Lecture 4, Jan 23, 2024

Linear Regression

- A linear model in general is represented by $\hat{f}(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{j=1}^{D} w_j x_d$
  - $\boldsymbol{w} = \{ w_0, \dots, w_D \}^T \in \mathbb{R}^{D+1}$ are undetermined weights of the model
  - This is a parametric supervised learning technique
- Using least squares loss gives the optimization problem: $\hat{\boldsymbol{w}} = \underset{\boldsymbol{w} \in \mathbb{R}^{D+1}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y^{(i)} - w_0 - \sum_{j=1}^{D} w_j x_j^{(i)} \right)^2$
  - Let the dummy feature $x_0 = 1$, then we have $\boldsymbol{x} = \{ x_0, \dots, x_D \}^T \in \mathbb{R}^{D+1}$ so $\hat{f}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}$
  - Let $\boldsymbol{X} \in \mathbb{R}^{N \times (D+1)}$ such that the ith row contains $\boldsymbol{x}^{(i)}$, i.e. $\boldsymbol{X}_{ij} = x_j^{(i)}$; this allows us to write the vector of predictions as $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w} \in \mathbb{R}^N$
  - Let $\boldsymbol{y} = \{ y^{(1)}, \dots, y^{(N)} \}^T \in \mathbb{R}^N$
- The problem is then $\hat{\boldsymbol{w}} = \underset{\boldsymbol{w} \in \mathbb{R}^{D+1}}{\operatorname{argmin}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$
  - The loss function is $\mathcal{L}(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})$
  - $$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} &= \frac{\partial}{\partial \boldsymbol{w}}(\boldsymbol{y}^T\boldsymbol{y} + \boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - 2\boldsymbol{y}^T\boldsymbol{X}\boldsymbol{w}) \\ &= 2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - 2\boldsymbol{X}^T\boldsymbol{y} \\ &= 2\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) = \boldsymbol{0} \end{aligned}$$
    - Note: $\frac{\partial}{\partial z}(\boldsymbol{z}^T\boldsymbol{A}\boldsymbol{z}) = (\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{z}$, $\frac{\partial}{\partial \boldsymbol{z}}(\boldsymbol{A}\boldsymbol{z}) = \boldsymbol{A}^T$
- Therefore we need to solve $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y}$
  - $\boldsymbol{X}^T\boldsymbol{X}$ is invertible if $\boldsymbol{X}$ is full rank, i.e. if the features are linearly independent
    - Note equations of this form are known as *normal equations*
  - Can be interpreted as a projection scheme since we are enforcing that $(\boldsymbol{y} - \hat{\boldsymbol{y}}) \perp \boldsymbol{X}_i$ for all columns $\boldsymbol{X}_i$ of $\boldsymbol{X}$ (the residual should be orthogonal to the column space)
- We're essentially trying to solve $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$ where $\boldsymbol{X} \in \mathbb{R}^{N \times (D+1)}, \boldsymbol{w} \in \mathbb{R}^{D+1}, \boldsymbol{y} \in \mathbb{R}^N$
  - The problem is *overdetermined* if $N > D+1$ (i.e. we have more data points than feature dimensions, so $\boldsymbol{X}$ is tall and skinny)
    - We therefore cannot find $\boldsymbol{w}$ to solve this equation, so we can only minimize the residual
  - The problem is *undetermined* if $N < D+1$ (i.e. we have more dimensions than data points, so $\boldsymbol{X}$ is short and fat)
    - This would have an infinite number of solutions, so we need to impose additional constraints

Solving for the Weights

- Cholesky decomposition: $\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{R}^T\boldsymbol{R}$ where $\boldsymbol{R} \in \mathbb{R}^{(D+1) \times (D+1)}$ is upper triangular
  - Note this is only possible since $\boldsymbol{X}^T\boldsymbol{X}$ is symmetric positive definite if it is full rank
  - Then we have $\hat{\boldsymbol{w}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} = \boldsymbol{R}^{-1}\boldsymbol{R}^{-T}\boldsymbol{X}^T\boldsymbol{y}$
    - Note $\boldsymbol{R}^{-T} = (\boldsymbol{R}^{-1})^T = (\boldsymbol{R}^T)^{-1}$
  - Computationally this involves a forward and backward substitution to invert the upper and lower triangular matrices
    - First solve for $\boldsymbol{z} = \boldsymbol{R}^{-T}\boldsymbol{x}^T\boldsymbol{y}$, then $\boldsymbol{w} = \boldsymbol{R}^{-1}\boldsymbol{z}$
    - Both inverses are easy to compute due to them being triangular
  - Note it is common to add a small perturbation, replacing $\boldsymbol{X}^T\boldsymbol{X}$ with $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ to prevent ill-conditioning; this is equivalent to l_2 regularization
  - Cost: $\mathcal{O}(N(D+1)^2 + \frac{1}{3}(D+1)^3)$
    - Computing $\boldsymbol{R}$ takes $mn^2 + \frac{1}{3}n^3$ flops
- Economic QR (aka. reduced or thin QR): $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$ where $\boldsymbol{Q} \in \mathbb{R}^{N \times (D+1)}$ is orthonormal, $\boldsymbol{R} \in$

$\mathbb{R}^{(D+1)\times(D+1)}$ is upper-triangular
  - $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y} \implies \boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{Q}^T\boldsymbol{R}\boldsymbol{w} = \boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{y} \implies \boldsymbol{R}^T\boldsymbol{R}\boldsymbol{w} = \boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{y}$
  - Then we have $\hat{\boldsymbol{w}} = \boldsymbol{R}^{-1}\boldsymbol{Q}^T\boldsymbol{y}$
  - Note instead of directly inverting R, we again use a backward substitution
  - This method can fail when $\boldsymbol{X}$ is nearly rank-deficient (i.e. two data points being close together); in this case, SVD is a more robust option
  - Cost: $\mathcal{O}(2N(D+1)^2 + \frac{2}{3}(D+1)^3)$ (approximate)
    * QR factorization costs about $2mn^2$ flops; for $m \gg n$ Cholesky is faster, but only a factor of 2 at most
- Singular value decomposition: $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ where $\boldsymbol{U} \in \mathbb{R}^{N\times N}, \boldsymbol{V} \in \mathbb{R}^{(D+1)\times(D+1)}$ are orthogonal and $\boldsymbol{\Sigma} \in \mathbb{R}^{N\times(D+1)}$ is rectangular diagonal
  - Note we can write this as $\boldsymbol{X} = \begin{bmatrix}\boldsymbol{U}_1 & \boldsymbol{U}_2\end{bmatrix}\begin{bmatrix}\boldsymbol{\Sigma}_1 \\ \boldsymbol{0}\end{bmatrix}\boldsymbol{V}^T$ or $\boldsymbol{X} = \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{V}^T$
  - $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 = \|\boldsymbol{U}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})\|_2^2 = \left\|\begin{bmatrix}\boldsymbol{U}_1^T\boldsymbol{y} \\ \boldsymbol{U}_2\boldsymbol{y}\end{bmatrix} - \begin{bmatrix}\boldsymbol{\Sigma}_1\boldsymbol{V}^T\boldsymbol{w} \\ \boldsymbol{0}\end{bmatrix}\right\|_2^2 = \|\boldsymbol{U}_1^T\boldsymbol{y} - \boldsymbol{\Sigma}_1\boldsymbol{V}^T\boldsymbol{w}\|_2^2 + \|\boldsymbol{U}_2^T\boldsymbol{y}\|_2^2$
    * Since $\boldsymbol{U}$ is orthogonal, we can multiply any vector by it and not change the norm
  - We can now minimize this by choosing $\hat{\boldsymbol{w}} = \boldsymbol{V}\boldsymbol{\Sigma}_1^{-1}\boldsymbol{U}_1^T\boldsymbol{y} = \sum_{i=1}^{D+1} \boldsymbol{v}_i \frac{\boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i}$
    * The summation format is more efficient since $\boldsymbol{\Sigma}$ is diagonal
    * If some singular values are very small, we can truncate this summation for better numerical stability
  - Alternatively the same result can be obtained by simply substituting the SVD into the original expression
  - Cost: $\mathcal{O}(2N(D+1)^2 + 11N(D+1)^3)$ (approximate)
- Moore-Penrose pseudoinverse: $\boldsymbol{X}^\dagger = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T = \boldsymbol{V}\boldsymbol{\Sigma}^\dagger\boldsymbol{U}^T$
  - Then $\hat{\boldsymbol{w}} = \boldsymbol{X}^\dagger\boldsymbol{y}$ when $\boldsymbol{X}$ is full rank (so $\boldsymbol{X}^T\boldsymbol{X}$ is symmetric positive definite)
  - Using QR and SVD we can also write $\boldsymbol{X}^\dagger = \boldsymbol{R}^{-1}\boldsymbol{Q}^T$ or $\boldsymbol{X}^\dagger = \boldsymbol{V}\boldsymbol{\Sigma}_1^{-1}\boldsymbol{U}_1^T$
  - If $\boldsymbol{X}$ is rank deficient, then we can take $\hat{\boldsymbol{w}} = \boldsymbol{V}\boldsymbol{\Sigma}_1^\dagger\boldsymbol{U}_1^T\boldsymbol{y}$
    * $\boldsymbol{\Sigma}_1^\dagger = \operatorname{diag}\{\sigma_1^\dagger, \dots, \sigma_{D+1}^\dagger\}$ and $\sigma_i^\dagger = \begin{cases}\frac{1}{\sigma_i} & \sigma_i > 0 \\ 0 & \text{otherwise}\end{cases}$
- The *condition number* for linear least-squares is defined as $\kappa(\boldsymbol{X}) = \|\boldsymbol{X}\|\|\boldsymbol{X}^\dagger\| = \frac{\sigma_{\max}}{\sigma_{\min}}$
  - This is a measure of the sensitivity of the weights to perturbations in the training data
  - High condition numbers can occur in learning problems where the features are strongly correlated
  - Rule of thumb: one digit of precision is lost for every power of 10 in the condition number
    * e.g. IEEE doubles have 16 digits of accuracy, so if a matrix has a condition number of 10^{10} we will only get 6 digits of accuracy
  - Note $\kappa(\boldsymbol{X}^T\boldsymbol{X}) = (\kappa(\boldsymbol{X}))^2$, i.e. when solving normal equations we square the condition number!
    * $\kappa(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}) \leq \kappa(\boldsymbol{X}^T\boldsymbol{X})$ for all positive λ
  - On the other hand, performing QR and SVD decomposition keeps the same condition number, so using these methods are a lot more stable
- In general SVD is more expensive than QR and Cholesky, but more numerically stable and can handle rank deficiencies
  - Which one to use is problem dependent
  - From Cholesky to QR to SVD we have increasing stability but also computational cost
- Storing $\boldsymbol{X}, \boldsymbol{y}$ use $\mathcal{O}(ND) + \mathcal{O}(N)$ memory
  - Using economy QR and SVD will require additional $\mathcal{O}(ND)$ memory
    * Full QR and SVD is never practical for large datasets!
  - If the problem is too large to fit into memory, we will need iterative methods that compute the result term-by-term
- Another alternative is to use gradient descent

- $\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{2N}\vec{\nabla}_{\boldsymbol{w}}\mathcal{L} = \boldsymbol{w} - \frac{\alpha}{N}\boldsymbol{X}^T(\hat{\boldsymbol{y}} - \boldsymbol{y}) = \boldsymbol{w} - \frac{\alpha}{N}\sum_{i=1}^{N}(\hat{y}^{(i)} - y^{(i)})\boldsymbol{x}^{(i)}$
- Each iteration requires an additional $\mathcal{O}(ND)$ cost

Underdetermined Least Squares

- Assume that $\text{rank}(\boldsymbol{X}) = N$
- We need to impose additional constraints to get a unique solution
- Heavily underdetermined equations routinely arise in the field of *compressive sensing* and bioinformatics
- One approach is to use $\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}\in\mathbb{R}^{D+1}}{\text{argmin}}\|\boldsymbol{w}\|_2^2$ such that $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$
    - This gives the *minimum norm solution* to the least squares-problem
    - Let $\boldsymbol{\lambda} \in \mathbb{R}^N$ be the Lagrange multipliers
    - The Lagrangian is $L(\boldsymbol{w}, \boldsymbol{\lambda}) = \boldsymbol{w}^T\boldsymbol{w} + \boldsymbol{\lambda}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$
    - The optimality condition is $\vec{\nabla}_{\boldsymbol{w}}L = 2\boldsymbol{w} + \boldsymbol{X}^T\boldsymbol{\lambda} = 0$ and $\vec{\nabla}_{\boldsymbol{\lambda}}L = \boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} = 0$
        * Solve: $\boldsymbol{w} = -\frac{1}{2}\boldsymbol{X}^T\boldsymbol{\lambda}$ and $\boldsymbol{\lambda} = -2(\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{y}$
    - Therefore $\hat{\boldsymbol{w}} = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{y}$
        * Note in practice we do not calculate this inverse explicitly but instead use a factorization scheme for better stability
    - Other options for constraints also exist such as minimizing the 1-norm
- Using QR factorization: factorize $\boldsymbol{X}^T = \boldsymbol{Q}\boldsymbol{R}$, then $\hat{\boldsymbol{w}} = \boldsymbol{Q}\boldsymbol{R}^{-T}\boldsymbol{y}$
- Using economy SVD: $\boldsymbol{X}^T = \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{V}^T$, then $\hat{\boldsymbol{w}} = \boldsymbol{U}_1\boldsymbol{\Sigma}_1^{-1}\boldsymbol{V}^T\boldsymbol{y}$
    - If $\boldsymbol{X}^T$ is not full rank we can use the same thresholding technique as overdetermined least squares (ignoring nearly zero singular values)

Regression Models for Classification

- For a binary classification problem ($y = +1, -1$), consider a model that minimizes the l_2 loss and makes predictions as $\text{sgn}\,\hat{f}(\boldsymbol{x})$

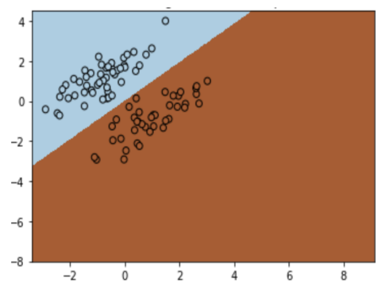

Figure 1: The case of two linearly separable classes that are in clusters.

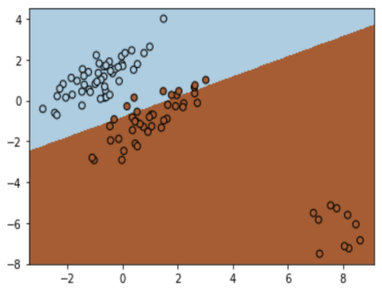

Figure 2: The model after including additional training points.

- Notice that in the example above, the model became much worse after including the additional data
  - This is because we used a loss function that is inappropriate for classification!
- Make the labels instead $y \in \{0, 1\}$, and normalize the predictions using the logistic (aka sigmoid) function: $\sigma(z) = \dfrac{1}{1+e^{-z}}$ and make predictions as $\hat{y} = \sigma(\boldsymbol{w}^T\boldsymbol{x})$ and threshold at 0.5

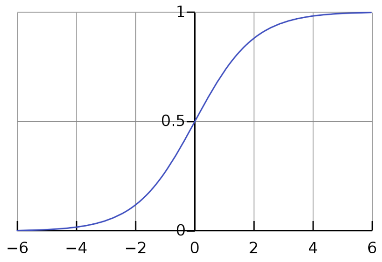

Figure 3: Plot of the logistic function.

- We can interpret $\hat{y}$ as the estimated probability of $y = 1$, and use a loss function that captures the idea that more confidence on a wrong prediction should incur a higher penalty
  - For this, use cross-entropy loss: $\mathcal{L}_{CE}(\boldsymbol{w}) = \dfrac{1}{N}\displaystyle\sum_{i=1}^{N}\left[-y^{(i)}\log\hat{y}^{(i)} - (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$
- Note for multi-class classification, we can have successive classifiers that pick out one class at a time (Lecture 2), or formulate as a multi-output regression problem and use one-hot encodings
- Note the gradient: $\vec{\nabla}_{\boldsymbol{w}}\mathcal{L}_{CE}(\boldsymbol{w}) = \dfrac{1}{N}\displaystyle\sum_{i=1}^{N}(\hat{y}^{(i)} - y^{(i)})\boldsymbol{x}^{(i)}$
  - The gradient is the same as the gradient of the least squares loss; this is not a coincidence

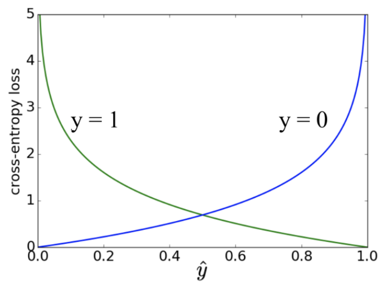

Figure 4: Plot of the cross-entropy loss.