CPSC 340: Machine Learning and Data Mining

More Regularization

Fall 2020

Last Time: L2-Regularization

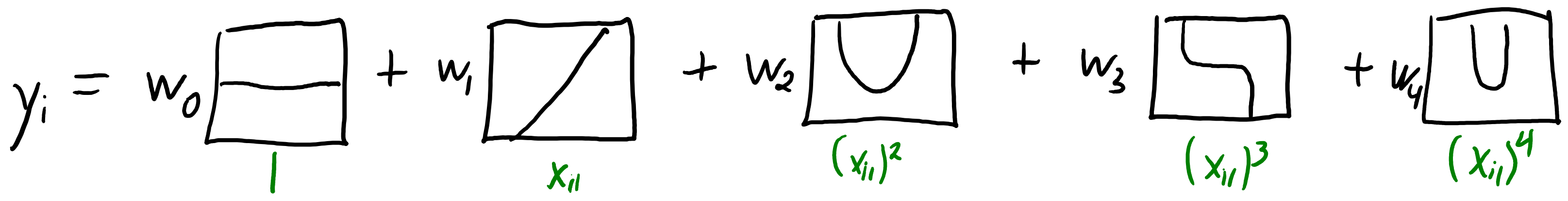
- We discussed regularization:
  - Adding a continuous penalty on the model complexity:

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2$$

  - Best parameter λ almost always leads to improved test error.
    - L2-regularized least squares is also known as "ridge regression".
    - Can be solved as a linear system like least squares.
  - Numerous other benefits:
    - Solution is unique, less sensitive to data, gradient descent converges faster.

Parametric vs. Non-Parametric Transforms

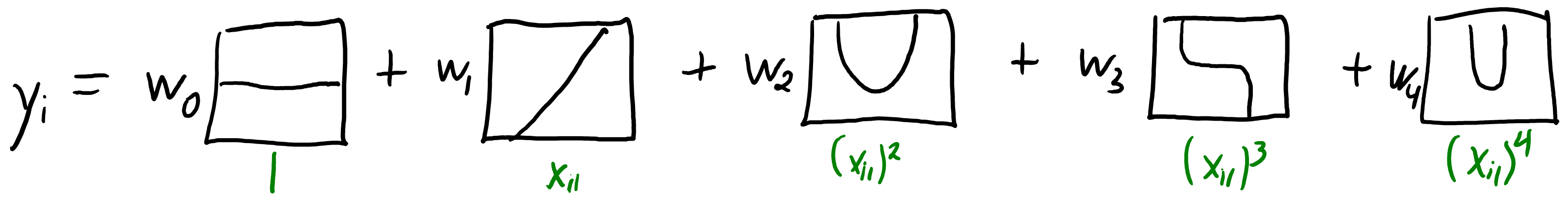
- We’ve been using linear models with polynomial bases:



- But polynomials are not the only possible bases:
  - Exponentials, logarithms, trigonometric functions, etc.
  - The right basis will vastly improve performance.
  - If we use the wrong basis, our accuracy is limited even with lots of data.
  - But the right basis may not be obvious.

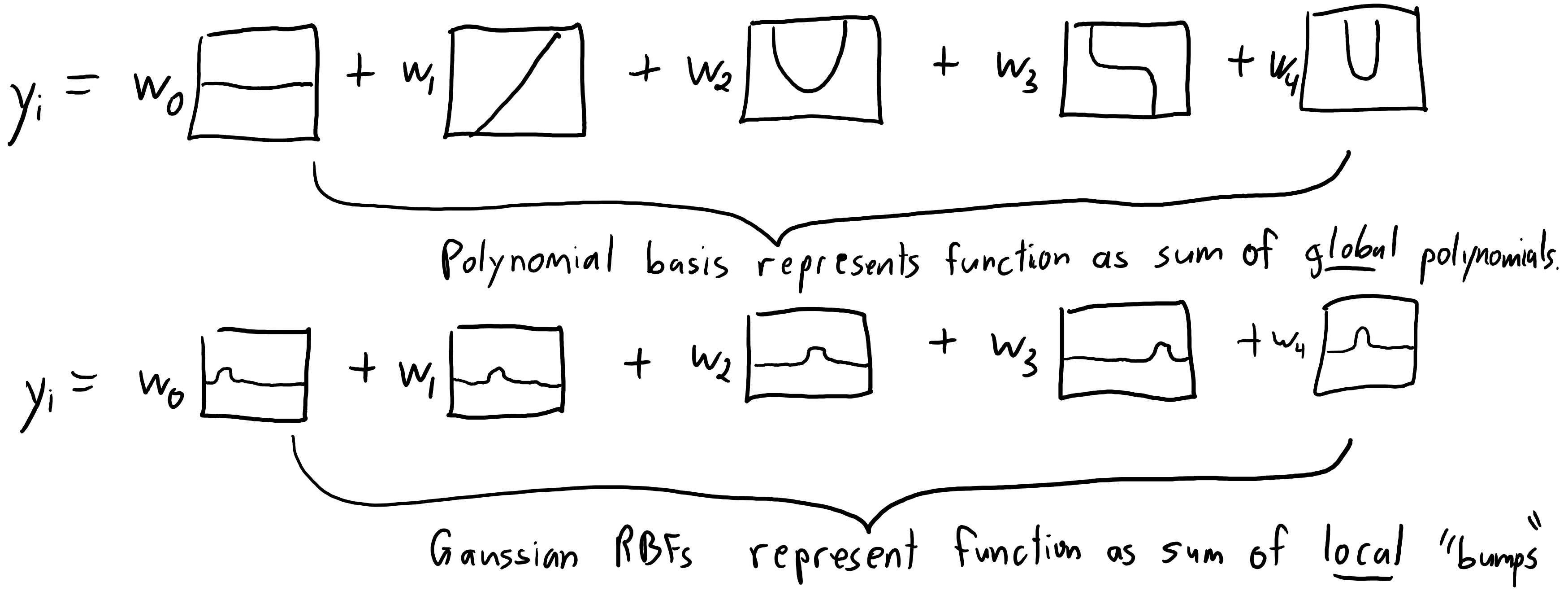
Parametric vs. Non-Parametric Transforms

- We've been using linear models with polynomial bases:



- Alternative is non-parametric bases:
  - Size of basis (number of features) grows with 'n'.
  - Model gets more complicated as you get more data.
  - Can model complicated functions where you don't know the right basis.
    - With enough data.
  - Classic example is "Gaussian RBFs" ("Gaussian" == "normal distribution").

Gaussian RBFs: A Sum of "bumps"



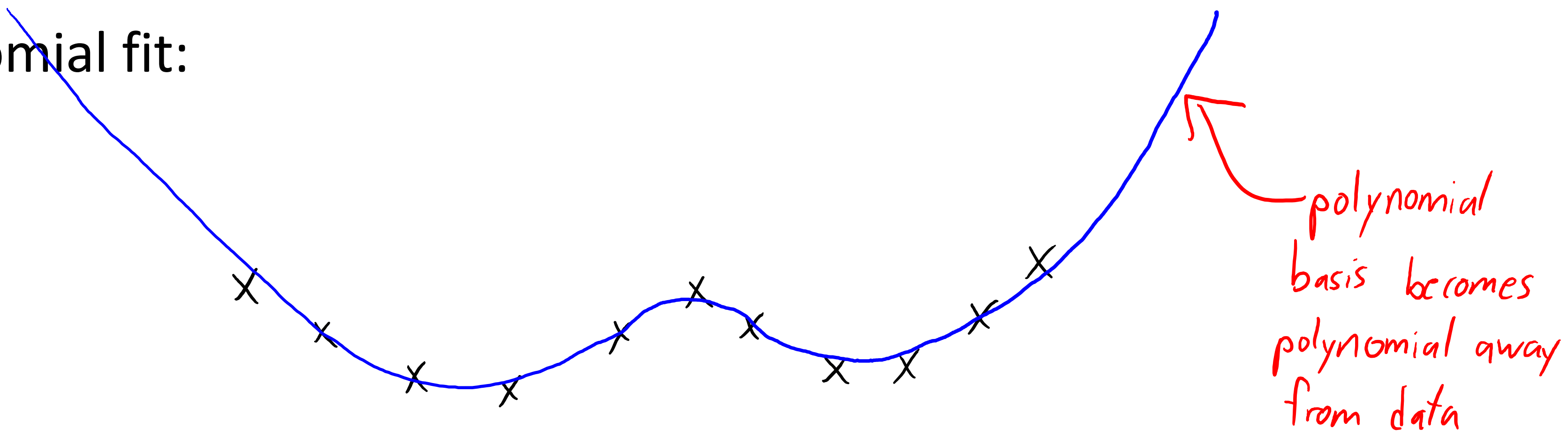
Polynomial basis represents function as sum of <u>global</u> polynomials.

Gaussian RBFs represent function as sum of <u>local</u> "bumps"

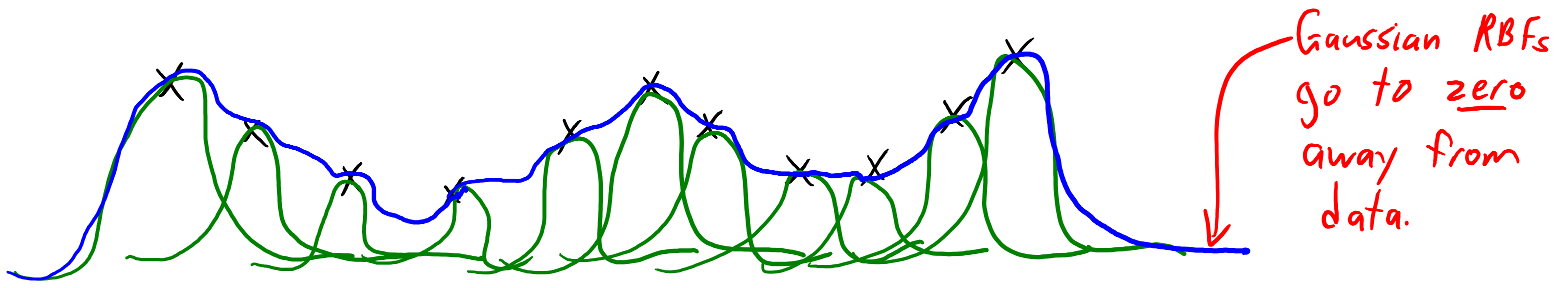
- Gaussian RBFs are universal approximators (compact subsets of $\mathbb{R}^d$)
  - Enough bumps can approximate any continuous function to arbitrary precision.
  - Achieve optimal test error as 'n' goes to infinity.

Gaussian RBFs: A Sum of "Bumps"

- Polynomial fit:

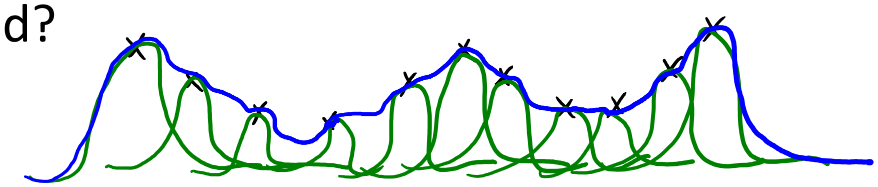


- Constructing a function from bumps ("smooth histogram"):



Gaussian RBF Parameters

- Some obvious questions:
  1. How many bumps should we use?
  2. Where should the bumps be centered?
  3. How high should the bumps go?
  4. How wide should the bumps be?



- The usual answers:
  1. We use 'n' bumps (non-parametric basis).
  2. Each bump is centered on one training example x_i.
  3. Fitting regression weights 'w' gives us the heights (and signs).
  4. The width is a hyper-parameter (narrow bumps == complicated model).

Gaussian RBFs: Formal Details

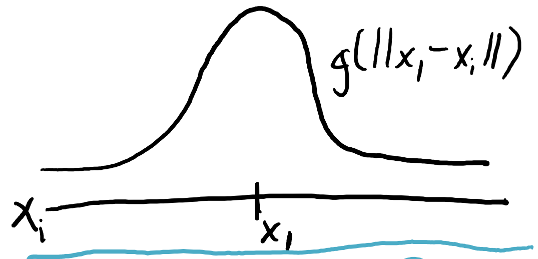
- What is a radial basis functions (RBFs)?
  - A set of non-parametric bases that depend on distances to training points.

Replace $x_i = \underbrace{(x_{i1}, x_{i2}, \dots, x_{in})}_{\text{'d' features}}$ with $z_i = (\underbrace{g(\|x_i - x_1\|), g(\|x_i - x_2\|), \dots, g(\|x_i - x_n\|)}_{\text{'n' features}})$

  - Have 'n' features, with feature 'j' depending on distance to example 'i'.
  - Most common 'g' is Gaussian RBF:

$$g(\varepsilon) = \exp\left(-\frac{\varepsilon^2}{2\sigma^2}\right)$$

    - Variance σ^2 is a hyper-parameter controlling "width".
      - This affects fundamental trade-off (set it using a validation set).



Do we need $\sigma\sqrt{2\pi}$?
– No because $v^T z_i = (\frac{1}{\beta} v)^T (\beta z_i)$

Gaussian RBFs: Formal Details

- What is a radial basis functions (RBFs)?
  - A set of non-parametric bases that depend on distances to training points.

Replace $X = \underbrace{\begin{bmatrix} \\ \\ \\ \end{bmatrix}}_{d} \Big\} n$ by $Z = \underbrace{\begin{bmatrix} g(\|x_1 - x_1\|) & g(\|x_1 - x_2\|) & \cdots & g(\|x_1 - x_n\|) \\ g(\|x_2 - x_1\|) & g(\|x_2 - x_2\|) & \cdots & g(\|x_2 - x_n\|) \\ \vdots & & \ddots & \vdots \\ g(\|x_n - x_1\|) & g(\|x_n - x_2\|) & \cdots & g(\|x_n - x_n\|) \end{bmatrix}}_{n} \Bigg\} n$

To make predictions on $\tilde{X} = \underbrace{\begin{bmatrix} \\ \\ \\ \end{bmatrix}}_{d} \Big\} t$ use $\tilde{Z} = \underbrace{\begin{bmatrix} \\ & g(\|\tilde{x}_i - x_j\|) & \\ \\ \end{bmatrix}}_{n} \Big\} t$

Number of "features" is number of training examples.

Gaussian RBFs: Pseudo-Code

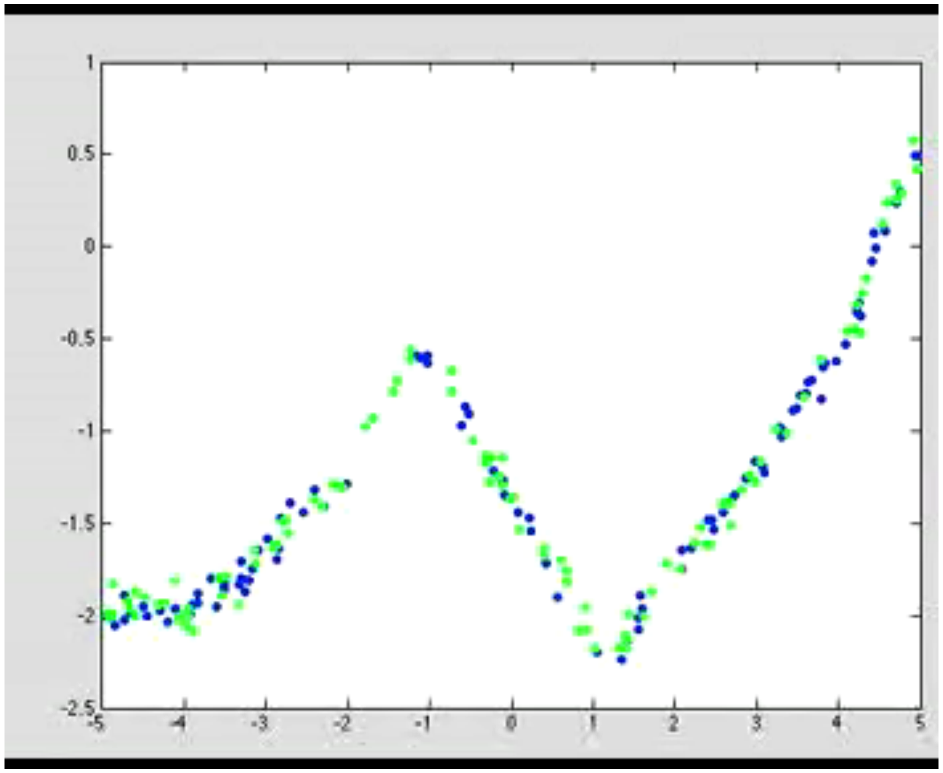
Constructing Gaussian RBFs given data 'X' and hyper-parameter σ:

```
Z = zeros(n,n)
for i1 in 1:n
    for i2 in 1:n
        $Z[i1, i2] = \exp(-\text{norm}(X[i1,:] - X[i2,:])^2 / 2\sigma^2)$
```

With test data $\tilde{X}$: form $\tilde{Z}$ based on distances to training examples.

Non-Parametric Basis: RBFs

- Least squares with Gaussian RBFs for different σ values:



Could add <u>bias</u> and linear basis:

$$Z = \begin{array}{c} \left[\begin{array}{ccccc} 1 & -x_1- & g(\|x_1-x_1\|) & \cdots & g(\|x_1-x_n\|) \\ 1 & -x_2- & \vdots & \ddots & \vdots \\ 1 & -x_3- & & & \\ \vdots & \vdots & & & \\ 1 & -x_n- & g(\|x_1-x_n\|) & \cdots & g(\|x_n-x_n\|) \end{array}\right] \\ \begin{array}{ccc} \underbrace{\quad}_{1} & \underbrace{\qquad}_{d} & \underbrace{\qquad\qquad\qquad\qquad\qquad}_{n} \end{array} \end{array}$$

This reverts to <u>linear regression</u> instead of 0 away from data.

RBFs and Regularization

- Gaussian Radial basis functions (RBFs) predictions:

$$\hat{y}_i = w_1 \exp\left(-\frac{\|x_i - x_1\|^2}{2\sigma^2}\right) + w_2 \exp\left(-\frac{\|x_i - x_2\|^2}{2\sigma^2}\right) + \cdots + w_n \exp\left(-\frac{\|x_i - x_n\|^2}{2\sigma^2}\right)$$

$$= \sum_{j=1}^{n} w_j \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

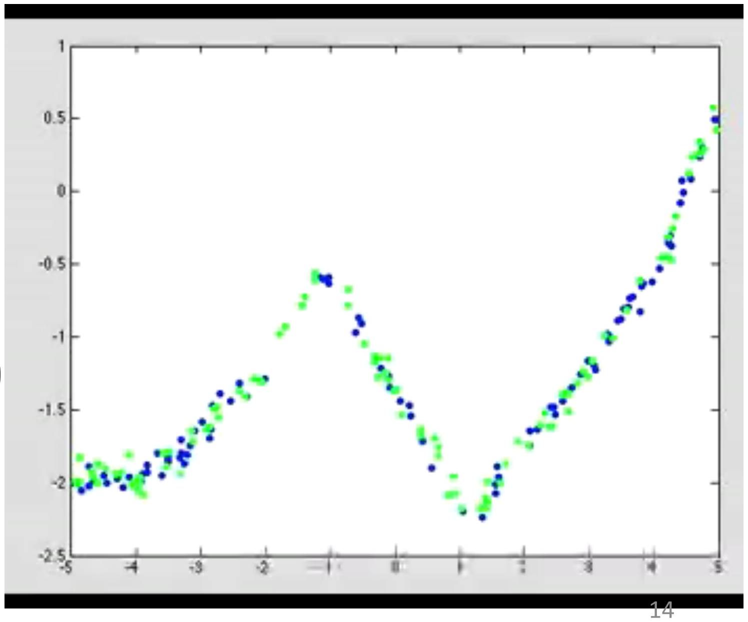
  - Flexible bases that can model any continuous function.
  - But with 'n' data points RBFs have 'n' basis functions.

- How do we avoid overfitting with this huge number of features?
  - We regularize 'w' and use validation error to choose σ and λ.

RBFs, Regularization, and Validation

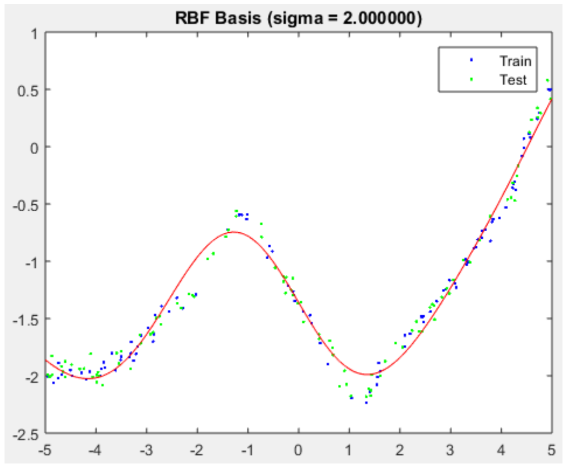
- A model that is hard to beat:
  - RBF basis with L2-regularization and cross-validation to choose σ and λ.
  - Flexible non-parametric basis, magic of regularization, and tuning for test error.

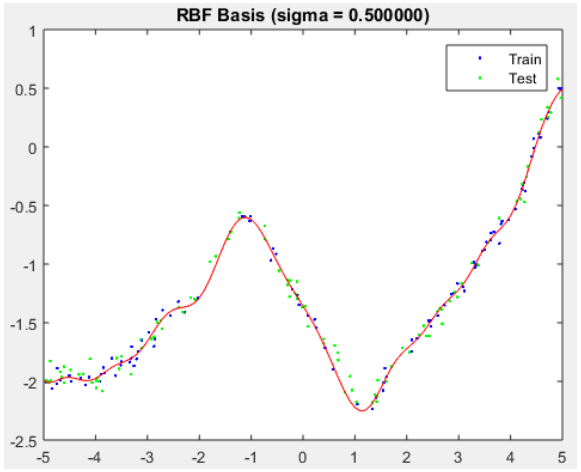
```
for each value of $\lambda$ and $\sigma$:
    - Compute Z on training data (and $\sigma$)
    - Compute best v: $v = (Z^T Z + \lambda I)^{-1} Z^T y$
    - Compute $\tilde{Z}$ on validation data (using train data distances)
    - Make predictions $\hat{y} = \underbrace{\tilde{Z}}_{t \times n} \underbrace{v}_{n \times 1}$
    - Compute validation error $\|\hat{y} - \tilde{y}\|^2$
```

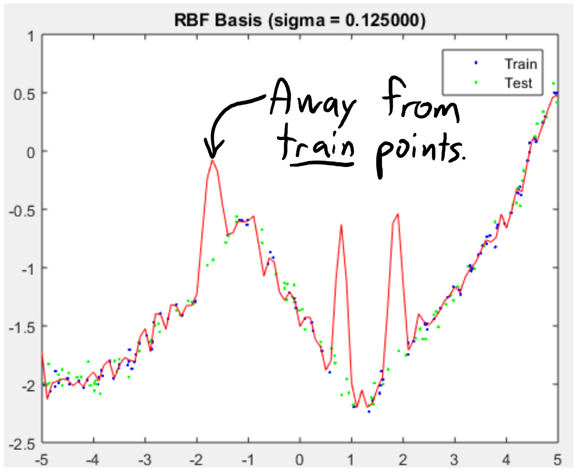


RBFs, Regularization, and Validation

- A model that is hard to beat:
  - RBF basis with L2-regularization and cross-validation to choose σ and λ.
  - Flexible non-parametric basis, magic of regularization, and tuning for test error!







  - Expensive at test time: needs distance to all training examples.

Hyper-Parameter Optimization

- In this setting we have 2 hyper-parameters (σ and λ).
- More complicated models have even more hyper-parameters.
  - This makes searching all values expensive (increases over-fitting risk).

- Leads to the problem of hyper-parameter optimization.
  - Try to efficiently find “best” hyper-parameters.

- Simplest approaches:
  - Exhaustive search: try all combinations among a fixed set of σ and λ values.
  - Random search: try random values.

Hyper-Parameter Optimization

- Other common hyper-parameter optimization methods:
  - Exhaustive search with pruning:
    - If it “looks” like test error is getting worse as you decrease λ, stop decreasing it.
  - Coordinate search:
    - Optimize one hyper-parameter at a time, keeping the others fixed.
    - Repeatedly go through the hyper-parameters
  - Stochastic local search:
    - Generic global optimization methods (simulated annealing, genetic algorithms, etc.).
  - Bayesian optimization:
    - Use RBF regression to build model of how hyper-parameters affect validation error.
    - Try the best guess based on the model.

(pause)

Previously: Search and Score

- We talked about search and score for feature selection:
  - Define a "score" and "search" for features with the best score.
- Usual scores count the number of non-zeroes ("L0-norm"):

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \lambda \underbrace{\|w\|_0}_{\substack{\text{number of} \\ \text{non-zeroes} \\ \text{in 'w'}}}$$

- But it's hard to find the 'w' minimizing this objective.
- We discussed forward selection, but requires fitting $O(d^2)$ models.

Previously: Search and Score

- What if we want to pick among millions or billions of variables?

- If 'd' is large, forward selection is too slow:
  - For least squares, need to fit $O(d^2)$ models at cost of $O(nd^2 + d^3)$.
  - Total cost $O(nd^4 + d^5)$.

- The situation is worse if we aren't using basic least squares:
  - For robust regression, need to run gradient descent $O(d^2)$ times.
  - With regularization, need to search for lambda $O(d^2)$ times.

L1-Regularization

- Instead of L0- or L2-norm, consider regularizing by the L1-norm:

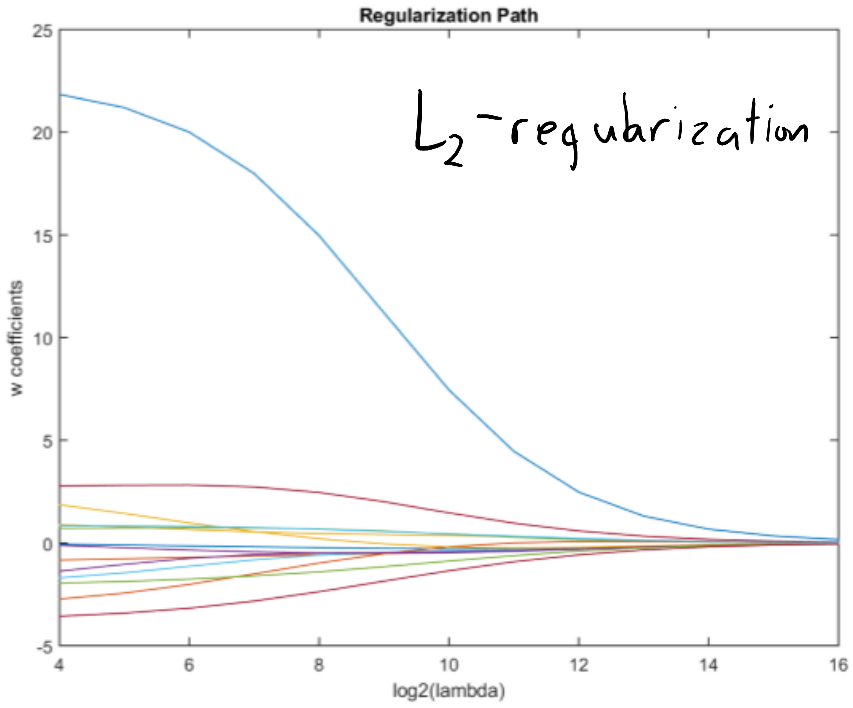
$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \lambda\|w\|_1$$

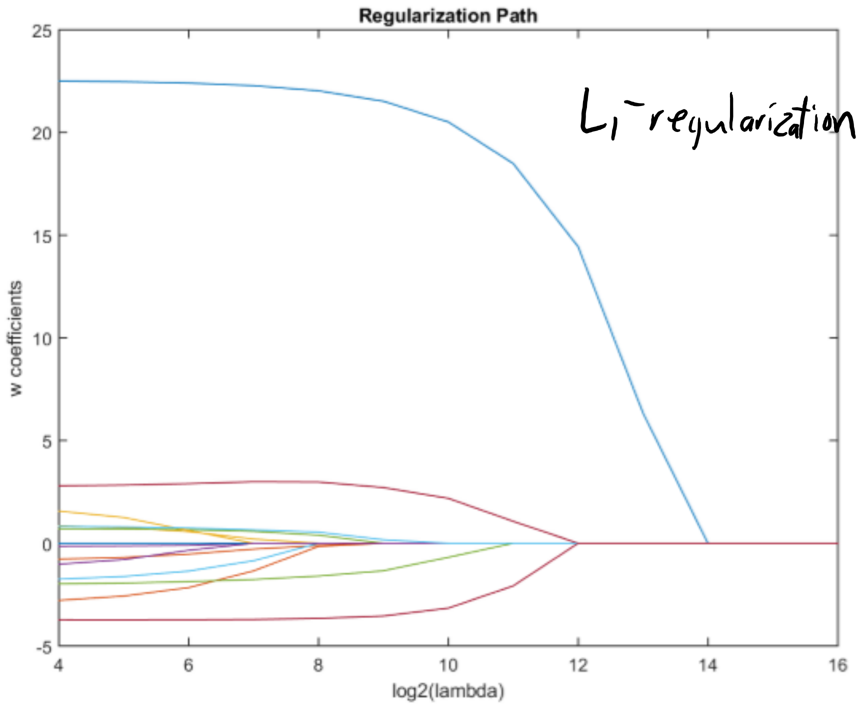
- Like L2-norm, it's convex and improves our test error.
- Like L0-norm, it encourages elements of 'w' to be exactly zero.

- L1-regularization simultaneously regularizes and selects features.
  - Very fast alternative to search and score.
  - Sometimes called "LASSO" regularization.

L2-Regularization vs. L1-Regularization

- Regularization path of w_i values as ‘λ’ varies:





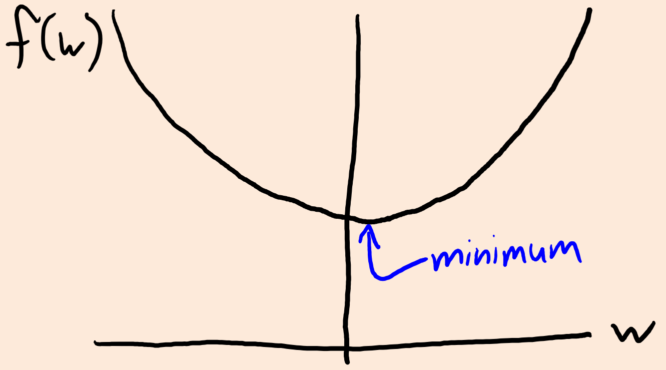
- L1-Regularization sets values to exactly 0 (next slides explore why).

Sparsity and Least Squares

- Consider 1D least squares objective:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w x_i - y_i)^2$$

- This is a convex 1D quadratic function of ‘w’ (i.e., a parabola):



- This variable does not look relevant (minimum is close to 0).
  - But for finite ‘n’ the minimum is unlikely to be exactly zero.

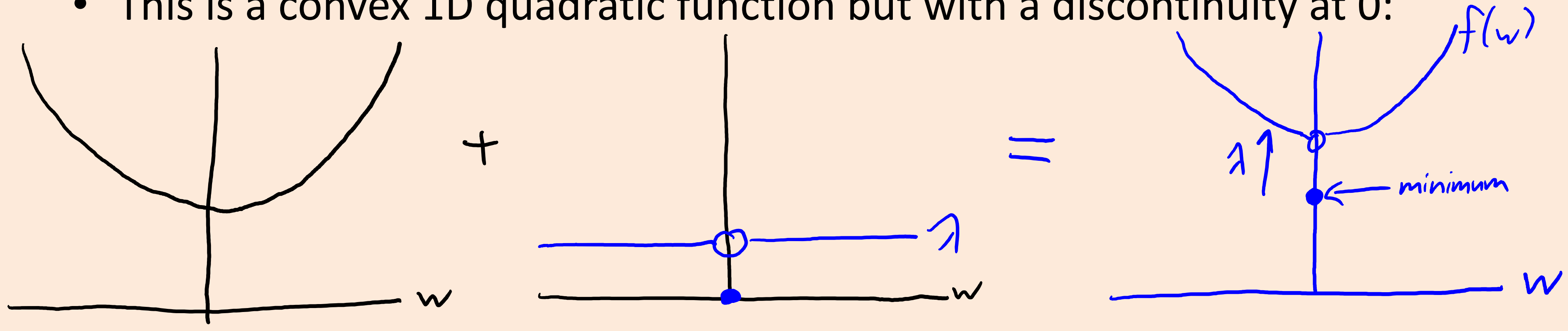
$f'(0) = 0$ only happens if $\sum_{i=1}^{n} y_i x_i = 0$. (bonus)

Sparsity and L0-Regularization

- Consider 1D L0-regularized least squares objective:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w x_i - y_i)^2 + \underbrace{\lambda \|w\|_0}_{} \rightarrow \begin{cases} \lambda & \text{if } w \neq 0 \\ 0 & \text{if } w = 0 \end{cases}$$

- This is a convex 1D quadratic function but with a discontinuity at 0:



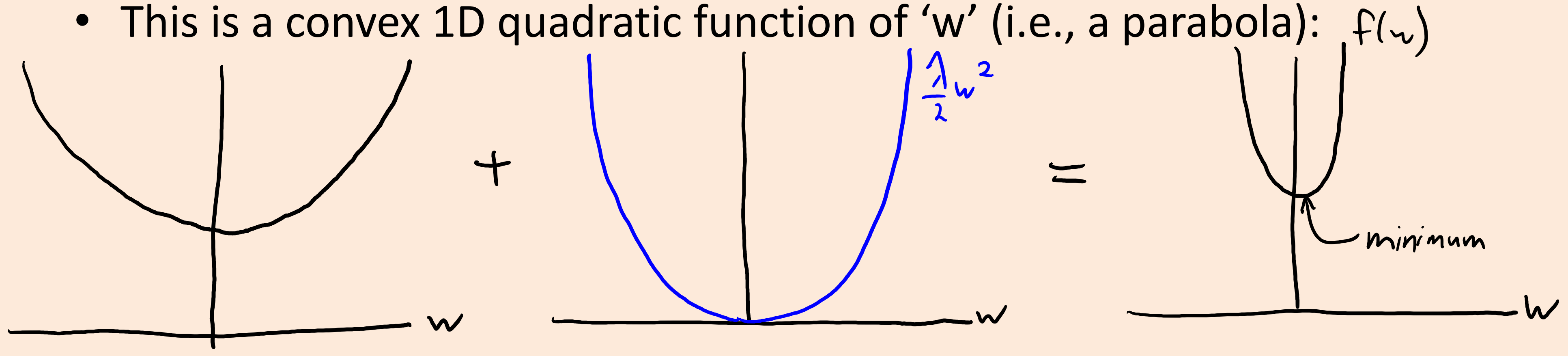
- L0-regularized minimum is often exactly at the 'discontinuity' at 0:
  - Sets the feature to exactly 0 (does feature selection), but is non-convex.

Sparsity and L2-Regularization

- Consider 1D L2-regularized least squares objective:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w x_i - y_i)^2 + \frac{\lambda}{2}w^2$$

- This is a convex 1D quadratic function of 'w' (i.e., a parabola):



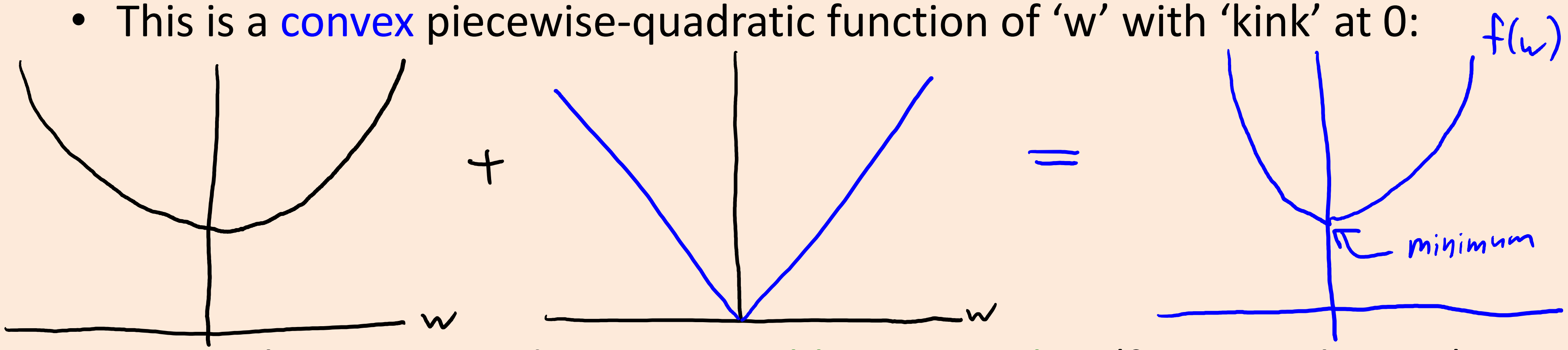
- L2-regularization moves it closer to zero, but not all the way to zero.
  - It doesn't do feature selection ("penalty goes to 0 as slope goes to 0"). $\rightarrow f'(0) = 0$ $\underline{\text{only}}$ if $\sum_{i=1}^{n} y_i x_i = 0$

Sparsity and L1-Regularization

- Consider 1D L1-regularized least squares objective:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w x_i - y_i)^2 + \lambda|w|$$

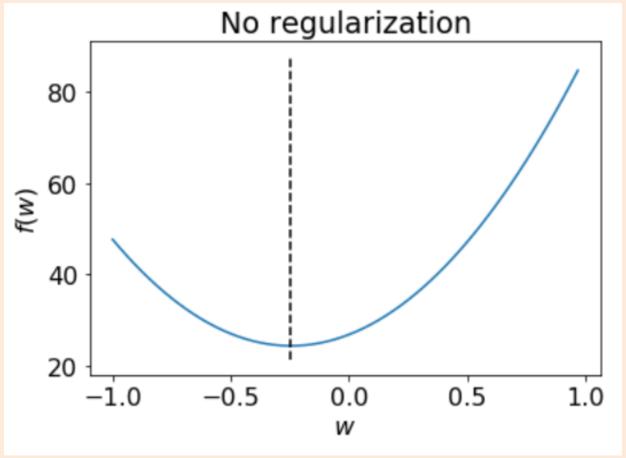
- This is a convex piecewise-quadratic function of 'w' with 'kink' at 0:

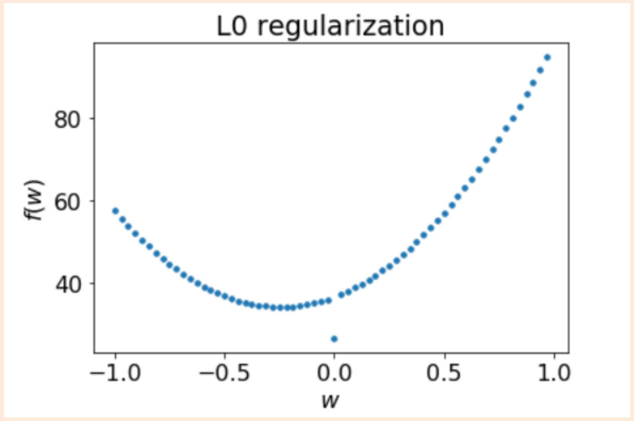


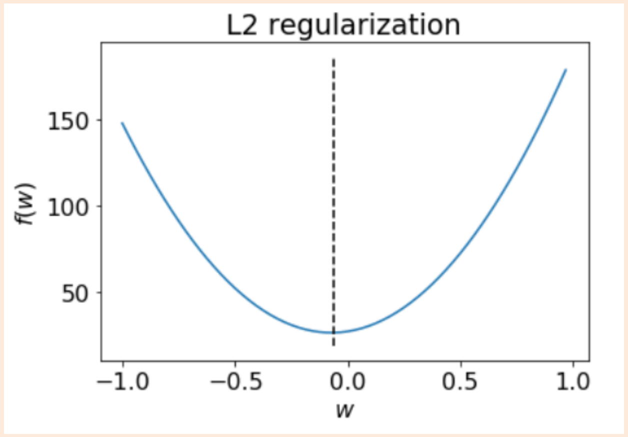
- L1-regularization tends to set variables to exactly 0 (feature selection).
  - Penalty on slope is λ even if you are close to zero.
  - Big λ selects few features, small λ allows many features.

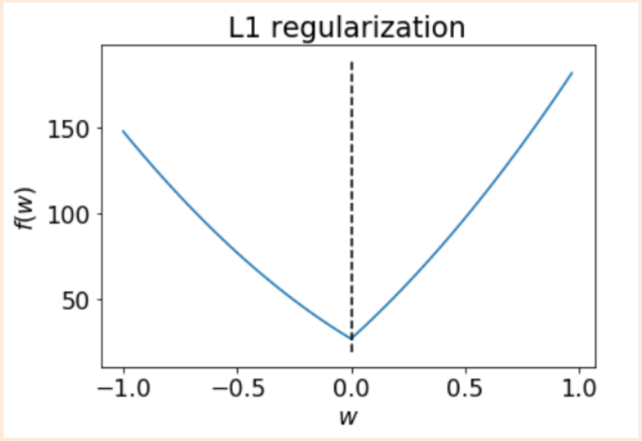
$\rightarrow$ Happens when $\left|\sum_{i=1}^{n} x_i y_i\right| \leq \lambda$
(bonus)

Sparsity and Regularization (with d=1)



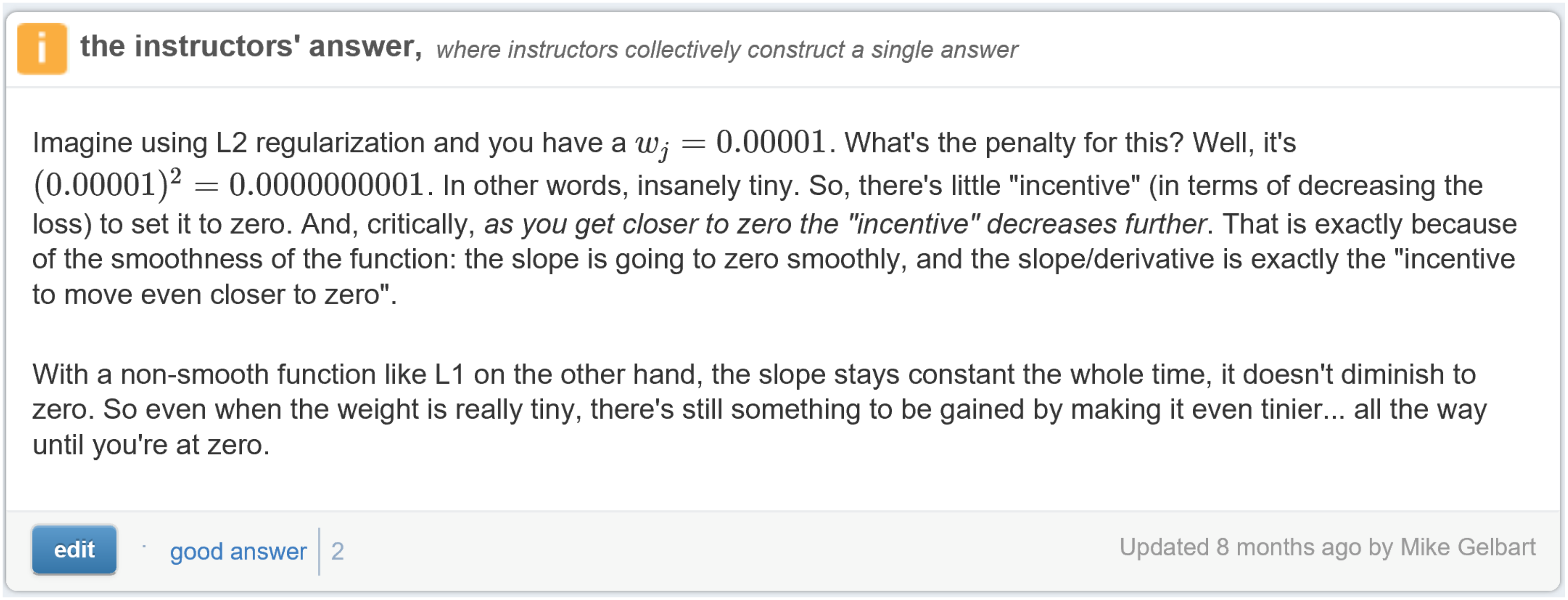






Regularizers and Sparsity

- L1-regularization gives sparsity but L2-regularization doesn't.
  - But don't they both shrink variables towards zero?



the instructors' answer, *where instructors collectively construct a single answer*

Imagine using L2 regularization and you have a $w_j = 0.00001$. What's the penalty for this? Well, it's $(0.00001)^2 = 0.0000000001$. In other words, insanely tiny. So, there's little "incentive" (in terms of decreasing the loss) to set it to zero. And, critically, *as you get closer to zero the "incentive" decreases further*. That is exactly because of the smoothness of the function: the slope is going to zero smoothly, and the slope/derivative is exactly the "incentive to move even closer to zero".

With a non-smooth function like L1 on the other hand, the slope stays constant the whole time, it doesn't diminish to zero. So even when the weight is really tiny, there's still something to be gained by making it even tinier... all the way until you're at zero.

edit · good answer | 2 Updated 8 months ago by Mike Gelbart

L2-Regularization vs. L1-Regularization

- L2-Regularization:
  - Insensitive to changes in data.
  - Decreased variance:
    - Lower test error.
  - Closed-form solution.
  - Solution is unique.
  - All 'w_j' tend to be non-zero.
  - Can learn with *linear* number of irrelevant features.
    - E.g., only O(d) relevant features.

- L1-Regularization:
  - Insensitive to changes in data.
  - Decreased variance:
    - Lower test error.
  - Requires iterative solver.
  - Solution is not unique.
  - Many 'w_j' tend to be zero.
  - Can learn with **exponential** number of irrelevant features.
    - E.g., only O(log(d)) relevant features. Paper on this result by Andrew Ng

L1-loss vs. L1-regularization

- Don't confuse the L1 loss with L1-regularization!
  - L1-loss is robust to outlier data points.
    - You can use this instead of removing outliers.
  - L1-regularization is robust to irrelevant features.
    - You can use this instead of removing features.
- And note that you can be robust to outliers and irrelevant features:

$$f(w) = \underbrace{\|Xw - y\|_1}_{L_1 - \text{loss}} + \lambda \underbrace{\|w\|_1}_{L_1 - \text{regularizer}}$$

- Can we smooth and use "Huber regularization"?
  - Huber regularizer is still robust to irrelevant features.
  - But it's the non-smoothness that sets weights to exactly 0.

L*-Regularization

- L0-regularization (AIC, BIC, Mallow's Cp, Adjusted R^2, ANOVA):
  - Adds penalty on the number of non-zeros to select features.

$$f(w) = \|Xw - y\|^2 + \lambda\|w\|_0$$

- L2-regularization (ridge regression):
  - Adding penalty on the L2-norm of 'w' to decrease overfitting:

$$f(w) = \|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2$$

- L1-regularization (LASSO):
  - Adding penalty on the L1-norm decreases overfitting and selects features:

$$f(w) = \|Xw - y\|^2 + \lambda\|w\|_1$$

L0- vs. L1- vs. L2-Regularization

	Sparse 'w' (Selects Features)	Speed	Unique 'w'	Coding Effort	Irrelevant Features
L0-Regularization	Yes	Slow	No	Few lines	Not Sensitive
L1-Regularization	Yes*	Fast*	No	1 line*	Not Sensitive
L2-Regularization	No	Fast	Yes	1 line	A bit sensitive

- L1-Regularization isn't as sparse as L0-regularization.
  - L1-regularization tends to give more false positives (selects too many).
  - And it's only "fast" and "1 line" with specialized solvers.
- Cost of L2-regularized least squares is $O(nd^2 + d^3)$.
  - Changes to $O(ndt)$ for 't' iterations of gradient descent (same for L1).
- "Elastic net" (L1- and L2-regularization) is sparse, fast, and unique.
- Using L0+L2 does not give a unique solution.

Summary

- Radial basis functions:
  - Non-parametric bases that can model any function.
- L1-regularization:
  - Simultaneous regularization and feature selection.
  - Robust to having lots of irrelevant features.

- Next time: are we really going to use regression for classification?

Regularizers and Sparsity

- L1-regularization gives sparsity but L2-regularization doesn't.
  - But don't they both shrink variables to zero?
- Consider problem where 3 vectors can get minimum training error:

$$w^1 = \begin{bmatrix} 100 \\ 0.02 \end{bmatrix} \qquad w^2 = \begin{bmatrix} 100 \\ 0 \end{bmatrix} \qquad w^3 = \begin{bmatrix} 99.99 \\ 0.02 \end{bmatrix}$$

- Without regularization, we could choose any of these 3.
  - They all have same error, so regularization will "break tie".
- With L0-regularization, we would choose w^2:

$$\|w^1\|_0 = 2 \qquad \|w^2\|_0 = 1 \qquad \|w^3\|_0 = 2$$

Regularizers and Sparsity

- L1-regularization gives sparsity but L2-regularization doesn't.
  - But don't they both shrink variables to zero?
- Consider problem where 3 vectors can get minimum training error:

$$w^1 = \begin{bmatrix} 100 \\ 0.02 \end{bmatrix} \qquad w^2 = \begin{bmatrix} 100 \\ 0 \end{bmatrix} \qquad w^3 = \begin{bmatrix} 99.99 \\ 0.02 \end{bmatrix}$$

- With L2-regularization, we would choose w^3:

$$\|w^1\|^2 = 100^2 + 0.02^2 = 10000.0004 \qquad \|w^2\|^2 = 100^2 + 0^2 = 10000 \qquad \|w^3\|^2 = 99.99^2 + 0.02^2 = 9998.0005$$

- L2-regularization focuses on decreasing largest (makes w_j similar).

Regularizers and Sparsity

- L1-regularization gives sparsity but L2-regularization doesn’t.
  - But don’t they both shrink variables to zero?
- Consider problem where 3 vectors can get minimum training error:

$$w^1 = \begin{bmatrix} 100 \\ 0.02 \end{bmatrix} \qquad w^2 = \begin{bmatrix} 100 \\ 0 \end{bmatrix} \qquad w^3 = \begin{bmatrix} 99.99 \\ 0.02 \end{bmatrix}$$

- With L1-regularization, we would choose w^2:

$$\|w^1\|_1 = 100 + 0.02 = 100.02 \qquad \|w^2\|_1 = 100 + 0 = 100 \qquad \|w^3\|_1 = 99.99 + 0.02 = 100.01$$

- L1-regularization focuses on decreasing all w_j until they are 0.

Why doesn't L2-Regularization set variables to 0?

- Consider an L2-regularized least squares problem with 1 feature:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(wx_i - y_i)^2 + \frac{\lambda}{2}w^2$$

- Let's solve for the optimal 'w':

$$f'(w) = \sum_{i=1}^{n} x_i(wx_i - y_i) + \lambda w$$

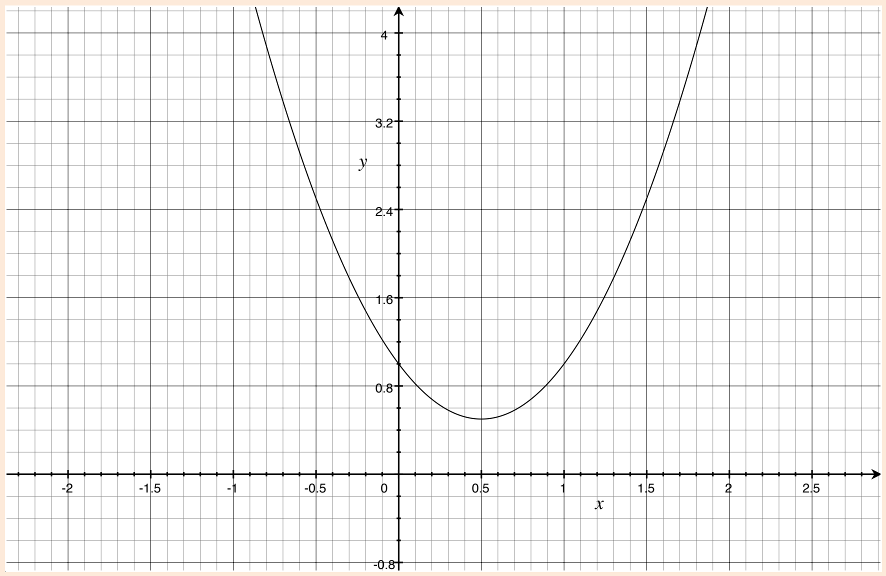
Set equal to 0: $\sum_{i=1}^{n} x_i^2 w - \sum_{i=1}^{n} x_i y_i + \lambda w = 0$ $\xrightarrow{\text{re-arrange}}$ $w(\underbrace{\sum_{i=1}^{n} x_i^2}_{\|x\|^2} + \lambda) = \underbrace{\sum_{i=1}^{n} x_i y_i}_{y^Tx}$

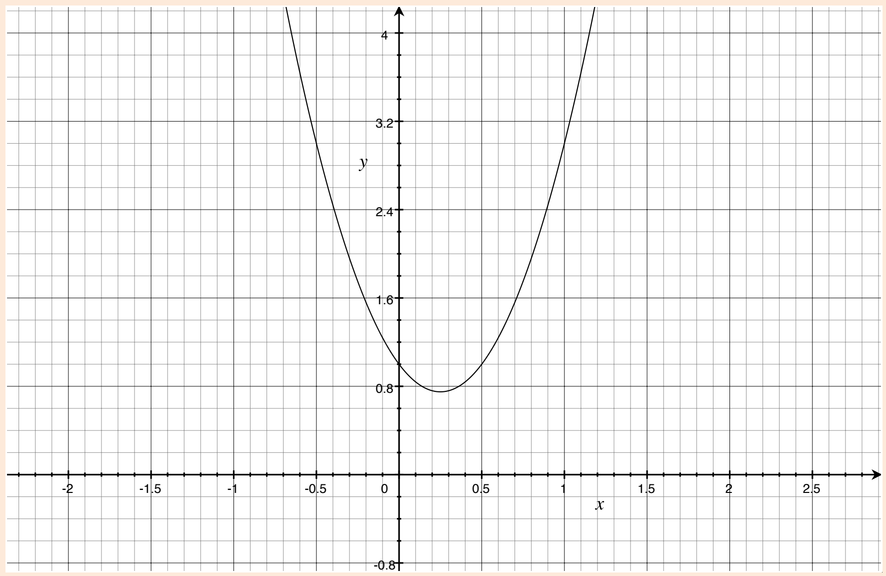
or $w = \frac{y^Tx}{\|x\|^2 + \lambda}$

- So as λ gets bigger, 'w' converges to 0.
- However, for all finite λ 'w' will be non-zero unless $y^Tx = 0$ exactly.
  - But it's very unlikely that y^Tx will be exactly zero.

Why doesn't L2-Regularization set variables to 0?

- Small λ Big λ



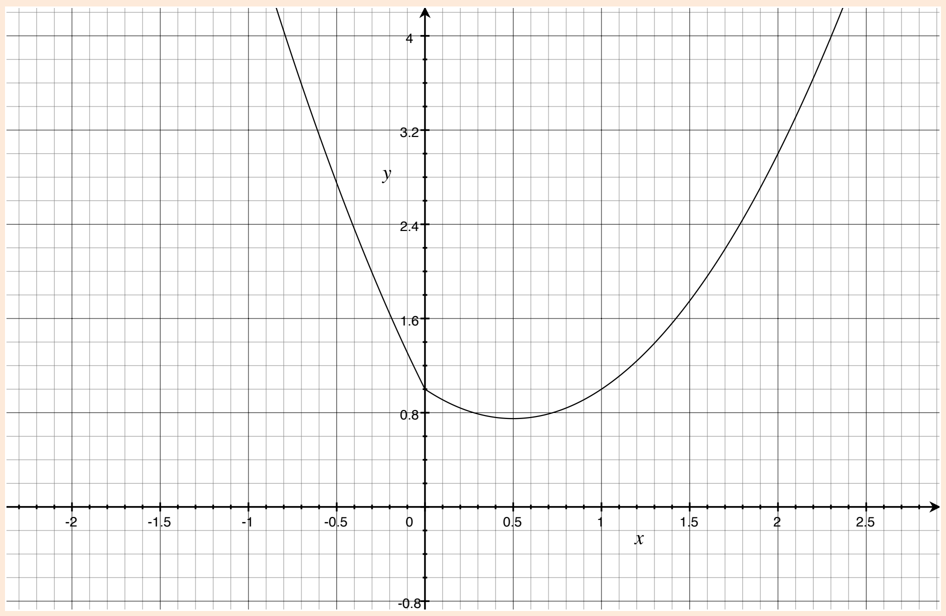


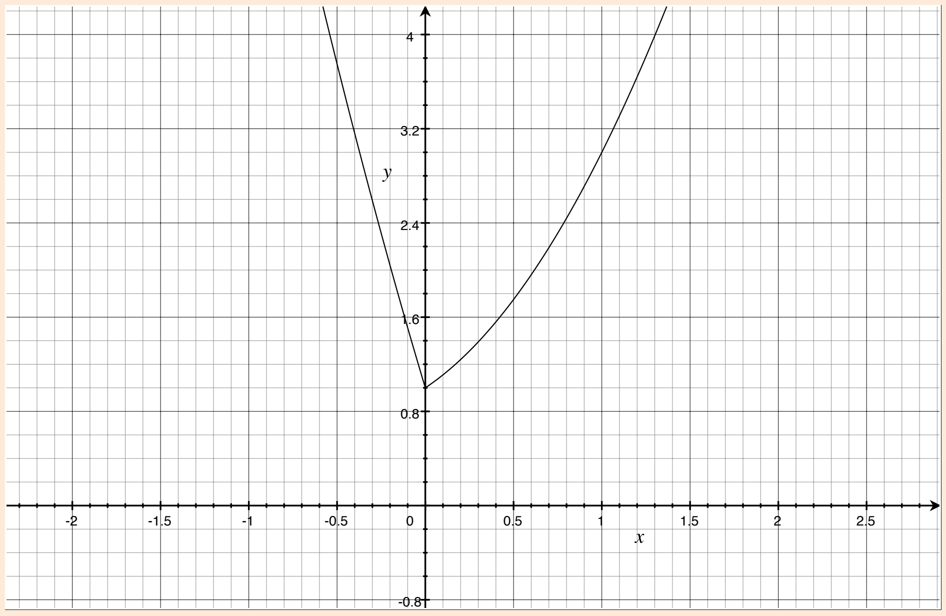
- Solution further from zero

Solution closer to zero
(but not exactly 0)

Why does L1-Regularization set things to 0?

- Small λ　　　　　　　　Big λ





- Solution nonzero　　　　　　　　Solution exactly zero

(minimum of left parabola is past origin, but right parabola is not)　　(minimum of both parabola are past the origin)

Why does L1-Regularization set things to 0?

- Consider an L1-regularized least squares problem with 1 feature:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(wx_i - y_i)^2 + \lambda|w|$$

- If (w = 0), then “left” limit and “right“ limit are given by:

$$f^-(0) = \sum_{i=1}^{n} x_i(0x_i - y_i) - \lambda \qquad\qquad f^+(0) = \sum_{i=1}^{n} x_i(0x_i - y_i) + \lambda$$

$$= \sum_{i=1}^{n} x_i y_i - \lambda \qquad\qquad = \sum_{i=1}^{n} x_i y_i + \lambda$$

- So which direction should “gradient descent” go in?

$$-f^-(0) = -y^Tx + \lambda$$
$$-f^+(0) = \underbrace{-y^Tx}_{} - \lambda$$

} If these are positive $(-y^Tx > \lambda)$, we can improve by increasing 'w'.

If these are negative $(y^Tx > \lambda)$, we can improve by decreasing 'w'.

} But if left and right "gradient descent" directions point in opposite directions $(|y^Tx| \leq \lambda)$, minimum is 0.

L2-regularization vs. L1-regularization

- So with 1 feature:
  - L2-regularization only sets 'w' to 0 if $y^T x = 0$.
    - There is a only a single possible $y^T x$ value where the variable gets set to zero.
    - And λ has nothing to do with the sparsity.
  - L1-regularization sets 'w' to 0 if $|y^T x| \leq \lambda$.
    - There is a range of possible $y^T x$ values where the variable gets set to zero.
    - And increasing λ increases the sparsity since the range of $y^T x$ grows.
- Note that it's important that the function is non-differentiable:
  - Differentiable regularizers penalizing size would need $y^T x = 0$ for sparsity.

L1-Loss vs. Huber Loss

- The same reasoning tells us the difference between the L1 *loss* and the Huber loss. They are very similar in that they both grow linearly far away from 0. So both are both robust but…
  - With the L1 loss the model often passes exactly through some points.
  - With Huber the model doesn’t necessarily pass through any points.

- Why? With L1-regularization we were causing the elements of ’w’ to be exactly 0. Analogously, with the L1-loss we cause the elements of ‘r’ (the residual) to be exactly zero. But zero residual for an example means you pass through that example exactly.

Non-Uniqueness of L1-Regularized Solution

- How can L1-regularized least squares solution not be unique?
  - Isn't it convex?
- Convexity implies that minimum value of f(w) is unique (if exists), but there may be multiple 'w' values that achieve the minimum.

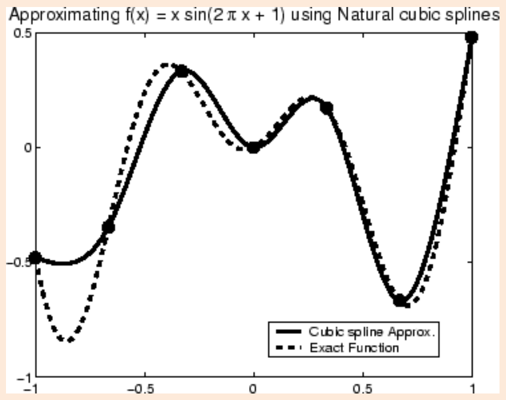
- Consider L1-regularized least squares with d=2, where feature 2 is a copy of a feature 1. For a solution (w_1,w_2) we have:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} = w_1 x_{i1} + w_2 x_{i1} = (w_1 + w_2) x_{i1}$$

- So we can get the same squared error with different w_1 and w_2 values that have the same sum. Further, if neither w_1 or w_2 changes sign, then $|w_1| + |w_2|$ will be the same so the new w_1 and w_2 will be a solution.

Splines in 1D

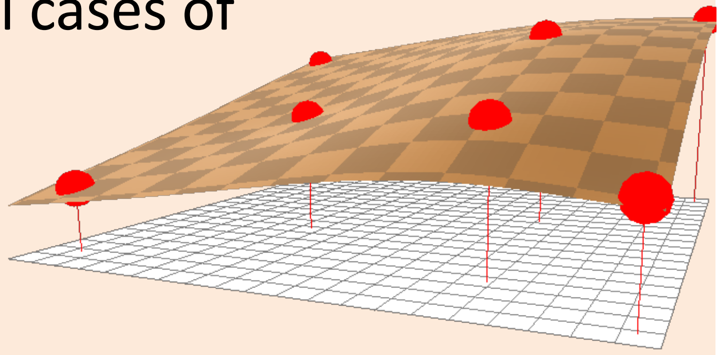
- For 1D interpolation, alternative to polynomials/RBFs are splines:
  - Use a polynomial in the region between each data point.
  - Constrain some derivatives of the polynomials to yield a unique solution.
- Most common example is cubic spline:
  - Use a degree-3 polynomial between each pair of points.
  - Enforce that f'(x) and f''(x) of polynomials agree at all point.
  - "Natural" spline also enforces f''(x) = 0 for smallest and largest x.
- Non-trivial fact: natural cubic splines are sum of:
  - Y-intercept.
  - Linear basis.
  - RBFs with $g(\varepsilon) = \varepsilon^3$.
    - Different than Gaussian RBF because it *increases with distance*.



http://www.physics.arizona.edu/~restrepo/475A/Notes/sourcea-/node35.html

Splines in Higher Dimensions

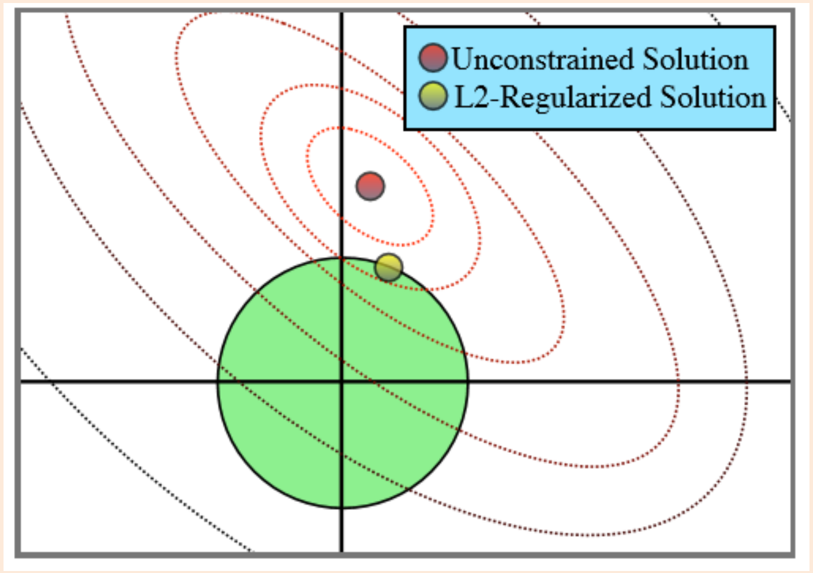
- Splines generalize to higher dimensions if data lies on a grid.
  - Many methods exist for grid-structured data (linear, cubic, splines, etc.).
  - For more general ("scattered") data, there isn't a natural generalization.
- Common 2D "scattered" data interpolation is thin-plate splines:
  - Based on curve made when bending sheets of metal.
  - Corresponds to RBFs with $g(\varepsilon) = \varepsilon^2 \log(\varepsilon)$.
- Natural splines and thin-plate splines: special cases of "polyharmonic" splines:
  - Less sensitive to parameters than Gaussian RBF.



http://step.polymtl.ca/~rv101/thinplates/

L2-Regularization vs. L1-Regularization

- L2-regularization conceptually restricts ‘w’ to a ball.



Minimizing $\frac{1}{2}||Xw-y||^2 + \frac{\lambda}{2}||w||^2$

is <u>equivalent</u> to minimizing

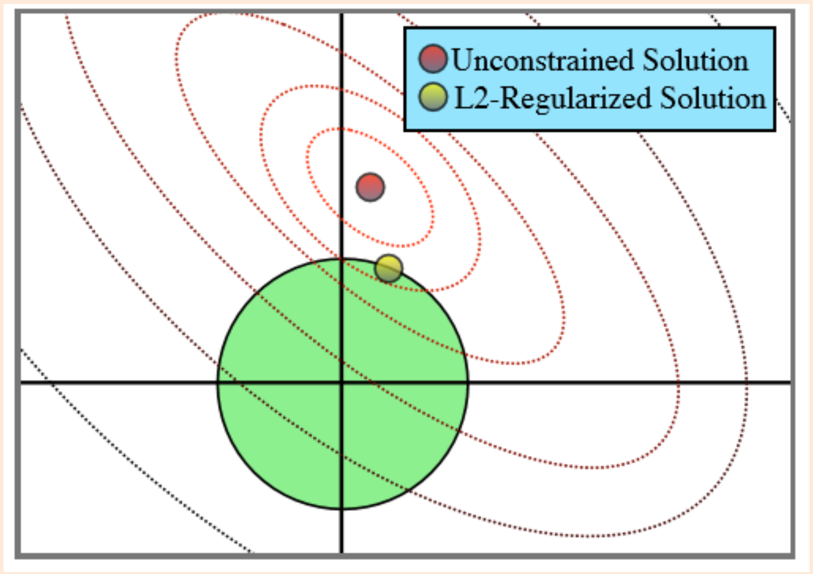
$\frac{1}{2}||Xw-y||^2$ subject to

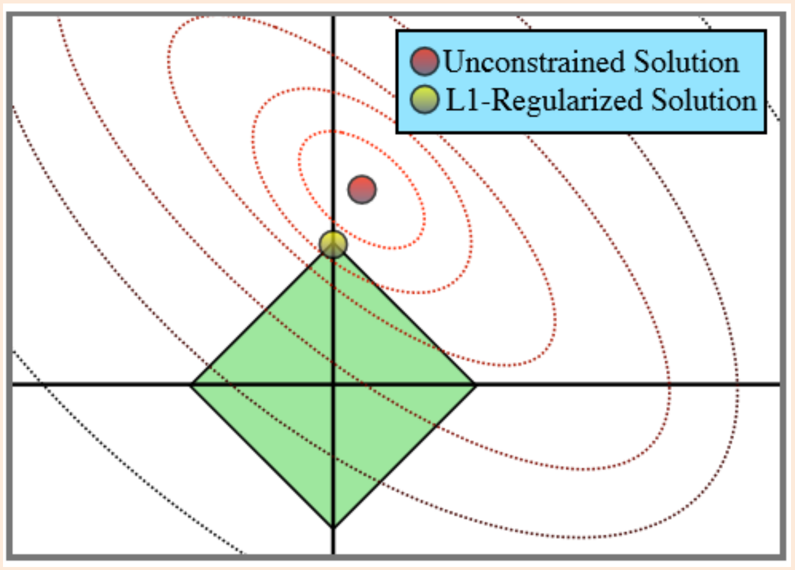
the constraint that $||w|| \leq \tau$

for some value ‘τ’

L2-Regularization vs. L1-Regularization

- L2-regularization conceptually restricts ‘w’ to a ball.





- L1-regularization restricts to the L1 “ball”:
  – Solutions tend to be at corners where w_j are zero.

Related Infinite Series video