Lecture 6, Feb 2, 2024

Generalized Linear Models (GLMs)

Definition

Generalized Linear Models: A GLM is given by

$$\hat{f}(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{i=1}^{M-1} w_i \phi_i(\boldsymbol{x})$$

where $\boldsymbol{w}$ is a set of undetermined weigths and $\phi_i \colon \mathbb{R}^D \mapsto \mathbb{R}$ are a set of known basis functions.

- The models may be nonlinear in the inputs $\boldsymbol{x}$, but still linear in the weights $\boldsymbol{w}$, which makes is still possible to use linear techniques
- To construct a GLM we need to select the appropriate basis functions, and formulate a strategy to estimate the weights
- Let $\phi_0(\boldsymbol{x}) = 1$ (the bias term) and $\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} \phi_0(\boldsymbol{x}) \\ \phi_1(\boldsymbol{x}) \\ \vdots \\ \phi_{M-1}(\boldsymbol{x}) \end{bmatrix} \in \mathbb{R}^M$
  - Then if we define the weight vector $\boldsymbol{w} = \begin{bmatrix} w_0 & \dots & w_m \end{bmatrix}^T$, we can write $\hat{f}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$
  - We are using ϕ to map from the input space $\mathcal{X}$ to the *feature space* $\mathcal{F}$, and performing linear regression in the feature space
- Let the vector of training targets $\boldsymbol{y} = \begin{bmatrix} y^{(1)} & \dots & y^{(N)} \end{bmatrix}^T \in \mathbb{R}^N$ and $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ where the ith row contains $\boldsymbol{\phi}(\boldsymbol{x}^{(i)}) \in \mathbb{R}^M$
  - Then $\hat{\boldsymbol{y}} = \boldsymbol{\Phi} \boldsymbol{w}$
- Use the l_2 loss function $\hat{\boldsymbol{w}} = \underset{\boldsymbol{w} \in \mathbb{R}^M}{\operatorname{argmin}} \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2$
  - Again the loss function can be written as $(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w})^T (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w})$
  - We can use the same techniques for the linear model, but instead of $D+1$ weights we have M weights
  - We essentially replace $\boldsymbol{X} \in \mathbb{R}^{N \times (D+1)}$ with $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$
- Derivation:
  - $\mathcal{L}(\boldsymbol{w}) = \boldsymbol{y}^T \boldsymbol{y} + \boldsymbol{w}^T \boldsymbol{\Phi}^T \boldsymbol{\Phi} \boldsymbol{w} - 2 \boldsymbol{y}^T \Phi \boldsymbol{w}$
  - $\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi} + (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^T) \boldsymbol{w} - 2 \boldsymbol{\Phi}^T \boldsymbol{y} = 2 \boldsymbol{\Phi}^T \boldsymbol{\Phi} \boldsymbol{w} - 2 \boldsymbol{\Phi}^T \boldsymbol{y} = 0$
  - Therefore $\boldsymbol{\Phi}^T \boldsymbol{\Phi} \boldsymbol{w} = \boldsymbol{\Phi}^T \boldsymbol{y}$
- We can use the same techniques as linear models to solve the normal equations:
  - Cholesky factorization
    - Avoid this because the condition number is squared
  - (Economy) QR factorization
    - Use this only if $\boldsymbol{\Phi}$ is not rank-deficient
  - (Economy) SVD, or truncated SVD if $\boldsymbol{\Phi}$ is rank-deficient
    - Slowest, but the most stable

Polynomial Regression

- The basis functions are the univariate polynomials $\{ 1, x_i, x_i^2, \dots, x_i^p \}$ up to order p
- If $D = 1$, then the basis functions are $\phi_i = x^i$ so $\hat{f}(x) = w_0 + w_1 x + \dots + w_p x^p$
  - Note taking tensor products of higher-order univariate polynomials is not a good idea since we will generate p^D basis functions

– For arbitrary D we would have $\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_D \\ x_1 x_2 \\ \vdots \\ x_{D-1} x_D \\ \vdots \\ x_1 x_2 \ldots x_D \end{bmatrix}$

– We can circumvent this with the *kernel trick* covered later

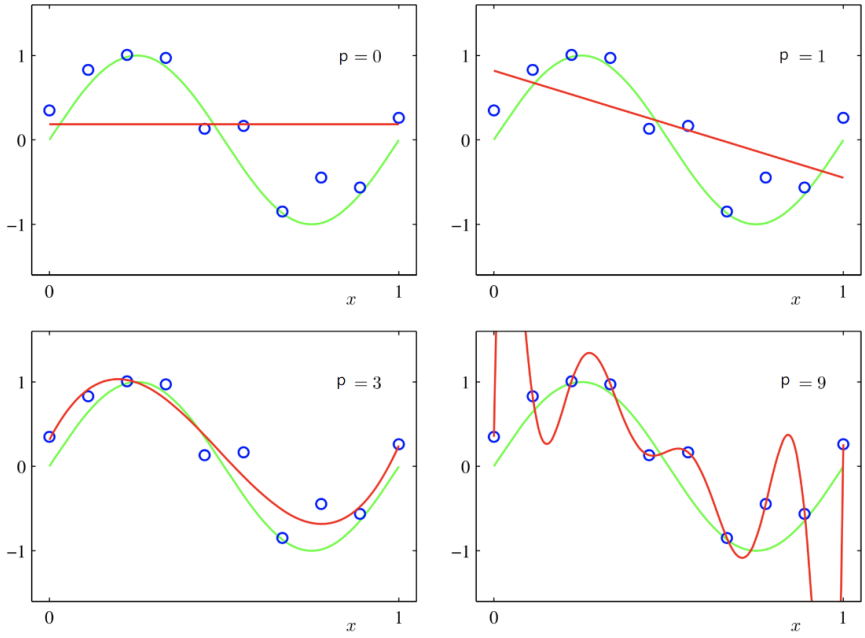

Figure 1: 1D polynomial regression for various values of p. (Note $M = p + 1$.)

- Results for various values of p are shown above
  - Notice that for smaller values of p the model doesn't fit well since it doesn't have enough complexity (underfitting)
  - But for large values of p, the polynomial matches the training points perfectly but approximates the underlying function poorly
- To prevent overfitting, we need to restrict the number of features M (which in this case restricts the degree of the polynomial
  - Increasing the number of data points also helps but we often can't just get more data
  - What if we know the underlying model is complex but we don't have enough data points?
  - How do we deal with noise?

Regularization

- One pattern we may notice is that when the model is overfitting (M too large), the weights start becoming very big in magnitude
- *Regularization* tries to keep the magnitudes of the weights reasonably small, as a way to prevent overfitting
- To keep the weight small, we can introduce the norm of the weights to the loss function, so the model is penalized for having weights that are too large

Definition

Ridge Regression Method: Choose the weights as

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}\in\mathbb{R}^M}{\operatorname{argmin}} |\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}|_2^2 + \lambda \|\boldsymbol{w}\|_2^2$$

where λ is the *regularization parameter*.

- Note w_0 is often excluded from the regularization term
- The regularized loss function is also quadratic in $\boldsymbol{w}$, so we can use the same steps as before
  - Expanded loss: $\boldsymbol{y}^T\boldsymbol{y} + \boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w} - 2\boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{y} + \lambda\boldsymbol{w}^T\boldsymbol{w}$
  - $\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 2\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w} - 2\boldsymbol{\Phi}^T\boldsymbol{y} + 2\lambda\boldsymbol{w} = 0$
  - Rearrange: $\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w} + \lambda\boldsymbol{w} = \boldsymbol{\Phi}^T\boldsymbol{y} \implies (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{1})\boldsymbol{w} = \boldsymbol{\Phi}^T\boldsymbol{y}$
    - Therefore l_2 regularization is equivalent to adding a small positive perturbation to the diagonal of $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$
    - We saw this in a previous lecture – this also helps with ill-conditioning
    - If λ is sufficiently large we can avoid ill-conditioning completely
- Using SVD: $\boldsymbol{\Phi} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  - $((\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^T\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T + \lambda\mathbf{1})\boldsymbol{w} = (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^T\boldsymbol{y}$
  - Simply to get $\boldsymbol{V}(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} + \lambda\mathbf{1})\boldsymbol{V}^T\boldsymbol{w} = \boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{y}$
  - Multiply each side by $\boldsymbol{V}^T$ to get $(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} + \lambda\mathbf{1})\boldsymbol{V}^T\boldsymbol{w} = \boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{y}$
  - Therefore $\boldsymbol{w} = \boldsymbol{V}(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} + \lambda\mathbf{1})^{-1}\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{y}$
  - This can be rewritten as $\hat{\boldsymbol{w}}(\lambda) = \sum_{i=1}^{M} \boldsymbol{v}_i \dfrac{\sigma_i \boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i^2 + \lambda}$
    - $\boldsymbol{v}_i, \boldsymbol{u}_i$ are the ith columns of $\boldsymbol{V}$ and $\boldsymbol{U}$
    - If $0 \approx \sigma_i \ll \lambda$ this goes to 0
    - If $\sigma_i \gg \lambda$ this goes to the original unregularized solution
- The regularization has almost no impact on the contributions of large singular values but zeros out the contribution of smaller singular values

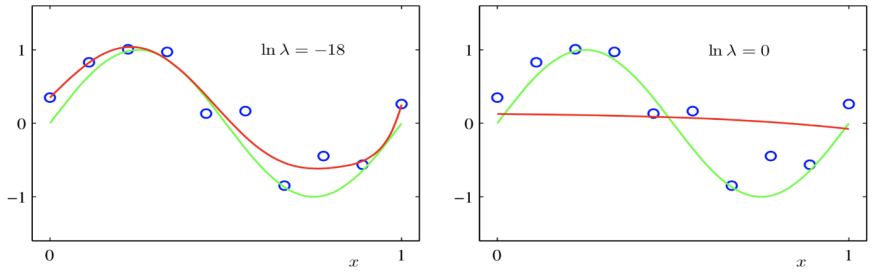

Figure 2: The same polynomial regression from above for $p = 9$, with different values of λ.

- λ is an important hyperparameter
  - Notice that with a reasonable value of λ we have a pretty good model even at $p = 9$
  - However if the regularization is too extreme, the model will underfit as the loss is too focused on minimizing $\|\boldsymbol{w}\|_2^2$
- To estimate λ we can again use ν-fold cross-validation just like we chose k for k-NN
  - If the training dataset is small we can use leave-one-out cross-validation (i.e. $\nu = 1$)
  - There are fast algorithms for calculating this
  - Use cross-validation to select the best value of λ, then retrain the model on all the data using this new value