CSCI 5525 Machine Learning Fall 2019

Lecture 3: Linear Regression (Part 2)

Sep 10 2019

Lecturer: Steven Wu *Scribe: Steven Wu*

Revisit SVD and Pseudoinverse

Given any matrix $M \in \mathbb{R}^{m\times n}$, we want to factorize the matrix as $M = USV^\intercal$, where

- r is the rank of the matrix M;
- $U \in \mathbb{R}^{m\times r}$ is orthonormal, that is $U^\intercal U = I_r$;
- $V \in \mathbb{R}^{n\times r}$ is orthonormal, that is $V^\intercal V = I_r$;
- $S \in \mathbb{R}^{r\times r}$ is a diagonal matrix $\text{diag}(s_1, \ldots, s_r)$.

We could also express the factorization as a sum

$$M = \sum_{i=1}^{r} s_i u_i v_i^\intercal$$

where each u_i is a column vector for U and each v_i is a column vector for V. Note that $\{u_i\}$ spans the column space of M and $\{v_i\}$ spans the row space of M. This allows us to define the *(Moore-Penrose) pseudoinverse*

$$M^+ = \sum_{i=1}^{r} \frac{1}{s_i} v_i u_i^\intercal .$$

Basically, we take the inverse of the singular values and reverse the positions of the v_i and u_i within each term. Now let's return to the problem of least squares regression with We can define the design matrix and response vector respectively:

$$A = \begin{bmatrix} \leftarrow x_1^\intercal \rightarrow \\ \vdots \\ \leftarrow x_n^\intercal \rightarrow \end{bmatrix} \qquad \mathbf{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

We learn that $\mathbf{w}^* = A^+ b$ is a solution for the first-order condition $(A^\intercal A)\mathbf{w} = A^\intercal \mathbf{b}$, and thus is the solution to the least square regression problem: $\arg\min_{\mathbf{w}} \|A\mathbf{w} - \mathbf{b}\|_2^2$.

Ridge Regression

Here is another way to handle the case where $A^\intercal A$ is not invertible. Consider the eigendecomposition of the covariance matrix:

$$A^\intercal A = Q \Lambda Q^\intercal$$

where

- $Q \in \mathbb{R}^{d \times d}$ with orthonormal column vectors $\{q_i\}_{i=1}^d$ that are also eigenvectors of $A^\intercal A$.
- $\Lambda \in \mathbb{R}^{d \times d}$ is a diagonal matrix $\mathrm{diag}(\lambda_1, \ldots, \lambda_d)$ with diagonal entries that are eigenvalues of $A^\intercal A$.

Of course, if you don't like matrices, you can also write the factorization as

$$A^\intercal A = \sum_{i=1}^{d} \lambda_i q_i q_i^\intercal$$

Note that the set of eigenvalues may not be distinct. In fact, when the rank $r < d$, $(d - r)$ of the eigenvalues will be 0. Now consider the following matrix: for some $\lambda > 0$,

$$A^\intercal A + \lambda I = A^\intercal A + QIQ^\intercal = Q(\Lambda + \lambda I)Q^\intercal.$$

All of the eigenvalues of this matrix are positive, and the matrix is invertible. We know that if $A^\intercal A$ is invertible, the least squares solution is given by $(A^\intercal A)^{-1}\mathbf{b}$. Now let's replace $A^\intercal A$ by $(A^\intercal A + \lambda I)$, and consider the following solution

$$\hat{\mathbf{w}} = (A^\intercal A + \lambda I)^{-1} A\mathbf{b}.$$

This is in fact the solution to the following regularized ERM problem:

$$\arg\min_{\mathbf{w}} \|A\mathbf{w} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{w}\|_2^2$$

This is also called *ridge regression*, and the solution is always unique even if $n < d$. The *regularization* or *penalty* term $\lambda\|\mathbf{w}\|_2^2$ encourages "shorter" solutions $\mathbf{w}$ with smaller ℓ_2 norm. The paramter λ manages the trade-off between fitting the data to minimize $\hat{\mathcal{R}}$ and shrinking the solution to minimize $\lambda\|\mathbf{w}\|_2^2$.

Lasso regression. Another common regularization is the *Lasso regression* that uses ℓ_1 penalty:

$$\arg\min_{\mathbf{w}} \|A\mathbf{w} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{w}\|_1$$

Lasso encourages sparse solutions, and is commonly used when d is much greater than the number of observations n. However, it does not admit a closed-form solution.

Feature Transformation

We can enrich linear regression models by transforming the features: first transform each feature vector x into $\phi(x)$, and then predict by using linear function over the transformed features, that is $\hat{f}(x) = \mathbf{w}^\intercal \phi(x)$. Consider the following examples of feature transformation:

- for $x \in \mathbb{R}$, $\phi(x) = \ln(1 + x)$
- for $x \in \{0, 1\}^d$, we can apply boolean functions such as
$$\phi(x) = (x_1 \wedge x_2) \vee (x_3 \vee x_4)$$
- for $x \in \mathbb{R}^d$, we can also apply polynomial expansion:
$$\phi(x) = (1, x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, x_1 x_2, \ldots, x_{d-1} x_d)$$
- for $x \in \mathbb{R}$, we can also apply trigonometry expansion:
$$\phi(x) = (1, \sin(x), \cos(x), \sin(2x), \cos(2x), \ldots)$$

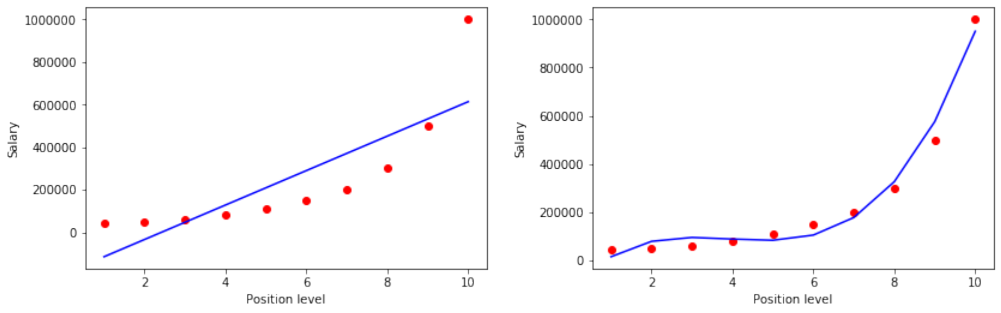

Figure 1: Examples shown in class. Fitting a linear function versus fitting a degree-3 polynomial. (More details here.)

Can we just use complicated linear mapping though? No, we won't gain anything: $\mathbf{w}^\intercal \phi(x)$ is just another linear function of x.

Feature engineering can get messy, and often requires a lot of domain knowledge. For example, we probably should not use polynomial expansion for periodic data.

Hyperparameters, Validation Set, and Test Set

The parameter λ in ridge regression and Lasso regression, and the order of polynomials in polynomial expansion, and also the paramter k in k-nearest neighbor are often called *hyperparamters* for the machine learning algorithms, which requires tuning. How do we optimize these parameters? A standard way is to perform the following three-way data splits:

- Training set: learn the predictor $\hat{f}$ (e.g. weight vector $\mathbf{w}$) by "fitting" this dataset.
- Validation set: a set of examples to tune the hyperparameters. We use the loss on this dataset to find the "best" hyperparameter.
- Test set: we use this data to assess the *risk* of the final model:

  $$\mathcal{R}(f) = \mathop{\mathbf{E}}_{(X,Y)\sim P}[\ell(Y, f(X))]$$

  In the case of squared loss, this is

  $$\mathcal{R}(f) = \mathop{\mathbf{E}}_{(X,Y)\sim P}[(f(X) - Y)^2]$$

  In general, we want to predict well on future instances, so the goal is formulated as finding a predictor $\hat{f}$ that minimizes the risk (instead of empirical risk on the training set).

What if we did not start with a validation set? We can always create a validation set from the training set. One standard method is *cross validation.*

k-fold cross validation We split the training set into k parts or folds of roughly equal size: $F_1, \dots, F_k$. (Typically, $k = 5$ or 10, but it also depends on the size of your dataset.)

1. For $j = 1, \dots, k$:
   - We will train on the union of folds $F_{-j} = \bigcup_{j' \neq j} F_{j'}$ and validate on fold F_j
   - For each value of the tuning parameter $\theta \in \{\theta_1, \dots, \theta_m\}$, train on F_{-j} to obtain predictor $\hat{f}_\theta^{-j}$, and record the loss on the validation set $\hat{\mathcal{R}}_j(\hat{f}_\theta^{-j})$.
2. For each paramter θ, compute the average loss over all folds

   $$\hat{\mathcal{R}}_{\text{CV}}(\theta) = \frac{1}{k} \sum_{j=1}^{k} \hat{\mathcal{R}}_j(\hat{f}_\theta^{-j})$$

   Then we will chose the parameter $\hat{\theta}$ that minimize $\hat{\mathcal{R}}_{\text{CV}}(\theta)$.