Model Complexity and Regularization

Machine Learning

Overview

1. Overfitting.
2. Complexity and capacity of a hypothesis class.
3. Complexity, bias and variance of a hypothesis.
4. Bias-variance decomposition.
5. Regularization.
6. Qualitative Effects of Regularization on Bias-Variance and Overfitting.
7. Example: ℓ_2 (Ridge) Regularization for Linear Regression
8. Example: ℓ_1 (LASSO) Regularization for Linear Regression

Overfitting

Definition (Overfitting)

- **Qualitatively:** When model performs much better on training data than on arbitrary data points drawn from the distribution (simulated by test data).
- **Quantitatively:** When "overfitting gap" $L_\mu(h) - L_{s_{\text{train}}}(h)$ is large.

Example (Memorizing)

A hypothesis h which "memorizes" training data and outputs ∞ elsewhere:

$$h(x) = \begin{cases} y & (x, y) \in s_{\text{train}} \\ \infty & x \notin s_{\text{train};X} \end{cases}$$

Gets all training data exactly right but everything else infinitely wrong.
Hence $L_{s_{\text{train}}}(h)$ minimized, $L_\mu(h)$ maximized, "overfitting gap" maximized.

Hypothesis Class Quality

- Say we have a hypothesis class $\mathscr{H}$ (given by a particular ML algorithm).
- We want to quantify the suitability of $\mathscr{H}$ for our problem.
- **Occam's Razor:** All else being equal, the lowest-complexity solution is best.
- But there are trivially low-complexity hypothesis classes, which suck at fitting the data.
- For example, $\mathscr{H} = \{x \mapsto 0\}$ is bad at fitting nonzero outputs.
- So we want to quantify how good $\mathscr{H}$ can be at fitting the data.
- Two related notions: **complexity** and **capacity**.

Hypothesis Class Complexity

- Hard to discuss non-parametric hypothesis class complexity, as there is less structure.
- It is safe to say hypothesis classes with very diverse hypotheses are high-complexity.
- Correspondingly, hypothesis classes with a lot of structure are low-complexity.
- For parametric hypothesis classes of the form

  $$\mathscr{H} = \{h_\theta : \theta \in \Theta\}$$

  We can be significantly more quantitative/clear.
- One idea is to count the number of independent parameters (“degrees of freedom”) in Θ:

  $$\mathrm{DF}(\mathscr{H}) = \dim(\Theta)\,.$$

  This works pretty well and for lots of algorithms it is obvious or well-known.

Hypothesis Class Complexity

Example (Linear Regression Hypothesis Classes)

The "naive" linear regression hypothesis class looks like

$$\mathcal{H}_{\text{dumb}} = \left\{ x \mapsto \theta^* x \colon \theta \in \mathbb{R}^{d \times k} \right\}.$$

In this case $\mathrm{DF}(\mathcal{H}_{\text{dumb}}) = dk$.
That's not quite the end of the story. Remembering the least squares solution,

$$\widehat{\theta} = \left(X_{\text{train}}^* X_{\text{train}}\right)^{-1} X_{\text{train}} Y_{\text{train}}$$

so the hypothesis class looks more like

$$\mathcal{H}_{\text{smart}} = \left\{ x \mapsto \left(\left(X_{\text{train}}^* X_{\text{train}}\right)^{-1} X_{\text{train}}^* Y_{\text{train}} \right)^* x \colon \begin{array}{l} X_{\text{train}} \in \mathbb{R}^{n \times d} \\ Y_{\text{train}} \in \mathbb{R}^{n \times k} \end{array} \quad \mathrm{rank}(X_{\text{train}}) = d \quad . \right\}$$

This has $\mathrm{DF}(\mathcal{H}_{\text{smart}}) \ll dk$, so linear regression isn't too complex.

Hypothesis Class Capacity

- We don't want our hypothesis class to be too complicated.
- But we also want the hypotheses in the class to have some predictive power.
- A hypothesis class with a low-dimensional (or even finite) parameter space Θ has low complexity, but is probably not very good at fitting the data.
- Measures of capacity of a hypothesis class $\mathcal{H}$:
  - VC dimension for binary classification ($\mathcal{Y} = \{1, 2\}$): size of largest set of points s such that, for every assignment $f : \mathcal{X} \to \mathcal{Y}$, there is $h \in \mathcal{H}$ such that $f|_s = h|_s$.
  - Rademacher complexity with respect to an integer n and distribution ν over $\mathcal{Z}$:

$$\text{Rad}_{\nu,n}(\mathcal{H}) = \frac{1}{n} \mathop{\mathbb{E}}_{\substack{s\sim\nu^n \\ \sigma\sim\text{Uni}\left(\{\pm1\}^n\right)}} \left[\sup_{h\in\mathcal{H}} \sum_{i=1}^{n} \sigma_i \ell_h(z_i) \right]$$

  Possible to get bounds on Rademacher complexity analytically, even for complex models such as neural networks.
- In general, capacity is hard to compute, but easy to give a qualitative/asymptotic estimate.

Model Complexity/Capacity Tradeoff

- Highly complex hypothesis classes can model arbitrarily complex functions.
- Thus, we expect complexity and capacity to be very correlated.
- We would like to maximize capacity and minimize complexity, which is a difficult problem.
- Each algorithm has a different complexity-capacity balance, so we pick one that works for our use-cases.

Complexity of a Hypothesis

- We can also talk about complexity of a hypothesis.
- (Capacity isn't a very useful notion since a hypothesis can model exactly one function.)
- We would like low-complexity hypotheses; those that are smooth and don't have very high curvature at every point. Possible complexity functional:

$$R(h) = \mathop{\mathbb{E}}_{x}\left[\det(\text{Hessian}(h(x)))^2\right].$$

- Easier/more concrete for parametric hypotheses.
- Larger coefficients $\implies$ probably overly complex hypothesis. (Occam's Razor again.)
- Possible complexity functionals:

$$R(h_\theta) = \lambda \begin{cases} \|\theta\|_1 = \text{sum of absolute values of entries of } \theta \\ \|\theta\|_2^2 = \text{sum of squares of entries of } \theta \\ \|\theta\|_\infty = |\text{largest entry of } \theta| \\ \|\theta\|_0 = \text{\# of nonzero entries of } \theta \end{cases}$$

Bias and Variance of an Algorithm

- Some analogous properties of “capacity” and “complexity” for a hypothesis class are **bias** and **variance** for an algorithm.
- **Bias**: measures the difference between the “systematic” (non-random) parts of the output, and the hypothesis given by the algorithm.
- **Variance**: measures the variability of the hypothesis given by the algorithm.
- How to measure distance? Loss function!

Bias and Variance of an Algorithm

- Let the algorithm's output be $h = A(s_{\text{train}})$.
- The *optimal prediction* is that which minimizes the loss pointwise:

$$y_*(x) = \operatorname*{argmin}_{u \in \mathcal{Y}} \mathbb{E}_{y}[\ell(y, u) \mid x] \,.$$

Note: This is the rule we would use if we knew $\mu_{Y \mid X}$, but we don't.

- The *main prediction* is that which minimizes the loss on the dataset:

$$\widehat{y}(x) = \operatorname*{argmin}_{u \in \mathcal{Y}} \mathbb{E}_{s_{\text{train}}} [\ell(h(x), u) \mid x] \,.$$

  - It represents the "central tendency" of A around y.

Bias and Variance of an Algorithm

- The **bias** of an algorithm A on an example x is $\mathrm{Bias}(A; x) = \ell(\widehat{y}(x), y_*(x))$.
  - It represents the systemic loss incurred by A.
- The **variance** of an algorithm A on an example x is $\mathrm{Var}(A; x) = \mathbb{E}_{S_{\text{train}}}[\ell(h(x), \widehat{y}(x))]$.
  - It represents the fluctuations of A due to the varying samples.
- The **noise** of an algorithm A on an example x is $\mathrm{noise}(A; x) = \mathbb{E}_y[\ell(y, \widehat{y}(x)) \mid x]$.
  - It represents the irreducible error regardless of the choice of A and sample.
- We can also talk about **average bias/variance/noise**:

$$\mathrm{Bias}(A) = \mathop{\mathbb{E}}_x\Big[\mathrm{Bias}(A; x)\Big] \qquad \mathrm{Var}(A) = \mathop{\mathbb{E}}_x\Big[\mathrm{Var}(A; x)\Big] \qquad \mathrm{noise}(A) = \mathop{\mathbb{E}}_x\Big[\mathrm{noise}(A; x)\Big]$$

Bias and Variance

Example (Squared-Error Loss)

Suppose if ℓ is squared error loss (so $\ell(y, y') = \|y - y'\|_2^2$), then

$$y_*(x) = \operatorname*{argmin}_{u \in \mathcal{Y}} \mathop{\mathbb{E}}_{y}\left[\|y - u\|_2^2 \,\middle|\, x \right] = \mathop{\mathbb{E}}_{y}[y \mid x]$$

$$\widehat{y}(x) = \operatorname*{argmin}_{u \in \mathcal{Y}} \mathop{\mathbb{E}}_{S_{\text{train}}}\left[\|h(x) - u\|_2^2 \,\middle|\, x \right] = \mathop{\mathbb{E}}_{S_{\text{train}}}[h(x) \mid x] \, .$$

Bias-Variance Decomposition

We can decompose loss at a point:

$$\mathrm{Loss}(A;x) = \mathop{\mathbb{E}}_{s_{\text{train}}}\left[\mathop{\mathbb{E}}_{y}[\ell(y,h(x)) \mid x]\right]$$

into functions of $\mathrm{Bias}(A;x)$, $\mathrm{Var}(A;x)$, and $\mathrm{noise}(A;x)$.

Example (Squared-Error Loss)

$$\mathrm{Loss}(A;x) = \mathrm{Bias}(A;x) + \mathrm{Var}(A;x) + \mathrm{noise}(A;x)\,.$$

Want low-bias and low-variance, but they are correlated, so we have to make a tradeoff. Different algorithms have different bias-variance tradeoffs.

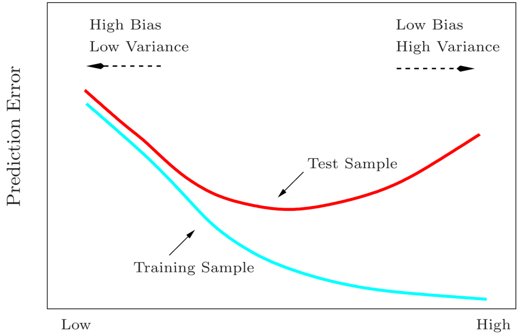
Regularization

- Idea: to get a good tradeoff, we do the following three step process:
  - Pick an algorithm family with low bias.
  - Change the loss function to penalize high-variance algorithms and complex hypotheses.
  - Pick the best algorithm and run it on the training set to get a good hypothesis.
- Regularized loss function:

$$\ell_h^{\text{reg}}(z) = \ell_h(z) + R(h).$$

Regularization, Bias, Variance, and Overfitting

- Our hypotheses are lower complexity, because we are more willing to sacrifice loss on the training set to get reduced hypothesis complexity.
- This means that regularized algorithms have:
  - Higher bias.
  - Lower variance.
  - Less overfitting.



Example: ℓ^2-Regularized Linear Regression

Example (Ridge Regression)

Let's suppose we have the linear regression algorithm and $R(h_\theta) = \lambda \|\theta\|_2^2$, for $\lambda > 0$. Then we want to minimize in θ the expression

$$L^{\text{reg}}_{S_{\text{train}}}(h_\theta) = \|X_{\text{train}}\theta - Y_{\text{train}}\|_2^2 + \lambda \|\theta\|_2^2$$

which has solution

$$\widehat{\theta}_\lambda = \left(X^*_{\text{train}} X_{\text{train}} + \lambda I_d\right)^{-1} X^*_{\text{train}} Y_{\text{train}}.$$

Pro: $X^*_{\text{train}} X_{\text{train}} + \lambda I_d$ is *always* invertible, and the problem is easier to optimize.
Con: we don't exactly have a line of best fit (or generalization) anymore.

Example: ℓ^1-Regularized Linear Regression

Example (LASSO Regression)

Let's suppose we have the linear regression algorithm and $R(h_\theta) = \lambda \|\theta\|_1$, for $\lambda > 0$. Then we want to minimize in θ the expression

$$L^{\text{reg}}_{s_{\text{train}}}(h_\theta) = \|X_{\text{train}}\theta - Y_{\text{train}}\|_2^2 + \lambda \|\theta\|_1$$

which has no easy closed-form solution.
Pro: ℓ^1-regularization promotes *sparsity*, so # of nonzero entries in θ will be high (depending on λ). This is a form of *feature selection*: features with coefficient 0 in $\widehat{\theta}_\lambda$ are considered insignificant and can be tossed out in the final model.
Con: we don't exactly have a line of best fit (or generalization) anymore.