Week 3: Linear Regression

Instructor: Sergey Levine

1 Recap

In the previous lecture we saw how linear regression can solve the following problem: given a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, learn to predict y from $\mathbf{x}$. In linear regression, we learn a function $f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} = \hat{y}$ or, when using features, $f(\mathbf{x}) = h(\mathbf{x}) \cdot \mathbf{w} = \hat{y}$, where $h(\mathbf{x})$ is the feature or basis function. We saw that linear regression corresponds to maximum likelihood estimation under the model $y \sim \mathcal{D}(\mathbf{w} \cdot \mathbf{x}, \sigma^2)$, and that the optimal parameters can be obtained according to

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y},$$

or, equivalently, according to

$$\hat{\mathbf{w}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{Y}$$

when using features. In today's lecture, we'll analyze overfitting in linear regression, and see how it can be addressed by imposing a prior on $\mathbf{w}$.

2 Overfitting & regularization

Let's imagine that we are trying to learn a 1D function, where $\mathbf{x}$ is one-dimensional and $h(\mathbf{x})$ corresponds to monomials up to some power d:

$$h(\mathbf{x}) = \begin{bmatrix} 1 \\ \mathbf{x} \\ \mathbf{x}^2 \\ \ldots \\ \mathbf{x}^d \end{bmatrix}.$$

If our dataset has size N, then we can always fit the dataset perfectly (with zero error) if $d \geq N - 1$. However, as d increases, a zero-error fit might not actually be desirable, because it might produce an extremely jagged and multimodal function that is unlikely to reflect the actual trends present in the data. More generally, whenever we have a high-dimensional input space or a highly expressive feature set, such that the dimensionality of $\mathbf{w}$ is large, we are liable to overfit. Recall the definition of overfitting: if we find a hypothesis $\mathbf{w}$, but there exists some *other* hypothesis $\mathbf{w}'$ such that its training error is worse but its test error is better, then we are overfitting.

In linear regression, one of the most recognizable symptoms of overfitting is the existence of very large values in $\mathbf{w}$. This would happen, for example, when erroneously fitting a high-degree polynomial with near-perfect accuracy to a noisy dataset. Note that this overfitting is quite similar to something we discussed last week in the context of maximum likelihood estimation: if we flip a coin and "accidentally" observe heads five times in a row, MLE might lead us to conclude the coin would always come up heads. But that is unreasonable.

Question. How can we mitigate overfitting in linear regression?

Answer. Same as last week, we can switch from MLE to a Bayesian approach, and compute the maximum a posteriori (MAP) estimate of the parameters $\mathbf{w}$ instead. This involves imposing a prior on $\mathbf{w}$: our reasonable prior belief about what the parameters *should* be, before we've even seen the data.

A reasonable prior belief is that the parameters $\mathbf{w}$ should be small: this would prevent the sort of huge parameters we might see when fitting a high-degree polynomial with zero error.

Question. What kind of distribution might be suitable for representing the prior on $\mathbf{w}$?

Answer. Since each entry in $\mathbf{w}$ is continuous, real-valued, and unconstrained, the Gaussian distribution is a good choice. In general, we could place a full multivariate Gaussian prior on the entire vector $\mathbf{w}$, but for now let's assume that we'll place an independent Gaussian prior on each dimension of $\mathbf{w}$, with prior mean zero and prior variance σ_0^2, such that

$$\log p(\mathbf{w}) = -\frac{1}{2\sigma_0^2}\sum_{j=1}^{d}\mathbf{w}_j^2 + \text{const.}$$

This means that for each dimension j of $\mathbf{w}$, we have $\mathbf{w}_j \sim \mathcal{N}(0, \sigma_0^2)$.[1] Combining this prior with the likelihood, we get

$$\log p(\mathbf{w}|\mathcal{D}) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \mathbf{x}_i \cdot \mathbf{w})^2 - \frac{1}{2\sigma_0^2}\sum_{j=1}^{d}\mathbf{w}_j^2 + \text{const.}$$

From the form of this likelihood, we can see that the posterior is also Gaussian. Just like before, we can compute the derivative of this quantity and set it to zero to determine the optimal weights:

$$\begin{aligned}\frac{d}{d\mathbf{w}}\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \mathbf{x}_i \cdot \mathbf{w})^2 - \frac{1}{2\sigma_0^2}\sum_{j=1}^{d}\mathbf{w}_j^2\right] &= \frac{1}{\sigma^2}\sum_{i=1}^{N}\mathbf{x}_i(y_i - \mathbf{x}_i \cdot \mathbf{w}) - \frac{1}{\sigma_0^2}\mathbf{w} \\ &= \vec{0}.\end{aligned}$$

[1] This means that $\mathbf{w} \sim \mathcal{N}(\vec{0}, \sigma_0^2\mathbf{I})$: that is, $\mathbf{w}$ is distributed according to a d-dimensional multivariate Gaussian.

Rewriting this in matrix notation like before, we get

$$\begin{aligned}
\frac{1}{\sigma^2}\mathbf{X}^T(\mathbf{Y}-\mathbf{X}\mathbf{w}) - \frac{1}{\sigma_0^2}\mathbf{w} &= \vec{0} \\
\frac{1}{\sigma^2}\mathbf{X}^T\mathbf{Y} - \frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X}\mathbf{w} - \frac{1}{2\sigma_0^2}\mathbf{w} &= \vec{0} \\
\frac{1}{\sigma^2}\mathbf{X}^T\mathbf{Y} &= \frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X}\mathbf{w} + \frac{1}{\sigma_0^2}\mathbf{w} \\
\mathbf{X}^T\mathbf{Y} &= \mathbf{X}^T\mathbf{X}\mathbf{w} + \frac{\sigma^2}{\sigma_0^2}\mathbf{w} \\
\mathbf{X}^T\mathbf{Y} &= (\mathbf{X}^T\mathbf{X} + \frac{\sigma^2}{\sigma_0^2}\mathbf{I})\mathbf{w} \\
(\mathbf{X}^T\mathbf{X} + \frac{\sigma^2}{\sigma_0^2}\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y} &= \mathbf{w}.
\end{aligned}$$

Our solution is therefore given by $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \frac{\sigma^2}{\sigma_0^2}\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$. The only change from standard linear regression is that we've added the term $\frac{\sigma^2}{\sigma_0^2}\mathbf{I}$ to the matrix that we are inverting. In practice, we will often use a single parameter $\lambda = \frac{\sigma^2}{\sigma_0^2}$, so that the solution has the form $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$. We will discuss how to choose the parameter λ in the next section.

This method corresponds to maximum a posteriori (MAP) estimation of the optimal parameters $\mathbf{w}$ under the objective $\log p(\mathbf{w}|\mathcal{D})$, and it is often referred to as *ridge regression*. But we can see here that it is simply the natural consequence of imposing a zero-mean Gaussian prior on the parameters $\mathbf{w}$.

In applying ridge regression in practice, we might also impose a different prior variance $\sigma_{0,j}^2$ on each dimension $\mathbf{w}_j$ of $\mathbf{w}$. For example, if we use features $h(\mathbf{x}_i)$ (recall that the math is exactly the same if we use features!), we might have a constant feature that is equal to 1, called the bias feature. We often do not want to regularize the weight on this feature to allow for whatever bias best fits the data, so we might set its weight to 0 (which corresponds to $\sigma_{0,j}^2 = \infty$). In the case where we use different weights on different features, the solution becomes

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \Lambda)^{-1}\mathbf{X}^T\mathbf{Y},$$

where Λ is a diagonal matrix of weights.

3 LASSO

This is covered in the slides.

4 Choosing the regularization amount

The value λ (or Λ) in ridge regression is a *hyperparameter*: it is not learned by our learning algorithm, but rather must be specified in advance. Hyperparam-

eters can be set by hand using domain knowledge, or they can be optimized by using a hold-out set.

First, let's try to understand how the setting of λ changes the weights that we get. First, as $\lambda \to 0$, ridge regression turns into ordinary linear regression (and our prior approaches the uniform prior). That means that we will fit the training data better (our training error will decrease), but we might experience more overfitting if we have too many parameters and too little data (our test error might increase).

As $\lambda \to \infty$, the $\mathbf{w}_j^2$ terms in the objective dominate, and $\mathbf{w} \to 0$. All of our weights zero out, and we just get a constant prediction of zero (or a constant if we don't regularize the bias term). In this case, we are least likely to see overfitting, but we will also experience very high training and test error, because we're essentially ignoring the input $\mathbf{x}_i$ in making our predictions.

For best results, we need to find the "perfect" value λ that gives the model enough expressive power to get low training and test error, but not so much expressive power as to overfit to the training data. In practice, even guessing a very low value of λ, such as $\lambda = 10^{-4}$, can already help a lot. For example, if $\mathbf{X}^T\mathbf{X}$ is nearly rank-deficient (that is, it has eigenvalues close to zero, making it very hard to invert), adding $\lambda\mathbf{I}$ to it before inversion can make it much easier to invert, making linear regression much more stable. It also quickly removes the *really* pathological solution that have coefficients in the millions or billions. So a quick fix to an ill-conditioned linear regression problem that is easy and often effective is to choose $\lambda = 10^{-4}$.

However, if we want to find a better setting of λ to get the best performance, we need to use our hold-out data. This can be done either manually or automatically. In the manual approach, we simply try a few different settings of λ that we think are reasonable, fit to the training data, and test how well we do on the hold-out data. We then take the best one. The automated approach consists of automating this process. Performance on the hold-out set does not necessarily follow a unimodal curve, but in practice this can be good enough to find a good value, so we could simply choose a lower and upper bound for λ, and then perform a search. We recursively update the lower bounds λ_0 and upper bound λ_1 to find the best value of λ. Letting $\mathcal{E}_{\text{holdout}}(\lambda)$ denote the error on the hold-out set for the optimal solution for hyperparameter λ, the search might look like this:
One good choice for the constant ρ is based on the golden ratio: $\rho = (3-\sqrt{5})/2$.

5 K-fold cross-validation

Using a hold-out set to manually or automatically optimize hyperparameters such as λ is reasonably effective, but it requires us to carve out a large enough hold-out set from our data to provide an accurate estimate of the generalization error of our model. This means we have less data to use for actually fitting the training data. One idea to reduce the size of the hold-out set and still get a good estimate of the generalization error for optimizing hyperparameters is

Algorithm 1 Hyperparameter search

Start with minimum λ_0 and maximum λ_1
while not converged (e.g. $|\mathcal{E}_{\text{holdout}}(\lambda_1) - \mathcal{E}_{\text{holdout}}(\lambda_0)| > \epsilon$) **do**
  $\lambda_0' \leftarrow \lambda_0 + \rho(\lambda_1 - \lambda_0)$
  $\lambda_1' \leftarrow \lambda_1 - \rho(\lambda_1 - \lambda_0)$
  if $\mathcal{E}_{\text{holdout}}(\lambda_0') < \mathcal{E}_{\text{holdout}}(\lambda_1')$ **then**
    $\lambda_0 \leftarrow \lambda_0'$
  else
    $\lambda_1 \leftarrow \lambda_1'$
  end if
end while

to use K-fold cross-validation. In this approach, we partition the dataset into K *folds* of equal size, each one somewhat smaller than an ordinary hold-out set. When we want to evaluate $\mathcal{E}_{\text{holdout}}(\lambda)$, we evaluate it as the average of K *separate* errors:

$$\mathcal{E}_{\text{holdout}}(\lambda) = \frac{1}{K} \sum_{k=1}^{K} \mathcal{E}_{\text{holdout},k}(\lambda),$$

where $\mathcal{E}_{\text{holdout},k}(\lambda)$ is the error we get by testing on the k^{th} fold a model that is trained on *all* of the other folds (with hyperparameter λ). Since we average together hold-out error on many folds to get $\mathcal{E}_{\text{holdout}}(\lambda)$, each fold can be smaller than a standard hold-out set. In fact, if we set $K = N$ (the total size of our dataset), we train N models, and test each of them on just *one* datapoint. This is called *hold one out cross-validation*. It is computationally expensive, but involves the least loss of data to a hold-out set.