CSCI 5525 Machine Learning Fall 2019

Lecture 3: Linear Regression (Part 2)

Feb 3rd 2020

Lecturer: Steven Wu *Scribe: Steven Wu*

Recall the problem of least squares regression with 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 aim to solve the following ERM problem:

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

We learn that $\mathbf{w}^* = A^+\mathbf{b}$ is a solution since it satisfies the first-order condition:

$$(A^\intercal A)\mathbf{w} = A^\intercal \mathbf{b}$$

This is sometimes called the *normal equation*. Note that if A is full rank, then

$$\mathbf{w}^* = A^+\mathbf{b} = (A^\intercal A)^{-1}A^\intercal \mathbf{b}$$

which is the unique minimizer of the least squares objective.

1 A Statistical View

We often study linear regression under the following model assumption: $y_i = \mathbf{w}^\top x_i + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$. In other words, the distribution of y_i given x_i is:

$$\Rightarrow y_i | x_i \sim N(\mathbf{w}^\top x_i, \sigma^2) \Rightarrow P(y_i | x_i, \mathbf{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i^\top \mathbf{w} - y_i)^2}{2\sigma^2}}$$

Consider the maximum likelihood estimation (MLE) procedure that aims to maximize

$$P(\text{observed data} \mid \text{model paramter})$$

In more details:

$$
\begin{aligned}
\mathbf{w} &= \underset{\mathbf{w}}{\operatorname{argmax}}\, P(y_1, x_1, ..., y_n, x_n|\mathbf{w}) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i, x_i|\mathbf{w}) && \text{(Independence)} \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i|x_i, \mathbf{w}) P(x_i|\mathbf{w}) && \text{(Chain rule of probability)} \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i|x_i, \mathbf{w}) P(x_i) && (x_i \text{ is independent of } \mathbf{w}) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i|x_i, \mathbf{w}) && (P(x_i) \text{ does not depend on } \mathbf{w}) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{n} \log\left[P(y_i|x_i, \mathbf{w})\right] && \text{(log is a monotonic function)} \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{n} \left[\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(e^{-\frac{(x_i^\top \mathbf{w} - y_i)^2}{2\sigma^2}}\right)\right] && \text{(Plugging in Gaussian distribution)} \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i^\top \mathbf{w} - y_i)^2 && \text{(First term is a constant, and } \log(e^z) = z\text{)} \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (x_i^\top \mathbf{w} - y_i)^2
\end{aligned}
$$

Now consider a similar maximum a posteriori estimation (MAP) with a prior assumption:

$$P(\mathbf{w}) = \frac{1}{\sqrt{2\pi\tau^2}} e^{-\frac{\mathbf{w}^\top \mathbf{w}}{2\tau^2}}$$

The MAP estimation instead aims to solve

$$P(\text{model parameter} \mid \text{observed data})$$

$$
\begin{aligned}
\mathbf{w} &= \operatorname*{argmax}_{\mathbf{w}} P(\mathbf{w}|y_1, x_1, ..., y_n, x_n) \\
&= \operatorname*{argmax}_{\mathbf{w}} \frac{P(y_1, x_1, ..., y_n, x_n|\mathbf{w})P(\mathbf{w})}{P(y_1, x_1, ..., y_n, x_n)} \\
&= \operatorname*{argmax}_{\mathbf{w}} P(y_1, x_1, ..., y_n, x_n|\mathbf{w})P(\mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}} \left[\prod_{i=1}^{n} P(y_i, x_i|\mathbf{w})\right] P(\mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}} \left[\prod_{i=1}^{n} P(y_i|x_i, \mathbf{w})P(x_i|\mathbf{w})\right] P(\mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}} \left[\prod_{i=1}^{n} P(y_i|x_i, \mathbf{w})P(x_i)\right] P(\mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}} \left[\prod_{i=1}^{n} P(y_i|x_i, \mathbf{w})\right] P(\mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}} \sum_{i=1}^{n} \log P(y_i|x_i, \mathbf{w}) + \log P(\mathbf{w}) \\
&= \operatorname*{argmin}_{\mathbf{w}} \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i^\top \mathbf{w} - y_i)^2 + \frac{1}{2\tau^2} \mathbf{w}^\top \mathbf{w} \\
&= \operatorname*{argmin}_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (x_i^\top \mathbf{w} - y_i)^2 + \lambda ||\mathbf{w}||_2^2 \qquad \lambda = \tfrac{\sigma^2}{n\tau^2}
\end{aligned}
$$

This actually corresponds to a regularized ERM problem called *ridge regression*.

2 Ridge Regression

Now consider following regularized ERM problem called ridge regression:

$$\min_{\mathbf{w}} \|A\mathbf{w} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \tag{1}$$

Now let's replace $A^\intercal A$ by $(A^\intercal A + \lambda I)$ in the ordinary least squares solution and obtain:

$$\hat{\mathbf{w}} = (A^\intercal A + \lambda I)^{-1} A^\intercal \mathbf{b}. \tag{2}$$

Again, by first-order condition, we can show that $\hat{\mathbf{w}}$ is the the solution to (1). Note that the solution is always unique even if A is not full rank (e.g., when $n < d$). 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 $\lambda\|\mathbf{w}\|_2^2$. Ridge regression can also be formulated as a *constrained optimization* problem:

$$\min_{\mathbf{w}} \|A\mathbf{w} - \mathbf{b}\|_2^2 \quad \text{such that} \quad \|\mathbf{w}\| \leq \beta.$$

Why do we care to make the weights $\mathbf{w}$ short or small? Intuively, larger $\mathbf{w}$ corresponds to higher *model complexity*. By bounding the model complexity, we can prevent *overfitting*—that is the model has small training error, but large test error. However, if we bound the norm of $\mathbf{w}$ to aggressively (by setting λ to be very large), then we might run into the problem of *underfitting*—that is the model has large training error and test error.

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.

3 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:

$$\phi(x) = (1, x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, x_1x_2, \ldots, x_{d-1}x_d)$$

- for $x \in \mathbb{R}$, we can also apply trigonometry expansion:

$$\phi(x) = (1, \sin(x), \cos(x), \sin(2x), \cos(2x), \ldots)$$

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, when ϕ is also a linear mapping 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.

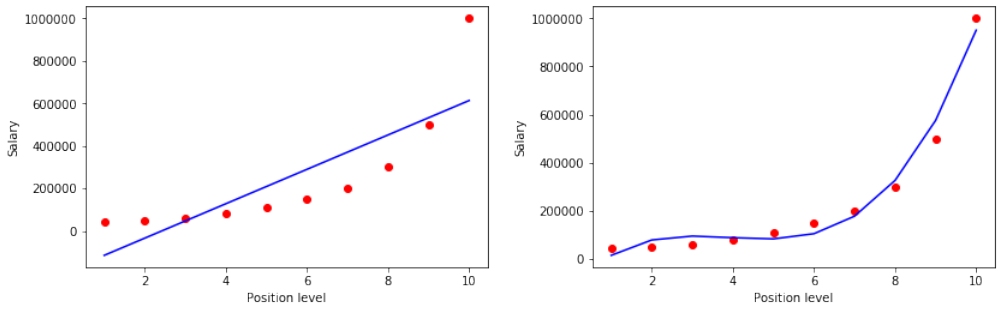

Figure 1: Examples shown in class. Fitting a linear function versus fitting a degree-3 polynomial. (More details here.)

4 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: $F_1, \ldots, F_k$. (Typically, $k = 5$ or 10, but it also depends on the size of your dataset.)

1. For $j = 1, \ldots, k$:
   - 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 $\theta \in \{\theta_1, \ldots, \theta_m\}$, 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 $\hat{\mathcal{R}}_{\text{CV}}(\theta)$.