05-Linear Regression (Regularization)

Regularization

Regularization to a machine learning system is to restrict the model capacity, so that it will have a lower variance and usually a higher bias.

Regularization usually hurts training performance. It may or may not improve validation/test performance.

ℓ_2 Regularization

Suppose again $h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$, where $\boldsymbol{w}$ is the parameters
This time, $\boldsymbol{x}$ may involve more input features or some transformations of input. We are not 100% sure if a feature x_i is useful or how useful the feature is. As usual, $\boldsymbol{x}$ here includes an augmented feature $x_0 = 1$. We now state a preference such that $\|\boldsymbol{w}\|_2^2 < C$ for some constant C.

That is to say, the hypothesis class becomes $\mathcal{H} = \left\{ h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} \;\middle|\; \boldsymbol{w} \in \mathbb{R}^{d+1}, \|\boldsymbol{w}\|_2^2 < C \right\}$

With MSE, the training objective is

$$\underset{\boldsymbol{w}}{\text{minimize}} \; \|\boldsymbol{t} - \boldsymbol{X}\boldsymbol{w}\|_2^2$$
$$\text{subject to} \; \|\boldsymbol{w}\|_2^2 - C \leq 0 \qquad \text{with} \; C > 0$$

Under Slater's condition, this optimization is equivalent to

$$\underset{\boldsymbol{w}}{\text{minimize}} \; \|\boldsymbol{t} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2$$

for some $\lambda \geq 0$. [HW: closed form solution]

Slater's condition (convex + strictly feasible => strong duality)
For a convex optimization problem

minimize $f_0(\boldsymbol{x})$
subject to $f_i(\boldsymbol{x}) \leq 0$, for $i = 1, \ldots, m$,
$h_i(x) = 0$, for $i = 1, \ldots, n$
where f's are convex functions and h's are affine functions.

The primal problem $p^* = \min_{\substack{x \\ f_i(x) \le 0,\ i=1,\dots,m \\ h_i(x)=0,\ i=1,\dots,n}} f_0(x)$

and the dual problem is

$$d^* = \max_{\lambda_i \ge 0,\ i=1,\dots,m} \min_{x} f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{n} \nu_i h_i(x)$$

Slater's condition says

If the convex problem is strictly feasible, i.e., $\exists x,\ g_i(x) < 0,\ i=1,\dots,m,\ h_i(x) = 0,\ i=1,\dots,n$

Then, strong duality holds, i.e., $p^* = d^*$.

See Secs 5.2.3 and 5.3.2 in [BV04] for details and the proof.

ℓ_1 Regularization

$$\mathcal{H} = \{ h(x) = w^T x \mid w \in \mathbb{R}^{d+1},\ \|w\|_1 < C \}$$

Note: ℓ_p-norm of x $\quad \|x\|_p = (x_1^p + x_2^p + \cdots + x_d^p)^{\frac{1}{p}}$

Thus, ℓ_1 regularized least square regression is

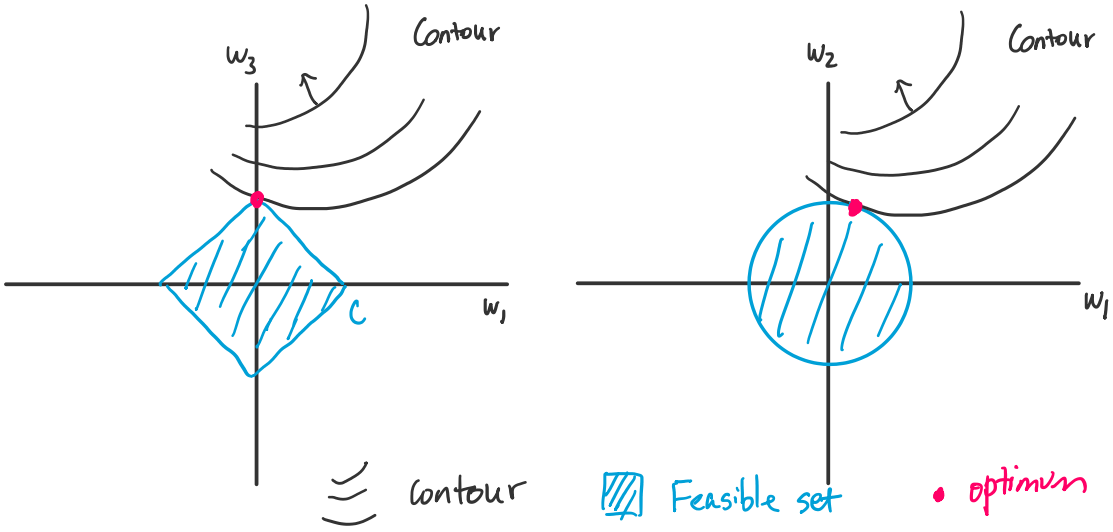
$$\underset{w}{\text{minimize}} \ \|t - Xw\|_2^2$$

$$\text{subject to} \ \|w\|_1 - C \le 0$$

which is equivalent to

$$\underset{w}{\text{minimize}} \ \|t - Xw\|_2^2 + \lambda \|w\|_1$$

for some $\lambda \ge 0$



Probabilistic interpretation for ℓ_1 and ℓ_2 regularization

The above analysis of bias-variance tradeoff (from a frequentist's point of view) has some bizarre assumptions:

- Dataset is given, but is a random variable
- Parameters are unknown, but are assumed to be some constants

More reasonable assumptions seem to be

- Dataset is given, so we don't care its expectation
- Parameters are unknown, and should be modeled as RVs.

Can we even model parameters as random variables?

- Frequentist: No, because parameters are NOT generated from a repeatable trial. Similar nonsense statements from a frequentist's perspective:
  - The probability of the speed of light in (-2.9×10^8, 3.1×10^8) m/s
  - The probability of being rainy tomorrow
- Bayesian: Yes, everything unknown is a random variable. The probability in the above sentences should be interpreted as subjective belief. Then why probability axioms?
  - Non-probability preferences can be converted to probabilities.
  - Convenient as we can re-use all results in the probability theory, especially for some subjective belief that is really close to frequentist's definition, for example, estimated probability.
  - The only fundamentally correct decision theory.

  Further reading on the philosophy of Bayesian analysis:

  Berger, *Statistical Decision Theory and Bayesian Analysis*. https://link.springer.com/book/10.1007/978-1-4757-4286-2

Stand point of this course:

- We acknowledge all theoretical results under respective assumptions.
- We agree that conditional Bayesian analysis is the only fundamentally correct decision principle, but we also see its intractability (unable to compute things exactly in a reasonable time).
- We appreciate both frequentist's analysis and Bayesian analysis, because both give us intuition.
- We treat these approaches pragmatically and do not enter endless debate.
- Science is subjective.

Max *a posteriori* (MAP) estimation

- We define the prior distribution on parameters $p(\boldsymbol{\theta})$

- We define the prior distribution on parameters $p(\boldsymbol{\theta})$
  - The terminology "prior" means our belief of parameters without seeing anything relevant to this task.
    - Prior can be non-informative, or based on previous experience
- We compute the likelihood of data as in frequentist's analysis, $p(\mathcal{D}|\boldsymbol{\theta})$
  - Here, we treat $\boldsymbol{\theta}$ as a random variable, so we use the conditional bar notation. Computationally, it is the same as $p(\mathcal{D};\boldsymbol{\theta})$
- We compute the posterior distribution on parameters

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\boldsymbol{\theta})\cdot p(\mathcal{D}|\boldsymbol{\theta})}{p(\mathcal{D})} \quad \text{[Bayes' rule]}$$

$$\propto_{\boldsymbol{\theta}} p(\boldsymbol{\theta})\cdot p(\mathcal{D}|\boldsymbol{\theta}) \quad \text{[}p(\mathcal{D})\text{ is a constant when comparing different }\boldsymbol{\theta}\text{]}$$

  - Posterior indicates our belief on parameters after we see data $\mathcal{D}$
- Maximum *a posteriori* estimation

$$\hat{\boldsymbol{\theta}}_{MAP} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D}) = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta})\,p(\mathcal{D}|\boldsymbol{\theta})$$

Cf. Maximum likelihood estimation (MLE)

$$\hat{\boldsymbol{\theta}}_{MLE} = \arg\max_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta})$$

MAP estimation for linear regression with independent, zero-mean Gaussian prior

- Suppose $\boldsymbol{w} = (w_0, w_1, \cdots, w_d)^T \in \mathbb{R}^{d+1}$
- Independent, zero-mean Gaussian prior

$$\forall i = 0, 1, \cdots, d, \quad w_i \overset{iid}{\sim} p(w) = \mathcal{N}(0, \sigma^2)$$

I.e., $$p(\boldsymbol{w}) = \prod_{i=0}^{d} p(w_i) \propto \prod_{i=0}^{d} \exp\left(-\frac{w_i^2}{2\sigma^2}\right) = \exp\left\{-\frac{1}{2\sigma^2}\sum_{i=0}^{d} w_i^2\right\}$$

- MAP estimation

$$\hat{\boldsymbol{w}}_{MAP} = \arg\max_{\boldsymbol{w}} p(\boldsymbol{w})\,p(\mathcal{D}|\boldsymbol{w})$$

$$= \arg\max_{\boldsymbol{w}} \left[\log p(\mathcal{D}|\boldsymbol{w}) + \log p(\boldsymbol{w})\right]$$

$$= \arg\max_{\boldsymbol{w}} \left[\underbrace{-\frac{1}{2\sigma_2^2}\sum_{m=1}^{M}\left(y - \boldsymbol{w}^T\boldsymbol{x}^{(m)}\right)^2}_{\text{likelihood + const.}} \underbrace{-\frac{1}{2\sigma_w^2}\sum_{i=1}^{d} w_i^2}_{\text{prior + const.}} + \text{const.}\right]$$

$$= \arg\min_{\boldsymbol{w}} \left[\underbrace{\frac{1}{2M}\sum_{i=1}^{M}\left(y^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)^2}_{\text{MSE}} + \underbrace{\lambda\sum_{i=1}^{d} w_i^2}_{\ell_2\text{-penalty}}\right]$$

for some $\lambda > 0$

Training-validation-test framework

- Machine learning is all about tradeoff.
- In many cases, a machine learning system involves some settings that are not to be optimized during training. They are called hyperparameters.

- We need a systematic way of tuning hyperparameters.
- However, we **CANNOT** use the test set to tune the hyperparameters.
  - Test set should always reserved for performance report. Thus, any tuning should be conducted **without** seeing the test set.
  - Analogous examples of wrong protocols of hyperparameter tuning
    1. We have a stochastic hypothesis function $h(x) \equiv \epsilon$, where $\epsilon \sim N(0, \sigma^2)$. We now introduce a dummy hyperparameter α into the hypothesis function $h(x, \alpha) \equiv \epsilon$, where α does not play a role. If I tune α on the test set, I can improve the MSE test loss (measure of success) arbitrarily low: I tune α many times, and the loss of some α will be less than others because of randomness, and eventually, I will be lucky enough with some α where the loss is arbitrarily close to zero.
    2. In a daily life example, say, a student was enrolled in CMPUT466/566. The student would like to tune his/her learning strategy (e.g., how much time to spend on the course) by the exam score. The student decided to try 10 minutes per week, but then did not perform well in the exam. Unfortunately, the student would not have another chance to improve his/her learning strategy.
- **A correct protocol of tuning hyperparameters**
  - Split the previously called "training data" (which now we call labeled data) into two parts: One is still called the training set, and the other is called the validation set.
  - The machine learning system is trained the model on the "training data" but is then tested on the "validation set" to have an estimate on the test data. This procedure is known as validation and cross-validation, and can be done by any protocol as long as the test data is not seen.
  - Eventually, test the machine learning model on the test data.
- **Hold-out validation**
  - Split $\mathcal{D}_{labeled} = \mathcal{D}_{train} \cup \mathcal{D}_{test}$ by, say, 3:1
  - **Algorithm:** Hold-out validation

    For different hyperparameters α:
    - Train machine learning model on $\mathcal{D}_{train}$

      $$\boldsymbol{\theta}_\alpha = \operatorname{argmin}_{\boldsymbol{\theta}} J_\alpha(\boldsymbol{\theta}; \mathcal{D}_{train})$$
    - Validate the performance of $\boldsymbol{\theta}_\alpha$ by the measure of success

- Select the model $\boldsymbol{\theta}_{\alpha^*}$ the yields the best measure of success
- Report the measure of success of $\boldsymbol{\theta}_{\alpha^*}$ on the test set

- **k-fold validation**
  - Drawback of hold-out validation: The validation set take samples out of the training set.
  - Want:
    - Large training set: better model
    - Large validation set: more stable validation performance
  - Idea: Repeat the training-validation procedure k times, and take the average as the validation performance.
  - Note: The test set can only be used once. Repeating train-val-test k times and report the average test performance is a WRONG protocol.
  - **Algorithm**: k-hold validation

Split $\mathcal{D}_{\text{label}} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_k$
For different hyperparameters α:
  For $\kappa = 1 \cdots k$:
    Train on $\mathcal{D}_{\text{label}} \setminus \mathcal{D}_\kappa$: $\boldsymbol{\theta}_{\alpha,\kappa} = \arg\min_{\boldsymbol{\theta}} J_{\alpha,\kappa}(\boldsymbol{\theta})$
    Validate on $\mathcal{D}_\kappa$ by the measure of success $S_\alpha(\mathcal{D}_\kappa)$
  Average the measure of success $S_\alpha = \frac{1}{k}\sum_{\kappa=1}^{k} S_\alpha(\mathcal{D}_\kappa)$
Pick α_* that yields the best S_α

  - Hyperparameter selection is done, but how can we have the model?
    - Convex problems: Re-train the model with the entire $\mathcal{D}_{\text{label}}$ because we oftentimes approximately find the (global) optimum
    - Non-convex problems: k-hold validation is discouraged by the instructor. In non-convex optimization, the validation set not only validates the hyperparameters, but also validates each run (trajectory) of training.

- **A few algorithms for tuning "different hyperparameters $\boldsymbol{\alpha}$"**
  - Suppose we have N hyperparameters $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_N)$. The number of combinations of different $\boldsymbol{\alpha}$ values grows exponentially with N.
  - **Coordinate ascent/descent**
    - Loop until satisfied
      - Pick i in 1, …, N
      - Fix $\boldsymbol{\alpha}_{-i}$, i.e., all $\alpha_1, \alpha_2, \cdots, \alpha_N$ except α_i
      - Tune α_i only to maximize the measure of success on $\mathcal{D}_{\text{val}}$

- **Local search (hill climbing)**
  - Loop until satisfied
    - Tune $\boldsymbol{\alpha}$ by changing the value in a few coordinates, e.g., α_i, α_j
    - Pick the best hyperparameter around $\boldsymbol{\alpha}$
- Note: The first approach picks the best α_i by searching on a line, whereas the second approach only looks around locally, giving an opportunity to coordinately tune a few hyperparameters.
- In general, hyperparameter tuning is an art, but may affect performance to a large extent.

- Tips for data split in validation
  - In a benchmark dataset contains train-val-test split, use it.
  - In a benchmark dataset contains train-test split, reserve the test set for reporting performance, use the "train" set for training and validation
  - If you collected the dataset, split train-val-test by yourself and make it available with the dataset.
  - If you would like to split train-val or train-val-test by yourself, here are some empirical settings:
    - If the dataset is super large (e.g., 1M samples)
      - Train = Most, Val = ~10K, Test = ~10K
      - Val can be smaller if the prediction is slow
    - If the dataset is relatively large (e.g., 10K samples)
      - Train : Val : Test = 3:1:1
    - If the dataset is small (e.g., 1K samples)
      - Perform k-fold validation for convex problems
      - If the problem is non-convex and we only have very few samples
        - Consider using a convex model, or
        - Collect more data

More Tips for Debugging Machine Learning Systems

- A machine learning program may exhibit undesired behaviors. This may involve many aspects:
  - Bugs in implementation/programming
  - Bad hyperparameters
  - Wrong machine learning model
  - Data preparation is wrong
  - The task is not learnable (e.g., predict the house price from nothing)

    - The task is not learnable (e.g., predict the house price from nothing)
- Therefore, we need a systematic approach for debugging a machine learning system, although such debugging still remains an art (instead of science or engineering).
- Before getting started, we shall keep in mind that, roughly speaking,

  $$\text{Machine Learning} = \text{Optimizaiton} + \text{Generalization}$$

  Thus, we always ask two questions "Is my model optimizing?" and "Is my model generalizing" in addition to "Is my coding correct?"
- It is suggested to always
    - Monitor the training loss, and
    - Validate performance by measure of success on the validation set

- Below are some further tips