boldsymbol{y}$
  - With this solution for $\boldsymbol{\alpha}$ we have $\hat{f}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{w} = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\Phi}^T (\boldsymbol{K} + \lambda \boldsymbol{1})^{-1} \boldsymbol{y}$
  - Note the ith entry of $\boldsymbol{\Phi} \boldsymbol{\phi}(\boldsymbol{x})$ is $\boldsymbol{\phi}(\boldsymbol{x}^{(i)}) \boldsymbol{\phi}(\boldsymbol{x}) = k(\boldsymbol{x}^{(i)}, \boldsymbol{x})$
  - Let $\boldsymbol{k}(\boldsymbol{x}) = \{ k(\boldsymbol{x}^{(1)}, \boldsymbol{x}), \ldots, k(\boldsymbol{x}^{(N)}, \boldsymbol{x}) \}^T \in \mathbb{R}^N$
- The model can then be rewritten as $\hat{f}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{k}(\boldsymbol{x})^T (\boldsymbol{K} + \lambda \boldsymbol{1})^{-1} \boldsymbol{y} = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$
  - This is known as the *dual representation*
  - We've defined our model entirely in terms of the kernel; we don't actually need to evaluate the basis functions themselves, and only the inner products between the bases are needed
  - The choice of a kernel implicitly characterizes the feature space mapping ϕ
- Using the kernel is often much more efficient than using the basis functions explicitly
  - e.g. for the polynomial features, $k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z}) = 1 + x_1 z_1 + x_2 z_2 + x_1 x_2 z_1 z_2 + \cdots + x_1 \ldots x_D z_1 \ldots z_D = \prod_{i=1}^{D} (1 + x_i z_i)$
  - The original features would need $\mathcal{O}(2^D)$ computation time, but using the kernel this is reduced to $\mathcal{O}(D)$ for the simple product
- The kernel can also be interpreted as a similarity metric, since it takes two points from $\mathcal{X}$ and returns a real scalar

Definition

The kernel trick: Any linear method that can be written in terms of dot products ${\boldsymbol{x}^{(i)}}^T \boldsymbol{x}^{(j)}$ can be *kernelized* by replacing ${\boldsymbol{x}^{(i)}}^T \boldsymbol{x}^{(j)} \to k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$, which results in a nonlinear generalization of the linear method.

- This allows us to do kernel PCA, kernel SVM, etc
  - e.g. kernel k-NN
    * Distance computation in feature space is $\|\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{z})\|_2^2 = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{\phi}(\boldsymbol{z})^T \boldsymbol{\phi}(\boldsymbol{z}) - 2\boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z})$
    * Replace this by $k(\boldsymbol{x}, \boldsymbol{x}) + k(\boldsymbol{z}, \boldsymbol{z}) - 2k(\boldsymbol{x}, \boldsymbol{z})$ to kernelize it
- Even though we derived this result for the squared loss specifically, the *representer theorem* states that this kernel form of the model will always be able to minimize the loss

Kernel Selection

- The kernel function must define a dot product for some Hilbert space $\mathcal{F}$, which means it must be symmetric and positive semi-definite
  - Symmetry means $k(\boldsymbol{x}, \boldsymbol{z}) = k(\boldsymbol{z}, \boldsymbol{x})$
  - PSD means $\iint u(\boldsymbol{x})k(\boldsymbol{x}, \boldsymbol{z})u(\boldsymbol{z})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{z} \geq 0$ for all square integrable functions u
  - By extension this means:
    * $\boldsymbol{K}$ is positive semi-definite
    * Cauchy-Schwartz inequality: $k(\boldsymbol{z}, \boldsymbol{z}) \leq \sqrt{k(\boldsymbol{x}, \boldsymbol{x})k(\boldsymbol{z}, \boldsymbol{z})}$
    * Definiteness: $k(\boldsymbol{x}, \boldsymbol{x}) \geq 0$
  - This all makes sense intuitively if the kernel is interpreted as a distance metric
- Example kernels:
  - Linear: $k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^T\boldsymbol{z}$
  - Polynomial: $k(\boldsymbol{x}, \boldsymbol{z}) = (1 + \boldsymbol{x}^T\boldsymbol{z})^n$
  - Isotropic Gaussian: $k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{1}{\theta}\|\boldsymbol{x} - \boldsymbol{z}\|_2^2\right)$
    * $\theta > 0$ is a hyperparameter
  - Anisotropic Gaussian: $k(\boldsymbol{x}, \boldsymbol{z}) = \exp(-(\boldsymbol{x} - \boldsymbol{z})^T\boldsymbol{\Theta}^{-1}(\boldsymbol{x} - \boldsymbol{z}))$
    * $\boldsymbol{\Theta} \in \mathbb{R}^{D\times D}$ is symmetric positive definite and a hyperparameter
- We can go from kernels back to features, e.g. for the polynomial kernel:
  - $k(\boldsymbol{x}, \boldsymbol{z}) = (1 + x_1z_1 + x_2z_2 + \cdots + x_Dz_D)^n$
  - For $D = 2$ and $n = 2$, $k(\boldsymbol{x}, \boldsymbol{z}) = 1 + x_1^2z_1^2 + x_2^2z_2^2 + 2x_1z_1 + 2x_2z_2 + 2x_1z_2x_2z_2$
  - Therefore $\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix}1 & x_1^2 & x_2^2 & \sqrt{2}x_1 & \sqrt{2}x_2 & \sqrt{2}x_1x_2\end{bmatrix}$
- The feature vector can even be infinite dimensional, e.g. for the Gaussian kernel:
  - For $D = 1, \theta = 1$, $k(x, z) = \exp(-(x - z)^2)$
    $$= \exp(-x^2)\exp(-z^2)\exp(2xz)$$
    $$= \exp(-x^2)\exp(-z^2)\sum_{k=0}^{\infty}\frac{2^kx^kz^k}{k!}$$
  - Therefore $\boldsymbol{\phi}(x) = \left[\exp(-x^2) \quad \sqrt{\frac{2^1}{1!}}x^1\exp(-x^2) \quad \sqrt{\frac{2^2}{2!}}x^2\exp(-x^2) \quad \ldots\right]$
- To select the kernel, we can use prior knowledge of the target function
  - If the target function is known to be smooth (i.e. differentiable k times) then we can use a kernel that also has the same degree of smoothness
  - If the function is finitely smooth, use the Gaussian or another C^∞ kernel
  - If the function is periodic we can use a periodic kernel
  - Plenty of literate exists in this area
- *Radial basis functions* (RBFs) are kernels that are translation invariant, i.e. their value only depends on the distance between the features
  - $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = k(\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|) = k(r)$
  - Examples of RBF kernels:
    * Gaussian: $k(r) = e^{-\frac{r^2}{\theta}}$
    * Multiquadratic: $k(r) = \sqrt{1 + \frac{r^2}{\theta}}$
    * Inverse multiquadratic: $k(r) = \frac{1}{\sqrt{1 + \frac{r^2}{\theta}}}$
    * Matern kernels: a family including
      - C^0: $\exp\left(-\frac{r}{\theta}\right)$
      - C^2: $\frac{1}{1 + \frac{r}{\theta}}\exp\left(-\frac{r}{\theta}\right)$
      - C^4: $\left(3 + 3\frac{r}{\theta} + \left(\frac{r}{\theta}\right)^2\right)\exp\left(-\frac{r}{\theta}\right)$

  - All the above kernels have θ has a hyperparameter; this is the shape parameter, where larger values spread out the function and gives a higher value for larger values of r

Sparsity

- The regression model is $\hat{f}(\boldsymbol{x}, \boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$ where $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{1})^{-1} \boldsymbol{y}$
- This can be interpreted as a GLM constructed using the N basis functions $k(\boldsymbol{x}, \boldsymbol{x}^{(1)}), \ldots, k(\boldsymbol{x}, \boldsymbol{x}^{(N)})$
  - We have one basis function per data point, so this is a *dense* regression model
- Note that when $\lambda = 0$, since we have N basis functions, we will match our N training points exactly
  - This can be useful if we know that there is no noise in the training data
  - When $\lambda = 0$, $\boldsymbol{K}$ is guaranteed to be non-singular if and only if the training data points are unique
- When $\lambda > 0$, $\boldsymbol{K} + \lambda \boldsymbol{1}$ is symmetric positive definite, so we can compute the Cholesky factorization without worrying about singularities
  - Since we never formed normal equations, we never squared the condition number, so this is stable
- Computing this will take $\mathcal{O}(N^2)$ memory and $\mathcal{O}(N^3)$ time, which makes it very difficult to scale up
  - We can improve this by choosing only a subset of the basis functions, which gives us a *sparse* regression model
  - Alternatively, we can use k-means clustering to extract a set of representative points
    * Then the model is $\hat{f}(\boldsymbol{x}, \boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i k(\boldsymbol{x}, \boldsymbol{z}^{(i)})$ and $\boldsymbol{\alpha}$ is computed with the $\boldsymbol{z}$ vectors
    * This also reduces inference cost
- Sparsity is generally a good idea because:
  - Reduction in computational and inference cost
  - Reduction in memory usage
  - Makes models more interpretable
  - Prevents overfitting
- *Orthogonal Marching Pursuit*: a greedy algorithm for sparse regression
  - Procedure:
    * Set $k = 0$ and let $\mathcal{D}_\phi = \{ \phi_1, \ldots, \phi_M \}$ be a dictionary of basis functions
    * Initialize $\mathcal{I}_s^{(k)}$, the set of selected basis functions, and $\mathcal{I}_c^{(k)}$, the set of candidate basis functions
    * Initialize $\boldsymbol{r}^{(0)} = \boldsymbol{y}$ as the residual, or training error vector
    * While $\|\boldsymbol{r}^{(k)}\|_2 > \epsilon$, do:
      - $k \leftarrow k + 1$
      - Pick $i_k = \underset{i \in \mathcal{I}_c^{(k-1)}}{\operatorname{argmax}} J(\phi_i)$
        - The metric is $J(\phi_i) = \dfrac{(\boldsymbol{\Phi}_i^T \boldsymbol{r}^{(k)})^2}{\boldsymbol{\Phi}_i^T \boldsymbol{\Phi}_i}$ where $\boldsymbol{\Phi}_i$ is the ith column of $\boldsymbol{\Phi}$
        - This is an approximation of the reduction in training error as a result of choosing the ith basis function
        - Think of this as checking how much the ith basis function is in the direction of the residual error
      - Add selected basis function index to $\mathcal{I}_s^{(k)}$ and remove it from $\mathcal{I}_c^{(k)}$
      - Solve $\boldsymbol{\phi}^{(k)} \boldsymbol{w}^{(k)} \approx \boldsymbol{y}$ for the weights
        - Note $\boldsymbol{w}^{(k)} \in \mathbb{R}^k$ since in this iteration we have k basis functions
        - $\boldsymbol{\Phi}^{(k)}$ has k columns corresponding to the basis functions
      - Update the residual by $\boldsymbol{r}^{(k)} = \boldsymbol{y} - \boldsymbol{\Phi}^{(k)} \boldsymbol{w}^{(k)}$
    * The final sparse model is $\sum_{i \in \mathcal{I}_s^{(k)}} w_i \phi_i(\boldsymbol{x})$
  - Updating the weights in each iteration can be done using incremental QR factorization to save time
  - The parameter ϵ can be chosen via cross-validation, or other model selection criteria
- For GLMs, if minimizing the least squares error with l_2 regularization, we can find a more efficient

method to calculate the leave-one-out error
  - Let $\boldsymbol{A} = \boldsymbol{K}(\boldsymbol{K} + \lambda \mathbf{1})^{-1} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T$
  - Let $\hat{f}^{\setminus i}$ denote the model constructed by leaving out the ith training point
  - Then $y^{(i)} - \hat{f}^{\setminus i}(\boldsymbol{x}^{(i)}) = \dfrac{y^{(i)} - \hat{f}(\boldsymbol{x}^{(i)})}{1 - A_{ii}}$
  - Therefore the total leave-one-out error is $\dfrac{1}{N}\displaystyle\sum_{i=1}^{N}\left(\dfrac{y^{(i)} - \hat{f}(\boldsymbol{x}^{(i)})}{1 - A_{ii}}\right)^2$
    * This is a function of λ, the regularization parameter; using this we can estimate the optimal value of λ
  - This means we don't have to train the model N times for each data point we leave out, making this much more efficient
- Using l_1 regularization can also give models that are more sparse and easy to interpret
  - However with l_1 regularization we can no longer use linear algebra to obtain a closed form solution
  - Optimization algorithms need to be used in this case
- In summary:
  - If M is high or possibly infinite, use kernel methods
  - If N is high, use explicit basis functions
  - When both are high, options include greedy algorithms for sparsity, clustering, scholastic algorithms, etc

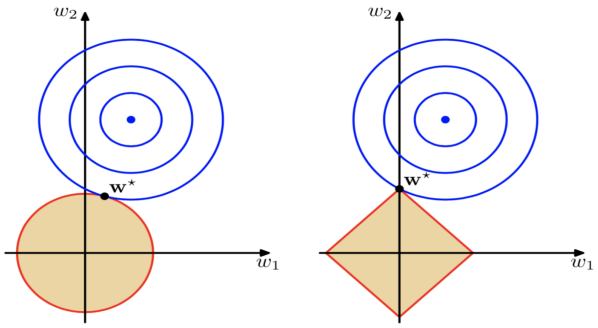

Figure 1: Comparison of l_1 vs l_2 regularization.