EE 194 Lecture 3: Review Lecture 3

Hilary Huang, Mesut Yang, Ryan Cheng

September 2018

1 Regularization

Recall that classic machine learning is about interpolation. Given data points (x_i, y_i) , our goal is to find $\hat{y}$ for x_{test} by fitting a function to the data we have.

1.1 Model order

In order to prevent overfitting, we need to put constraints on our model to regularize it. One way is to control the number of parameters used in the model, or "model order". For example, consider fitting a polynomial to a bunch of data points. The higher the order of the polynomial is, the better the polynomial is able to fit the points. However, this makes the model prone to consider noise as data, and thus have a high variance.

1.2 Prior

A second way of regularization is to enforce a prior on our data distribution. Consider the following:

$$\begin{gathered} y = wx + N \\ w \sim \mathcal{N}(\mu, \sigma^2) \\ N \sim \mathcal{N}(0, 1) \end{gathered}$$

And we do the Maximum a posteriori estimation (MAP) for w:

$$\begin{aligned} \arg\max_w f(w|\vec{x}, \vec{y}) &= \arg\max_w \frac{f(w) \cdot f(\vec{x}, \vec{y}|w)}{f(\vec{x}, \vec{y})} \\ &= \arg\max_w f(w) \cdot f(\vec{x}, \vec{y}|w) \end{aligned}$$

Substituting in a gaussian prior gives:

$$f(w|\vec{x}, \vec{y}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(w-\mu)^2}{2\sigma^2}} \cdot \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-\frac{(y_i - w \cdot x_i)^2}{2}}$$

Taking negative log converts argmax to argmin:

$$\begin{aligned}
\arg\max_w f(w|\vec{x},\vec{y}) &= \arg\min_w -\log f(w|\vec{x},\vec{y}) \\
&= \arg\min_w -\log(\frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{-(w-\mu)^2}{2\sigma^2}} \cdot \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}}e^{-\frac{(y_i-w\cdot x_i)^2}{2}}) \\
&= \arg\min_w -\log(\frac{1}{\sqrt{2\pi\sigma^2}}) + \frac{(w-\mu)^2}{2\sigma^2} - \sum_{i=1}^{n}\log(e^{-\frac{(y_i-w\cdot x_i)^2}{2}}) - \log(\frac{1}{\sqrt{2\pi}}) \\
&= \arg\min_w \frac{(w-\mu)^2}{2\sigma^2} + \frac{1}{2}\sum_{i}^{n}(y_i - wx_i)^2
\end{aligned}$$

Therefore,

$$\hat{w} = \arg\min_w \frac{(w-\mu)^2}{2\sigma^2} + \frac{1}{2}\sum_{i}^{n}(y_i - wx_i)^2$$

We see the second term is just normal squared error but the first term comes from the Gaussian prior.
Take derivative and solve, we get:

$$\hat{w} = \frac{\sum_{i=1}^{n} x_i y_i + \frac{\mu}{\sigma^2}}{\sum_{i=1}^{n} x_i^2 + \frac{1}{\sigma^2}}$$

We can take a closer look at the role of σ^2, the variance of our prior, in the optimal $\hat{w}$:

- $\sigma \to 0$: This corresponds to extreme confidence in the prior. In this case, the terms with σ as denominator ($\frac{\mu}{\sigma^2}$ and $\frac{1}{\sigma^2}$) completely dominate the terms containing actual data ($\sum_{i=1}^{n} x_i y_i$ and $\sum_{i=1}^{n} x_i^2$). Therefore, the optimal $\hat{w}$ would just be μ,the mean of prior

- $\sigma \to \infty$: This is opposite to the previous case, and corresponds to no confidence in our prior. Here, all the term containing σ vanishes, and we end up with an optimal guess as if we have no prior.

Recall that $Y_i = W_{true}X_i + N_i$. When we substitute in y_i, we can think of $\hat{w}$ as a random variable because $N_i \sim \mathcal{N}(0,\,1)$.

$$\begin{aligned}
\hat{w} &= \frac{\frac{\mu}{\sigma^2} + \sum_{i=1}^{n} x_i^2 W_{true}}{\sum_{i=1}^{n} x_i^2 + \frac{1}{\sigma^2}} + \frac{\sum_{i=1}^{n} x_i N_i}{\sum_{i=1}^{n} x_i^2 + \frac{1}{\sigma^2}} \\
&= w_{true} + \frac{\mu - w_{true}}{\sigma^2\sum_{i=1}^{n} x_i^2 + 1} + \frac{\sum_{i=1}^{n} x_i N_i}{\sum_{i=1}^{n} x_i^2 + \frac{1}{\sigma^2}} \\
&= w_{true} + \text{approximation error} + \text{estimation error}
\end{aligned}$$

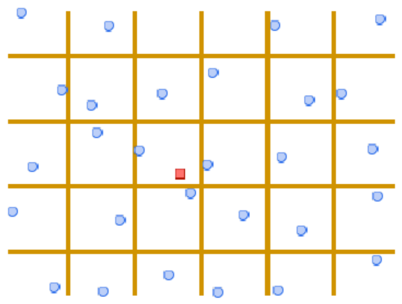
The first term is fixed and represents the approximation error, whereas the second term varies with N and represents the estimation error.

Priors are like penalty functions in optimization, but we can also consider them as fake data points X_{fake}, Y_{fake}. Verify that having a prior is similar to adding an extra data point, $(\frac{1}{\sigma^2}\ \frac{\mu}{\sigma^2})$ to the original $\hat{w} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}$.

2 How can we optimize?

2.1 Brute Force

1. Try all possibilities (if discrete)
2. Try all possibilities on a grid[1].



[2]

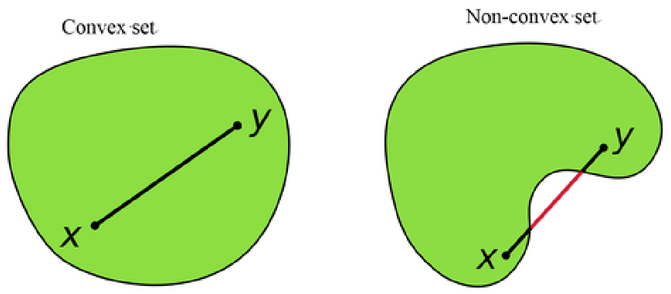
3. Try all possibilities from a random set[3]

2.2 Analytic Solution (With the help of vector calculus)

Special case: **Convex Problem**
For points $\vec{x_1}$ and $\vec{x_2}$ in a **convex set** A,

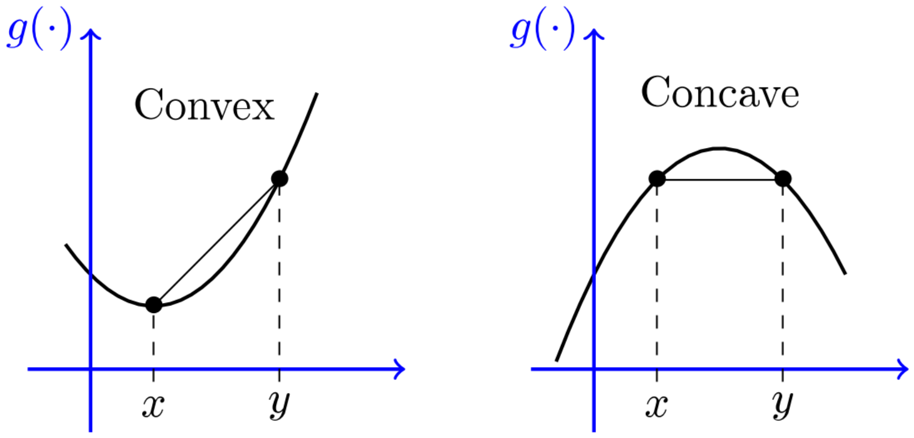
$$\lambda\vec{x_1} + (1-\lambda)\vec{x_2} \in A$$
$$\forall\lambda \in [0,1]$$

[1]Ke Li and Jitendra Malik. Fast k-Nearest Neighbour Search via Prioritized DCI. *In Proceedings of the 34th International Conference on Machine Learning*, pages 2081–2090, 2017.
[2]CS189 SP18 HW13
[3]James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research 13 (2012)*, pages 281-305



[4]

Convex and concave functions have nice properties: it is easy to find minimum/maximum of them



2.3 Gradient Descent

The main idea is to move the weight w closer to its optimum value over time.

1. w_0 is initialized to be a random vector in R^d, where d is the dimensionality of the parameter vector w.
2. For $t = 1, 2, 3 \ldots$:

$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

3. if sufficient progress is not made, terminate and return w_t. Otherwise, repeat step 2

A few Notes on Gradient Descent :

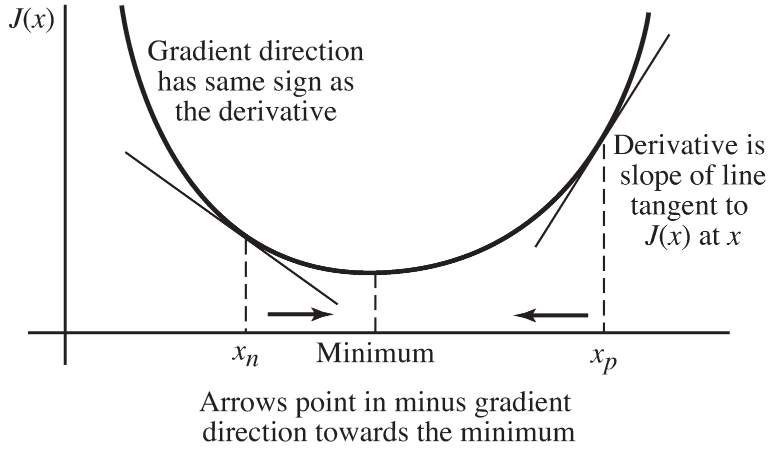
- η is the step size or learning rate. It controls the amount we move in each iteration.

[4]http://www.gtmath.com/2016/03/convexity-and-jensens-inequality.html

  - η too big: w will oscillate and diverge
  - η too small: the algorithm will take too long to converge

  Sometimes, we may decrease / step size to ensure w converge$\nabla f(w_t)$

- If the problem is convex, we are guaranteed to find global minimum if we choose step size appropriately
- The gradient, $\nabla f(w_t)$ is the direction of the steepest ascent. This is easily seen in 2D, and can be proved by using the Taylor Expansion Theorem on f in higher dimensions.
- $f(w_t + tu) = f(w_t) + (tu)^T \nabla f(w_t) = f(w_t) + t(u^T \nabla f(w_t))$
- Recall that $< u, \nabla f(w_t) >= |u||\nabla f(w_t)|cos(\psi)$. This expression is maximized when $\psi = 0$. Which means that u is collinear with the gradient. And u maximizes the expression.


$J(x)$
Gradient direction has same sign as the derivative
Derivative is slope of line tangent to $J(x)$ at x
x_n
Minimum
x_p
Arrows point in minus gradient direction towards the minimum

2.4 Coordinate Descent and line search

Coordinate Descent and line search look for **sparse** solutions, but what are their advantages? In the context of primal problems, if the weight vector $\vec{w}$ is sparse, we can discard features with 0 weight after training, since they will have no effect on the regression values of test data. A similar reasoning applies to dual problems with dual weight vector $\vec{v}$, allowing us to discard the training points corresponding to dual weight 0, ultimately allowing for faster evaluation of our hypothesis function on test points.[5]
The general procedure of Coordinate Decent follows:

1. Pick a direction $\vec{d}$

[5]CS189 SP18 Note 24

2. Optimize $f(\vec{w_i} + \eta\vec{d})$ as a function of η

3. Move to $\vec{w_{t+1}} = \vec{w_t} + \eta^*\vec{d}$

Ideally, we want to find an algorithm that solves the **Sparse Lest Square** objective

$$\min_w ||Xw - y||_2^2$$
$$s.t. ||w||_0 \leq k$$

But l_0 norm (the number of non-zero elements) does not actually satisfy the properties of a norm, so solving this optimization problem is NP-Hard. But by a relaxed version, where l_0 is replaced with l_1, can be solved. This is what we called **LASSO (least absolute shrinkage and selection operator)**.

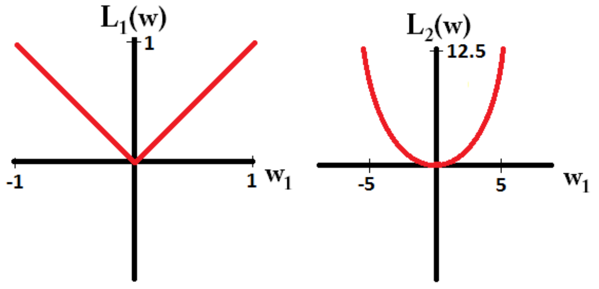
The objective of LASSO:

$$\min_w ||Xw - y||_2^2$$
$$s.t. ||w||_1 \leq k$$

Due to strong duality, we can express the LASSO objective in unconstrained form

$$\min_w ||Xw - y||_2^2 + \lambda||w||_1$$

LASSO works well with coordinate descent
The "pointiness" of L1 norm encourages sparsity seeking: It applies a constant pull towards 0 on each dim unless enough force from data



2.5 Matching pursuit (Gradient Boosting)

Rather than relaxing the l_0 constraint (as seen in LASSO), the matching pursuit algorithm keeps the constraint, and instead finds an approximate solution to the spare least squares problem in a greedy fashion. Therefore, it might be easier to view Matching Pursuit as "greedy coordinate descent"
A general procedure of Matching Pursuit Algorithm follows:

1. The algorithm starts with with a completely sparse solution ($\vec{w}^0 = \vec{0}$)
2. At each step, pick "the direction", which **must be axis aligned**, that maximizes inner product with gradient of loss. In mathematical language, at each time step t, the algorithm can only update one entry of $\vec{w}^{t-1}$
3. $\vec{w}$ will be iteratively updated until the sparsity constraint $||\vec{w}||_0 \leq k$ can no longer be met.

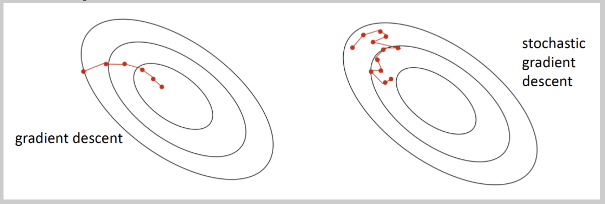
From the process, we can see that the resulting $\vec{w}$ is guaranteed to be sparse (it will contain at most k non-zero entries). This is closely related to the idea of *early stopping*

2.6 Stochastic Gradient descent and Mini-batch Gradient Descent

Why might it be sensible to compute gradient with fewer sample?
We can approximate the gradient with a *stochastic* estimate of the gradient.[6]

$$\nabla f(w_t) \approx \nabla f_S(w_t) \equiv \frac{1}{S} \sum_{i}^{S} \nabla f_i(w_t)$$

Story setting 1
Professor visits colleagues in new city. After dinner, she needs to get to hotel, but her phone battery is dead. Every few blocks, asks for directions from students, but students have been partying, so when they point to her hotel, she gets only a noisy estimate of the direction. [a]



[a]Analogy courtesy of David Bleiat Columbia

Now back to the algorithm. Assume that f can be expressed in the form $\sum_{i=1}^{n}[\frac{1}{n} * f_i(w)]$.

- SGD

[6]CS 189 SP18 SGD lecture slides

1. w_0 is initialized to be a random vector in R^d, where d is the dimensionality of the parameter vector w.
2. For $t = 1, 2, 3 \ldots$:

   $$w_{t+1} = w_t - \eta \nabla f_i(w_t)$$

   where i is an index sampled uniformly from 1,2,3…n
3. if sufficient progress is not made, terminate and return w_t. Otherwise, repeat step 2

- Mini-batch
  Similar to SGD, but pick a random subset of sub-samples and use the average of gradients on examples in the subset.

The Idea behind SGD /minibatch :

- The time complexity for computing the gradient is linear in n, the number of gradients
- When n is large, it might be really slow. In order to speed up the computing process, can we randomly sub-sample the training set and still get similar convergence behavior.