

Regularization for linear models

L1 (Lasso), L2 (Ridge)

Terence Parr
MSDS program
University of San Francisco

MIGHT BE MOST POPULAR
INTERVIEW QUESTION

In depth reading:

<https://explained.ai/regularization/index.html>



Motivation for regularization

- 3 main problems with least-squares (OLS) regression:
 - Model with “too many” parameters (e.g., neural nets) will overfit
 - Data sets w/outliers can skew line too much to fit outliers; bad generalization
 - Data sets w/many features can get extreme coefficients in linear models
(see notebook “L1 regularization with normalization”)
- Often unregularized models work but we get extreme coefficients
(extreme negative and positive coefficients must be canceling out)
- L1 (Lasso) regularization also has the advantage that it allows superfluous coefficients to shrink to zero
- Having zero coefficients helps reduce model complexity (fewer coefficients), improving interpretability and usually improving generality

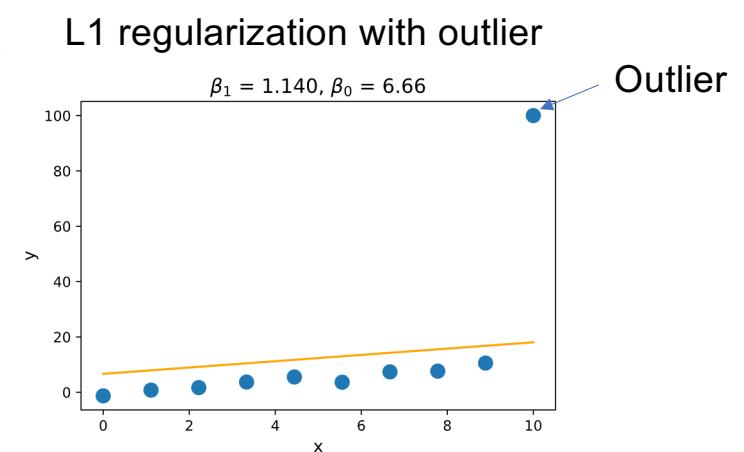
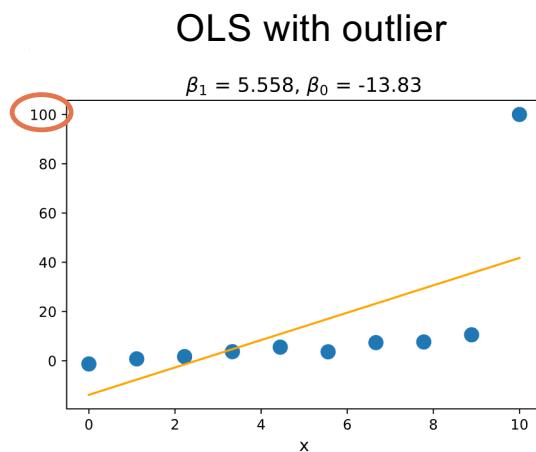
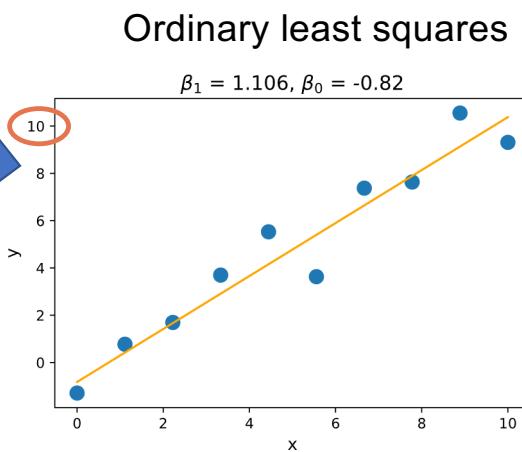
See <https://github.com/parrt/msds621/blob/master/notebooks/linear-models/regressor-regularization.ipynb>

Regularization premise

- **Extreme coefficients are unlikely to yield good generalization**
- So, we're simply going to constrain model coefficient magnitudes
- Same technique works for linear and logistic regression
(logistic is just a sigmoid of output of a linear model anyway)
- Preview: add L1 or L2 norm of β coefficient vector to loss function, so loss functions are now a function of known y , predicted y , **AND** model parameters

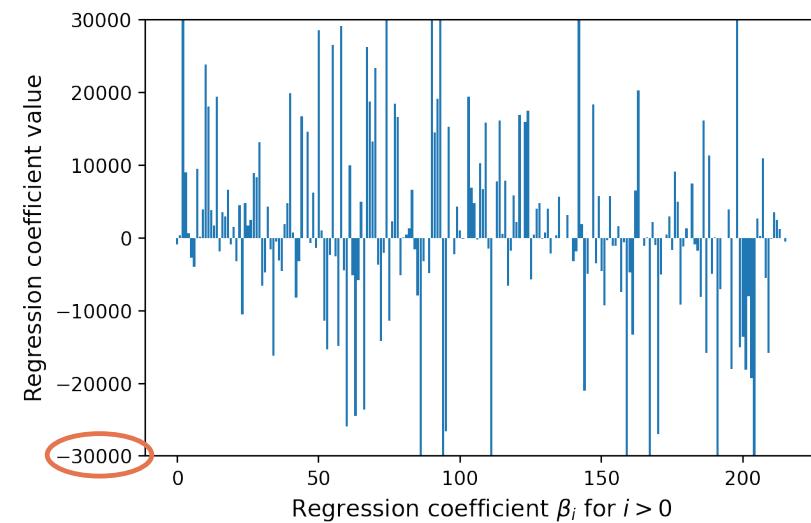
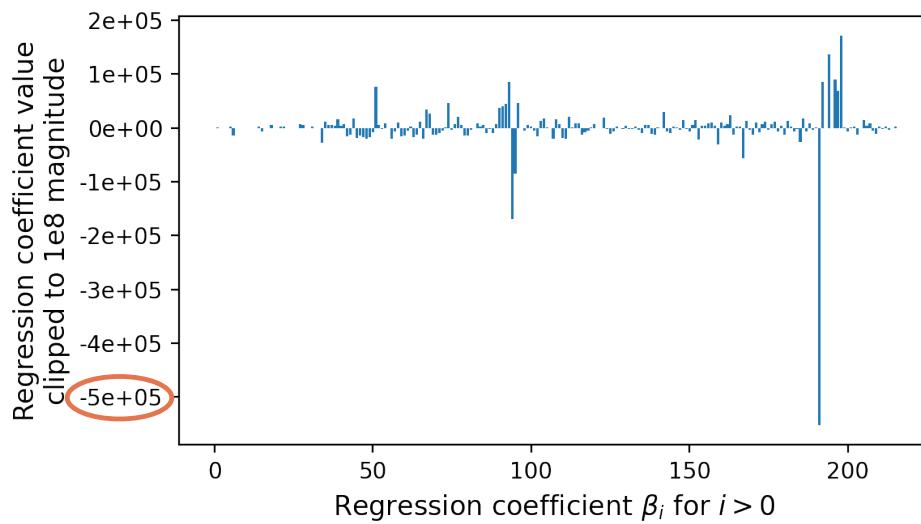
Let's make a deal!

- Let's trade some model bias (accuracy) for improved generality
- Consider an example of a simple data set with OLS fit
- Now, send $y[x==10]$ to 100; we get skewed line & bad R^2
- Regularization brings slope back down but with some bias



Ames housing data set (regressor)

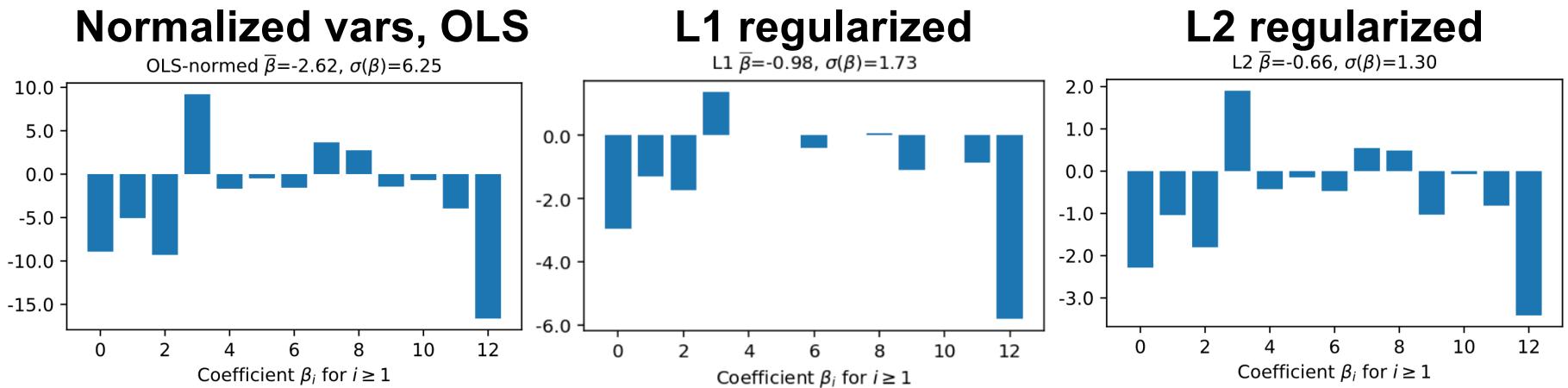
- With dummy vars, number of columns explodes from 81 to 216
- Compare scale of coeff (huge coefficients don't generalize)
- Regressor test R² is -3e20 w/o regularization and ~0.85 with L1



See <https://github.com/parrt/msds621/blob/master/notebooks/linear-models/regressor-regularization.ipynb>

Wine classifier less accurate w/o reg.

- Wine data set (130 records, 14 numeric vars)
- Test accur=.96 (normalized data) w/OLS
- L2 regularization 1.0 test score
- L1 regularization .96 but can drop 4 coefficients from model



See <https://github.com/parrt/msds621/blob/master/notebooks/linear-models/classifier-regularization.ipynb>

Example: classifier with many features

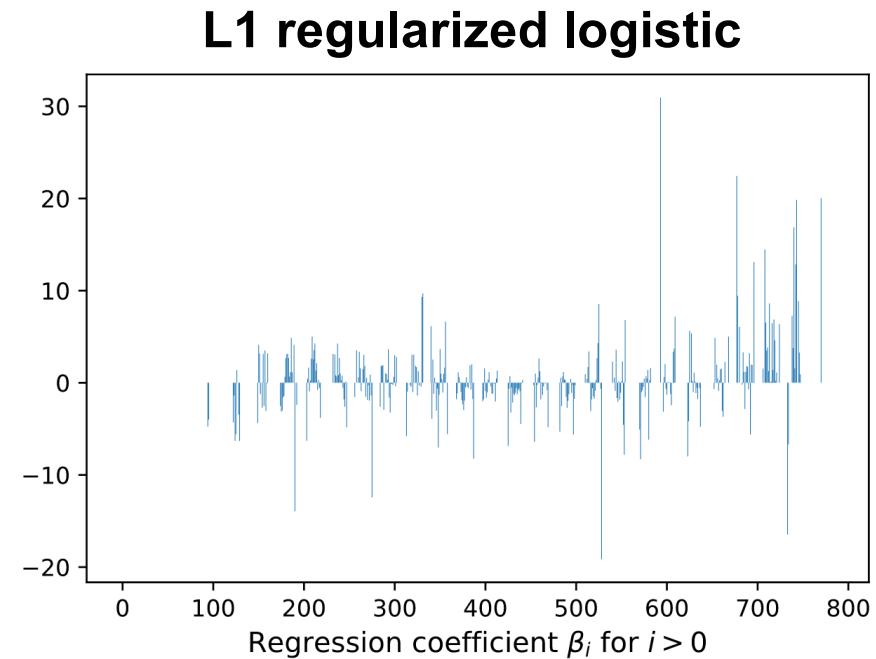
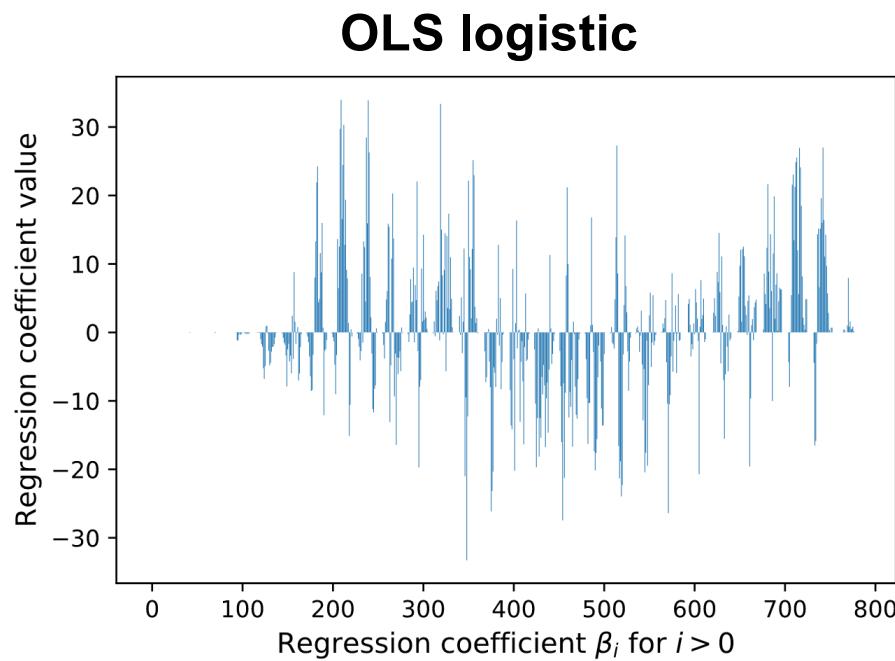
- Distinguish between ones and sevens (MNIST dataset)

MNIST sample	ones sample	sevens sample
3 8 6 9 6	1 1 1 1 1	7 7 7 7 7
4 5 3 8 4	1 1 1 1 1	7 7 7 7 7
5 2 3 8 4	1 1 1 1 1	7 7 7 7 7
8 1 5 0 5	1 1 1 1 1	7 7 7 7 7
9 7 4 1 0	1 1 1 1 1	7 7 7 7 7

See <https://github.com/parrt/msds621/blob/master/notebooks/linear-models/classifier-regularization.ipynb>

Compare coefficients w/o & with L1 reg

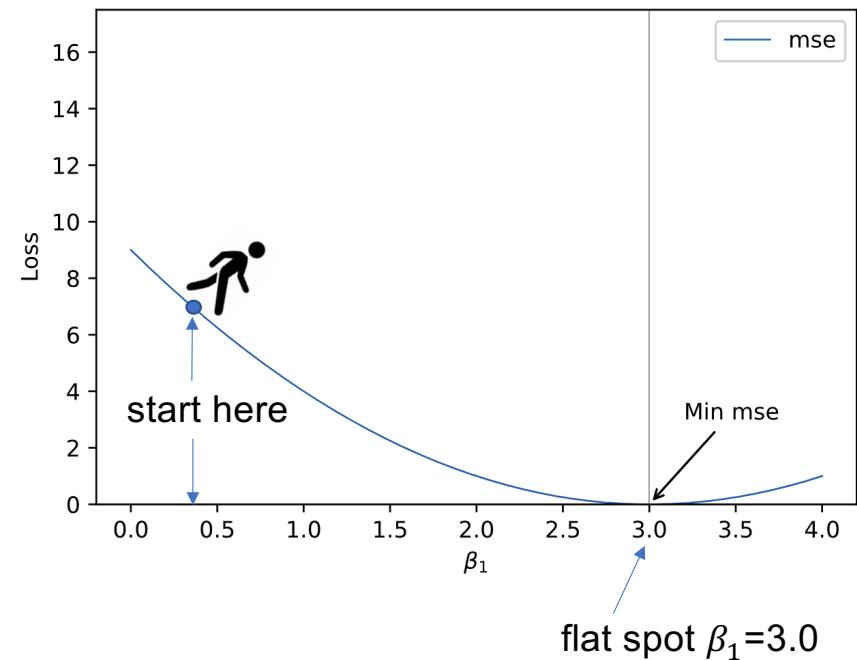
- Test accur=96% w/OLS and L1 but **log loss** improves a lot
- Also L1 regularization has zeroed out 423 coefficients and still accurate!



Quick detour:
Training models by walking
downhill and normalizing data

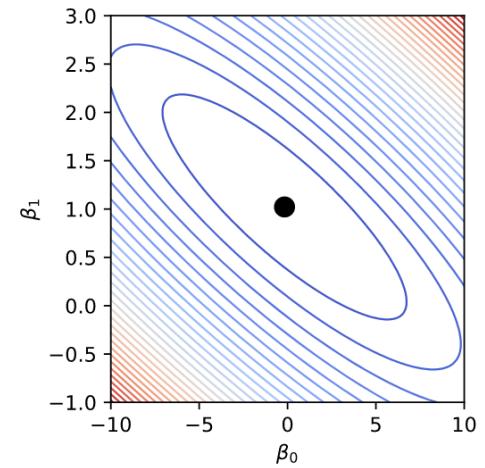
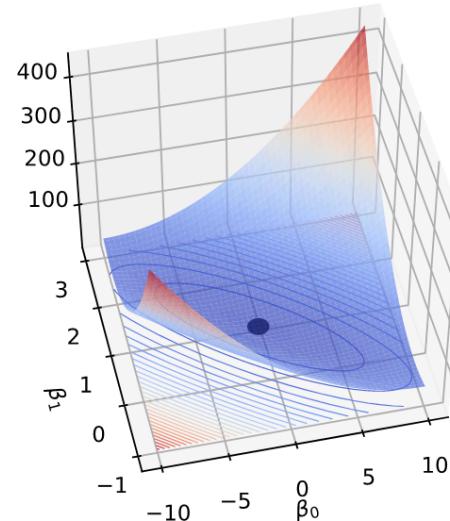
How training works in 1 dimension

- Loss function (cost) is a function of model predictions \hat{y} , known y values
- Minimize MSE loss computed on the **training set**
- Loss is a quadratic $(y - \hat{y})^2$ so pick a random starting point for β_1 then just walk β_1 downhill until you hit flat spot
- The derivative/slope of loss function is 0 at the minimum loss
- The β_1 at loss minimum is best fit coefficient for training set



Training in 2D; loss function shape

- We still walk downhill, but different directions can have different slopes
- Move search "particle" in 2D to find coefficients associated with minimum loss
- (We'll have full lecture on gradient descent, don't worry!)



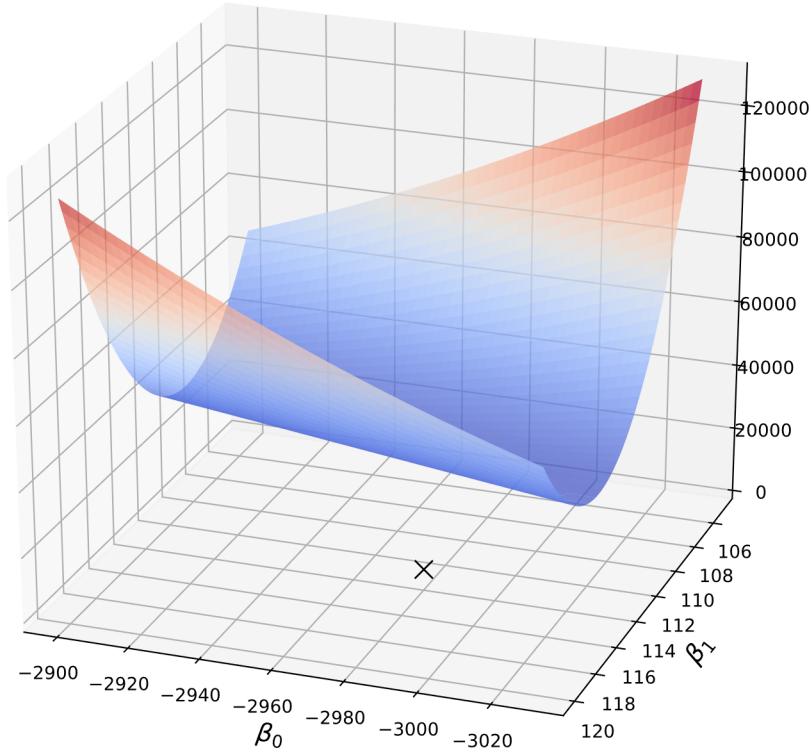
See <https://github.com/parrt/msds621/blob/master/notebooks/linear-models/viz-gradient-descent.ipynb>

For training linear models, always normalize/standardize data

- Regularization requires that you normalize your data set
- Zero center each variable and divide by the standard deviation

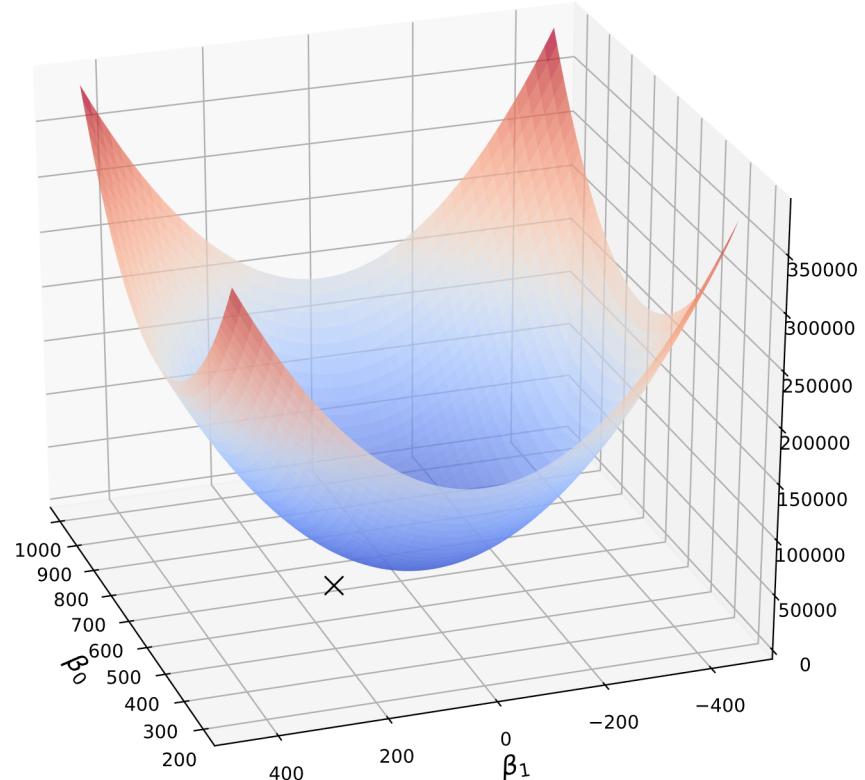
$$x^{(i)} = \frac{(x^{(i)} - \bar{x})}{\sigma_x}$$

2D loss function shape for normalized vars



Untouched variables

Cheese consumption vs bedsheets strangulation death data set



Normalized variables

See <https://github.com/parrt/msds621/blob/master/notebooks/linear-models/viz-gradient-descent.ipynb>

Standardization yields faster training

- Loss contours for normalized vars are less erratic due to nature of finite precision floating point; operations (like subtraction) on data in very different ranges are ill-conditioned
- Loss contours for normalized vars are more spherical and yield gradients that point more directly at min loss
- If x_1 is in $[0, 100]$ and x_2 in $[0, 0.001]$, then a **single** learning rate, η , that is small enough for x_2 (so we converge) is too slow for x_1

See <http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf>

Must standardize for regularized training

- We have just one λ for all coefficients so vars must be in same range or big β 's prevent regularization of small coefficients (big var ranges cause big β 's)
- Also β_0 is simply $\text{mean}(y)$ for L1/L2 linear regression if we standardize variables
- (We'll examine that regularization next)

$$\lambda \sum_{j=1}^p \beta_j^2$$

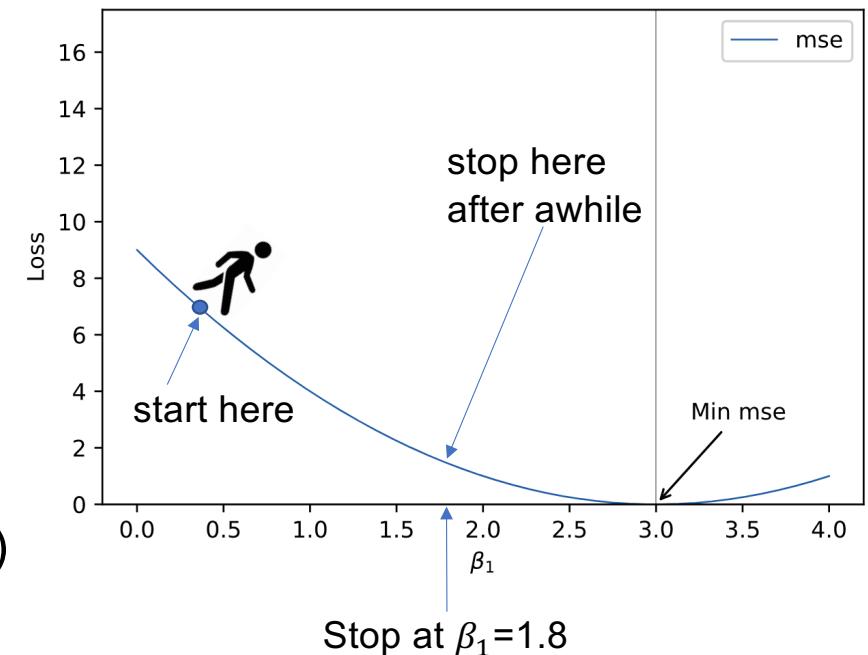
The regularization mechanism

The goal: Don't let optimization get too specific to training data; inhibit best fit

The cost: Sacrifice some model bias (underfit) for generality

Simplest regularization

- Just terminate minimization process early; don't let the coefficients fully reach the minimal loss location
- (Called *early stopping* in neural nets)
- Walk “downhill” on training set loss curve until either:
 - You run out of AWS CPU credits
 - After fixed amount of time (you're bored)
 - Loss on **validation set** (a moving avg thereof) starts going up, not down

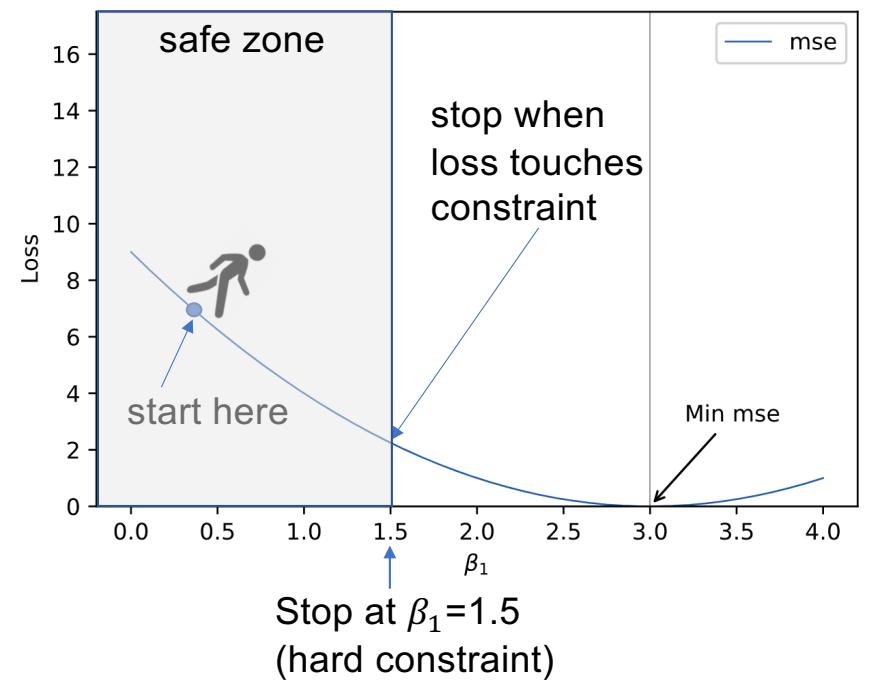


Improving early termination of training

- Stopping after fixed amount of time depends on how fast our “hiker” moves along loss curve and it’s hard to pick duration
- Instead, let’s just restrict magnitude of $\beta_1 < t$ for some t
- We agreed earlier that extreme β coefficients would likely be bad for generality
- Strategy: Pick initial β_1 in $[0, t)$ for some max t and go downhill and stop at minimum or when $\beta_1 \geq t$

Minimizing loss, subject to a *hard constraint*

- Let $t=1.5$, which defines “safe zone”
- Stop at min loss point or $\beta_1 = t$
- The “best” coefficient is where constraint t and loss function meet, if min loss is outside safe zone
- If min loss is in safe zone, then regularization constraint wasn’t needed (same as OLS)



Minimizing in 2D subject to hard constraint

- Spinning line segment $[0,t]$ around origin 360° , gives a circle
- Recall formula for circle; constrain β_i such that $\beta_1^2 + \beta_2^2 < t$ where t is radius squared
- The “best” coefficient is min loss function **on constraint curve** (if loss min outside zone)
- If loss min inside radius, same as OLS
- Pick initial $[\beta_1, \beta_2]$ inside safe zone, then start walking downhill, stop at min loss or edge of the safe zone
- **This is L2 (Ridge) regularization**
$$\sum_{j=1}^p \beta_j^2 < t$$

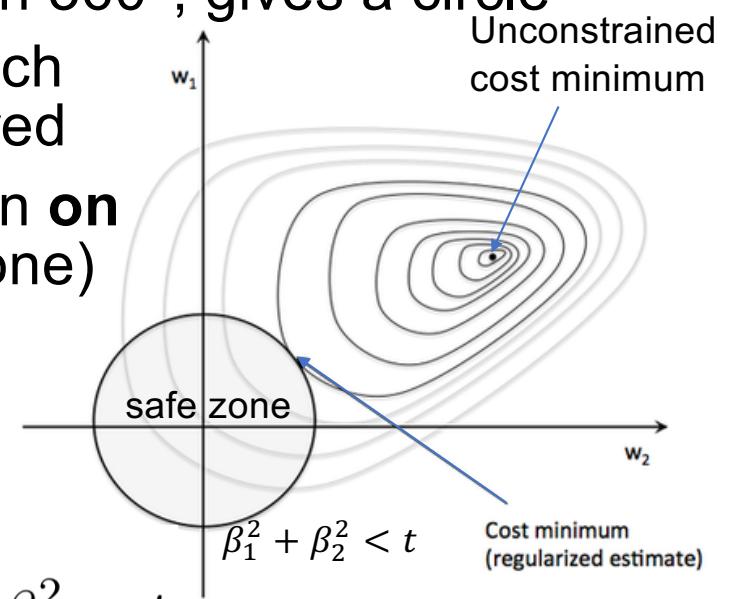
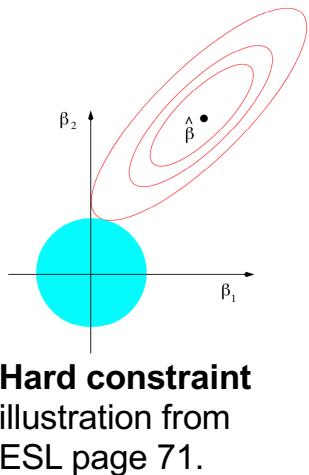
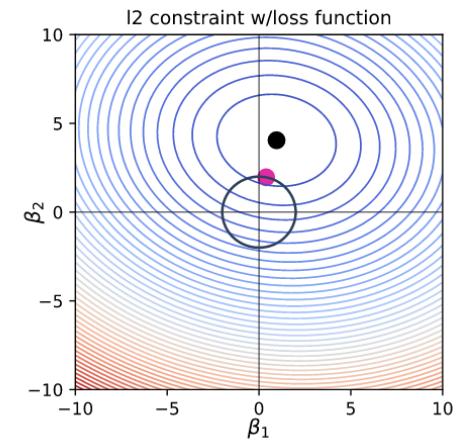
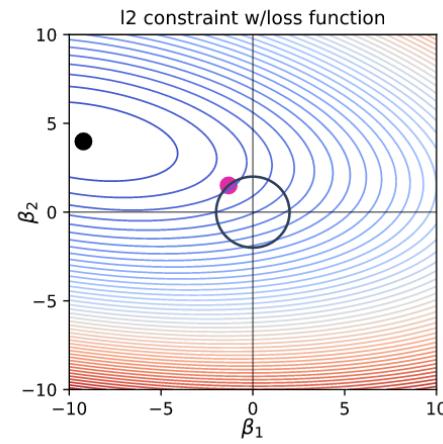
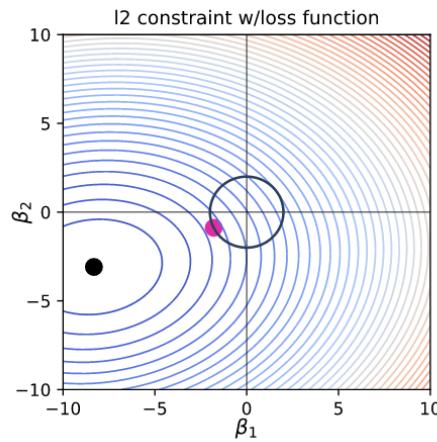
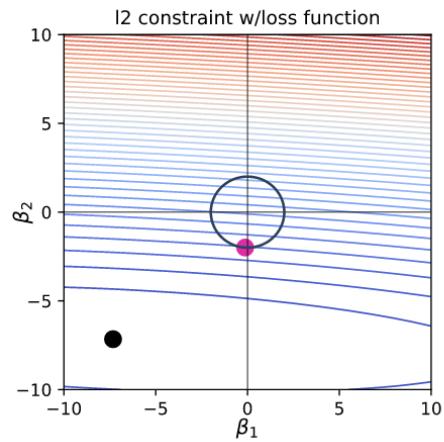


Image credit: <https://www.kdnuggets.com/2016/06/regularization-logistic-regression.html>

L2 regularization examples

- The L2 regularized coefficients sit on the L2 boundary circle where the loss function has the minimum value.
- Walk around the circle and identify the location with the minimum loss function value

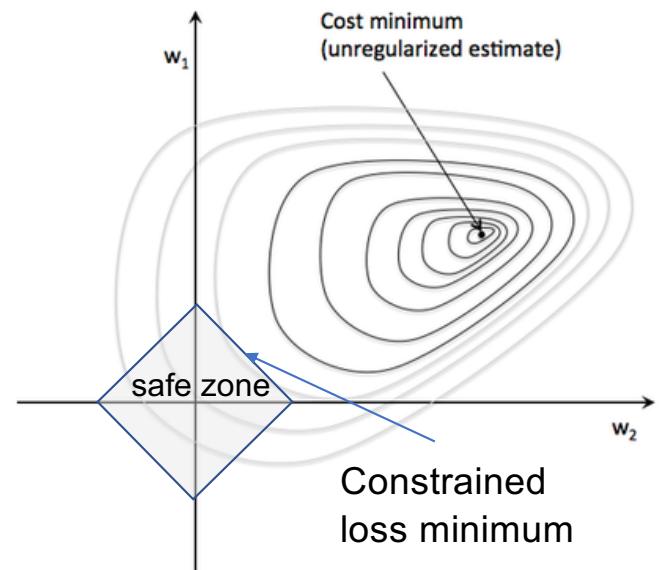


Minimizing L1 in 2D (hard constraint)

- Instead, we can sum $|\beta_i|$'s not β_i^2 's, giving diamond-shaped safe zone
- **This is L1 (Lasso) regularization**

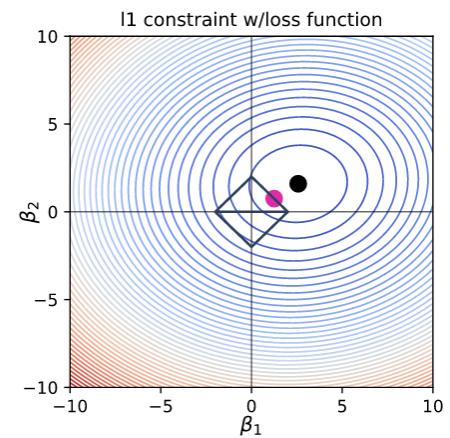
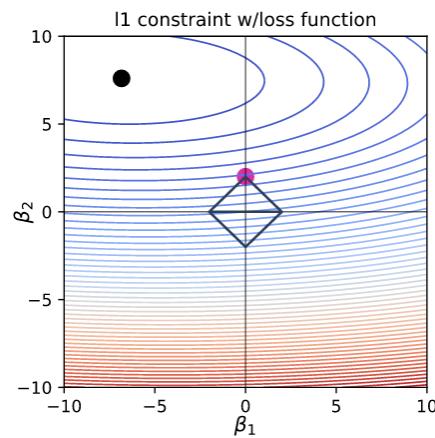
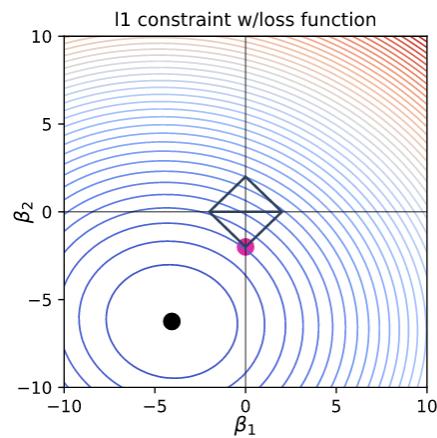
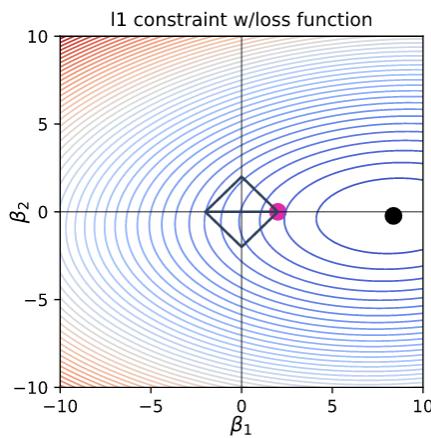
$$\sum_{j=1}^p |\beta_j| < t$$

- Notice how statisticians have this backwards; L1 constraint zone looks like a ridge not lasso, and L2 safe zone (circle) looks like a lasso



L1 regularization examples

- The L1 regularized coefficients sit on the L1 boundary diamond where the loss function has the minimum value.
- Walk around the diamond and identify the location with the minimum loss function value



Please note how often the red dot sits at a diamond corner, with a $\beta_i = 0$

Fitting regularized linear model (Conceptually)

- Minimize the usual MSE loss function:

$$\mathcal{L}(\beta) = \sum_{i=1}^n (y^{(i)} - (\mathbf{x}'^{(i)} \cdot \beta))^2$$

- **Subject to** either (L2 or L1):

$$\sum_{j=1}^p \beta_j^2 < t \quad \text{or} \quad \sum_{j=1}^p |\beta_j| < t$$

- Problem is “**subject to**” constraint is tough to implement

How we *actually* fit regularized models

- Minimizing loss function subject to a constraint is more complicated to implement than just function minimization so...
- Invoke magic of Lagrange multipliers:

$$\text{For L2: } \mathcal{L}(\beta) + \lambda \sum_{j=1}^p \beta_j^2$$

$$\text{For L1: } \mathcal{L}(\beta) + \lambda \sum_{j=1}^p |\beta_j|$$

- Drop t since t is constant & doesn't affect minimizing
- This is a **soft constraint** and penalty increases as β_i 's move away from origin; *there's no hard cutoff!*
- Net effect is that regularization pulls min loss location closer to origin!

Detour: Lagrange Multipliers

- Magic of Lagrange multipliers lets us incorporate constraint into loss function:

$$\mathcal{L}(\beta) \text{ s.t. } \sum_{j=1}^p \beta_j^2 \leq t \text{ same as}$$

$$\mathcal{L}(\beta, \lambda) = \mathcal{L}(\beta) + \lambda \left(\sum_{j=1}^p \beta_j^2 - t \right) \text{ for some } \lambda$$

(λ and t are related one-to-one per ESLII book
but by no relationship I can find)

Fitting β_i 's and picking hyperparameter λ

- λ is unknown like t but at least we have a single loss function now
- Find λ by finding minimum loss for different λ values, pick λ that gets min loss on validation set (more on these sets later)

$$\mathcal{L}(\beta) + \lambda \sum_{j=1}^p \beta_j^2$$

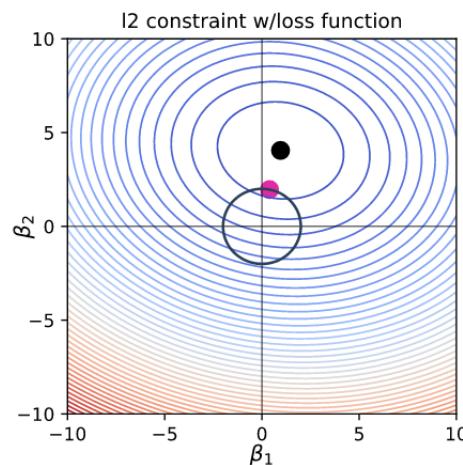
$$\mathcal{L}(\beta) + \lambda \sum_{j=1}^p |\beta_j|$$

- β_0 is just \bar{y} , assuming zero-centered data set
- **Note:** β_0 is NOT included in penalty

Hard vs soft constraints in 2D

Hard constraint

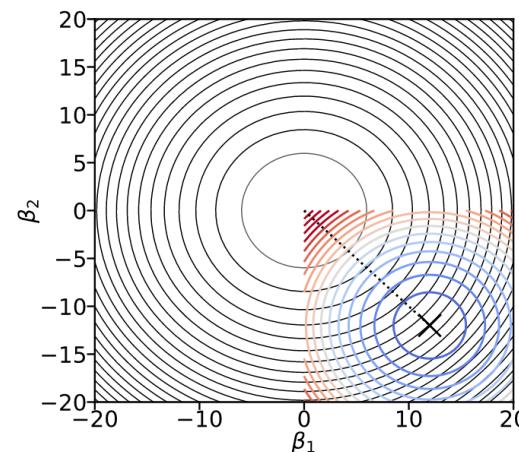
$$\mathcal{L}(\beta) \text{ s.t. } \sum_{j=1}^p \beta_j^2 \leq t$$



Constrained loss function

Soft constraint

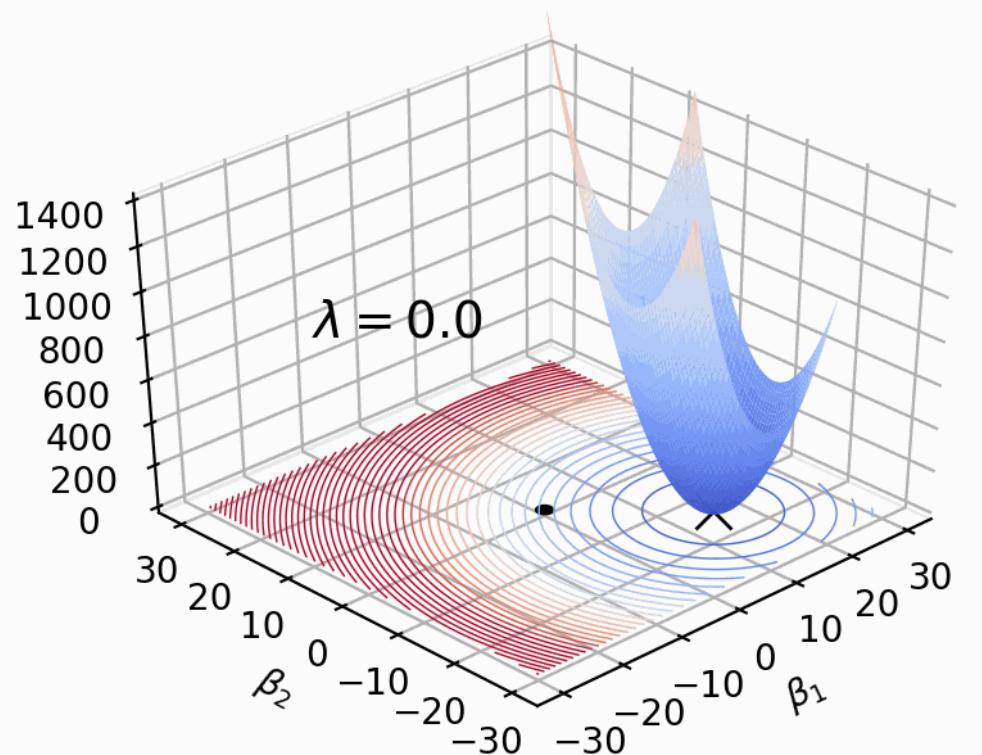
$$\mathcal{L}(\beta) + \lambda \sum_{j=1}^p \beta_j^2$$



Sum of loss and penalty functions

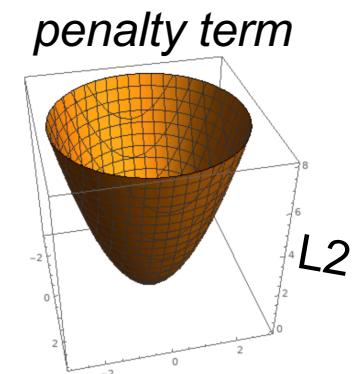
Affect of λ on combined loss function

- MSE loss function + soft constraint as λ varies from 0 to 6
- Same training data so MSE loss surface is the same
- Only λ is changing here, increasing penalty for same β_i coefficients
- Combined loss function moves upwards and towards the origin



How penalty term restricts β_i 's

- Think of regularization as two different cost functions, MSE and regularization, added together
- Regularization penalty term **increases loss** but skews min loss β_i location towards origin as penalty curve is anchored at origin
- Soft constraint makes larger β_i very unlikely due to increased penalty away from origin (hard constraint makes large coefficients $> t$ impossible)
- L1&L2 **shift β_i 's towards 0** because penalty approaches 0 only at $\beta=0$



The effect of regularization

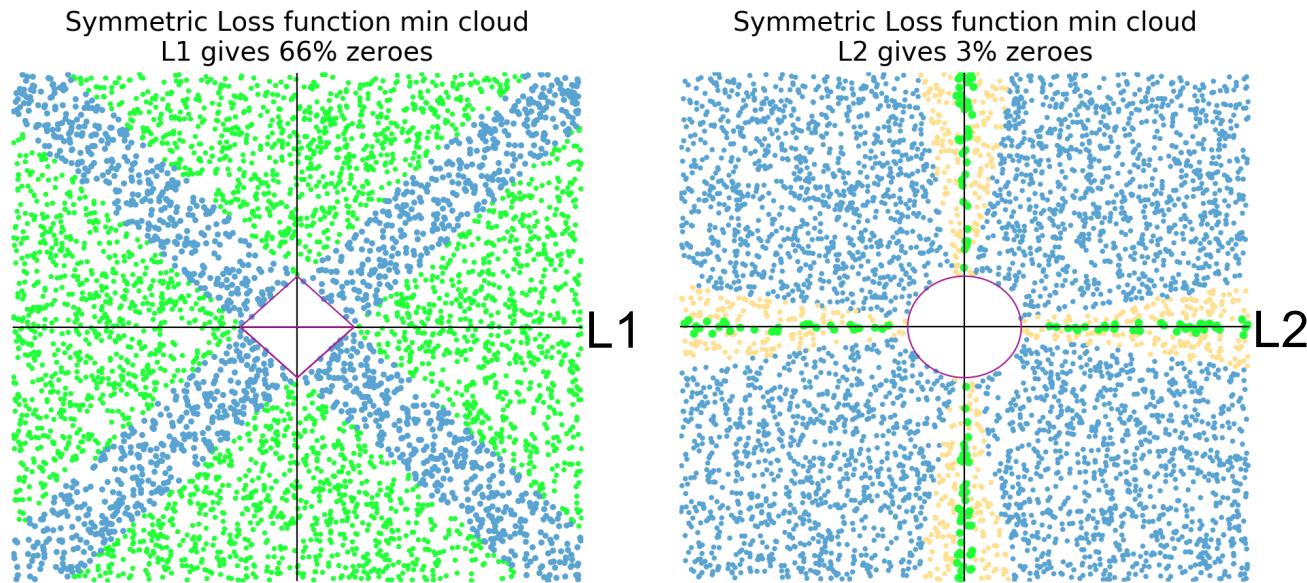
- What happens when λ is 0? Regularization is turned off
- What happens when we crank up λ ? Loss function strives for small β
- L1 tends to shrink coefficients to zero; useful for feature selection since we can drop features with zero coefficients
- L2 tends to shrink coefficients evenly; discourages any single β_i from getting much bigger than the others
- L2 useful when you have collinear/codependent features
 - e.g., **gender** and **ispregnant**
 - codependence tends to increase coefficient variance, making coefficients unreliable/unstable, which hurts model generality
 - L2 reduces the variance of these estimates,

Comparing L1 and L2 regularization

L1 regularization encourages coefficients=0

- Simulate random symmetric two-variable quadratic loss functions at random locations

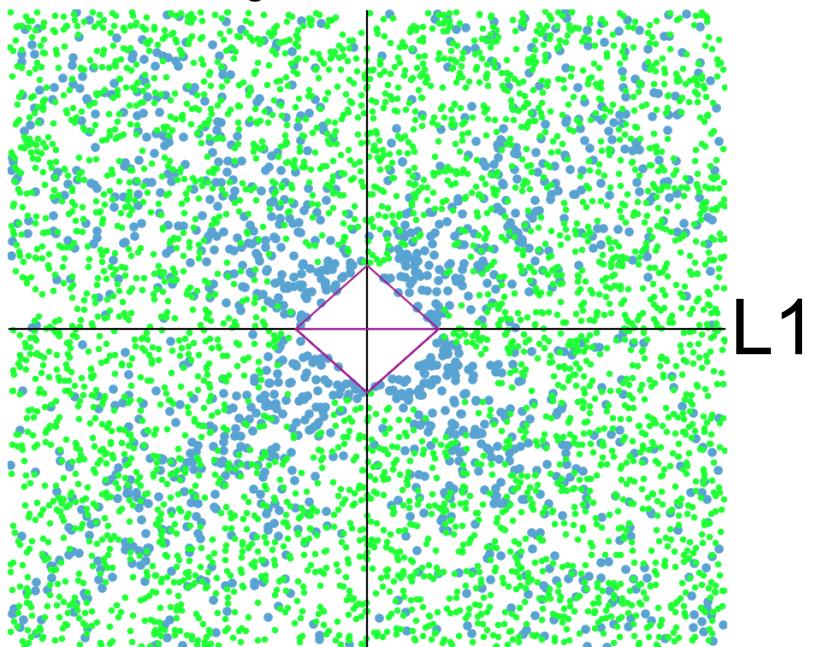
Dots: loss function minimum locations; green: a zero coefficient, blue no zeros, orange a coeff close to 0



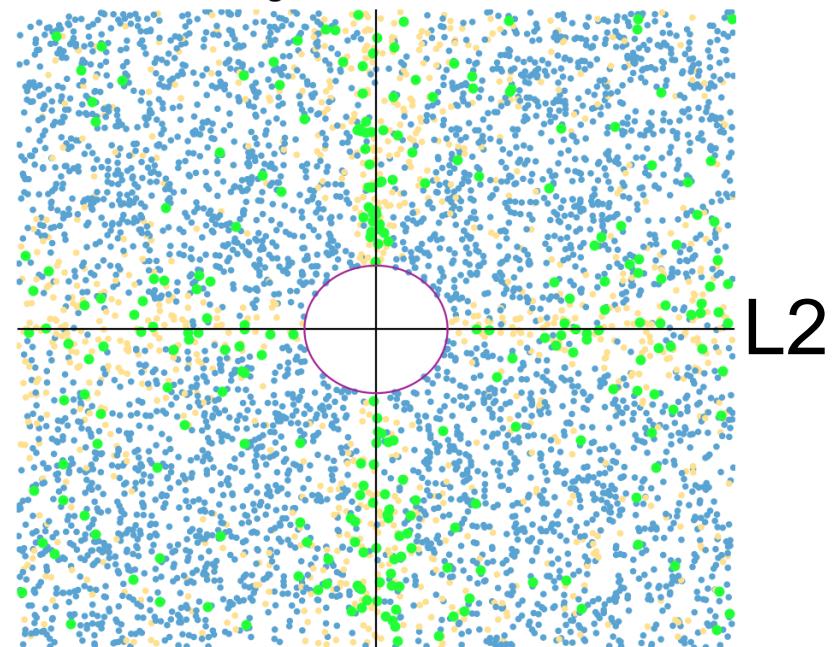
L1 has many more zero parameters than L2 for symmetric loss functions

Random asymmetric, angled loss functions

Loss function min cloud
L1 gives 72% zeroes



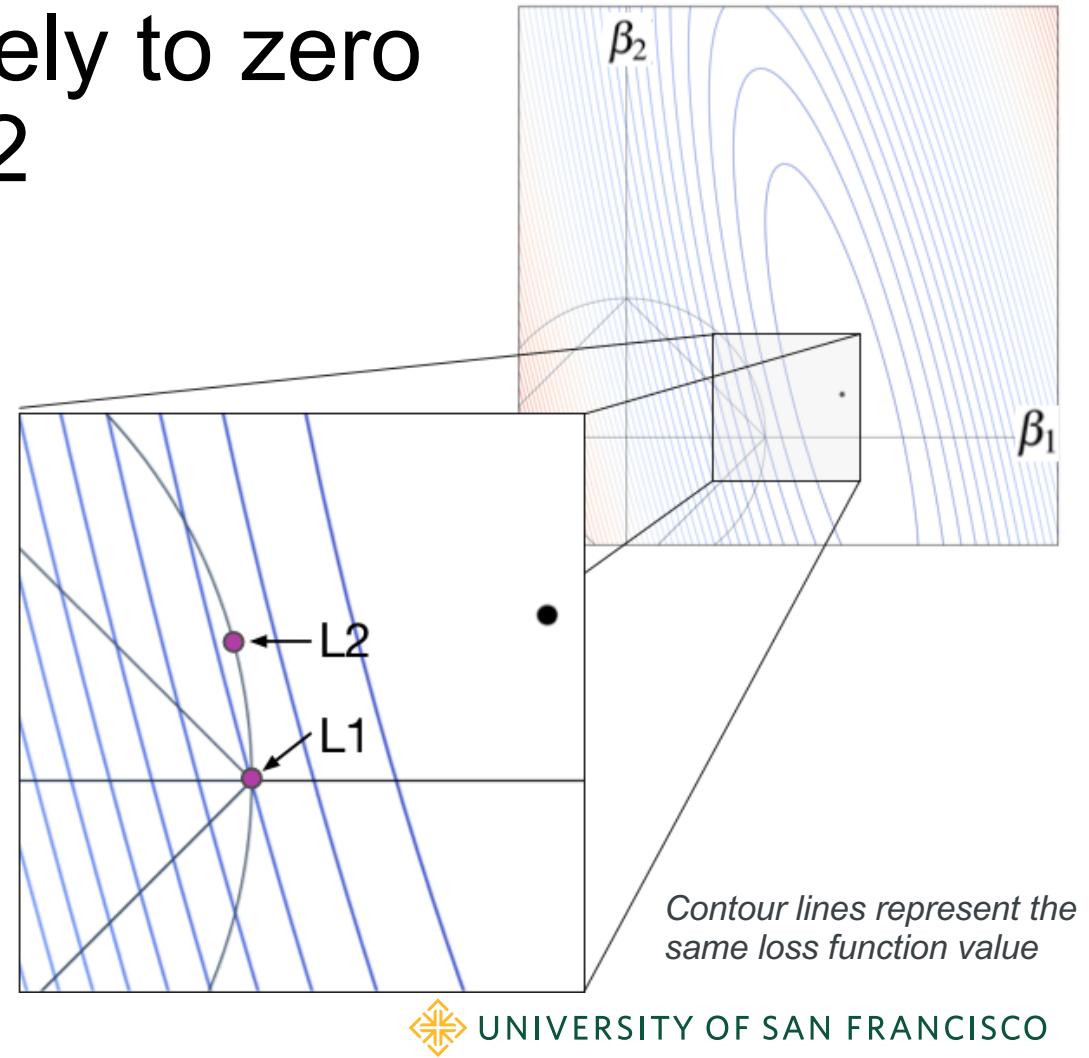
Loss function min cloud
L2 gives 5% zeroes



L1 still has many more zero parameters than L2 for arbitrary loss functions

Why L1 is more likely to zero coefficients than L2

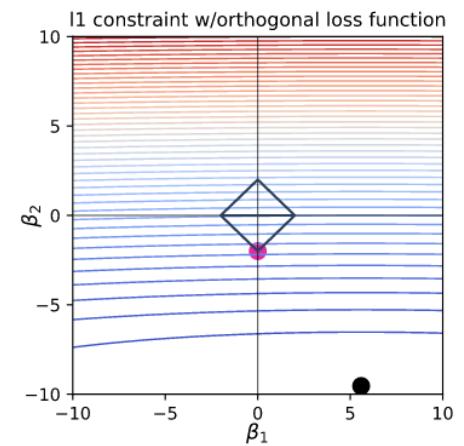
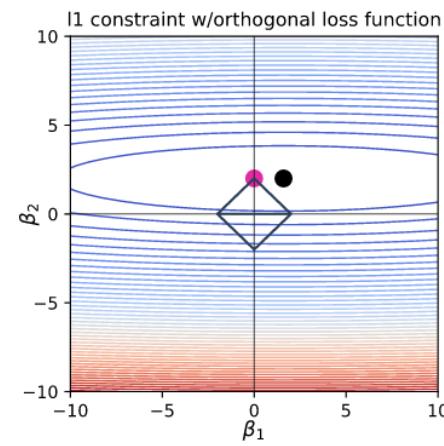
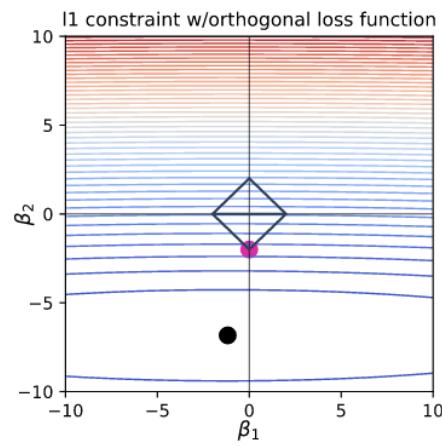
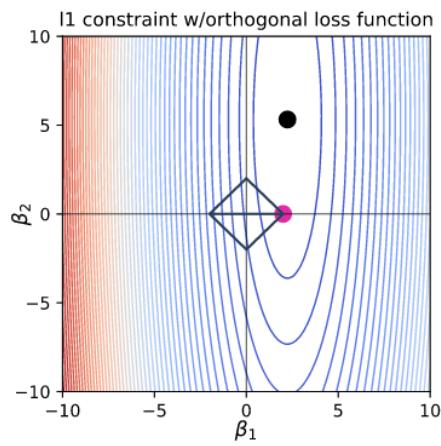
- Moving away from the L1 diamond point immediately increases loss
- L2 can move upwards a bit before moving leftward away from loss function minimum
- As black dot approaches β_1 axis, L2 purple dot approaches β_1 axis, but L1 gets $\beta_2 = 0$ even when loss min location far from axis



UNIVERSITY OF SAN FRANCISCO

L1 regularization strongly encourages zero coefficients for less predictive features

- Imagine one of two features is very important and the other isn't
- Implies loss function looks like a taco shell or canoe, and at or close to 90 degrees to one of the axes
- Movement in one direction can be much more expensive than movement in the other



Lab time

- Exploring regularization for linear regression

<https://github.com/parrt/msds621/tree/master/labs/linear-models/regularization-regr.ipynb>

Regularized logistic regression

Optimizing likelihood with penalty terms

(Logistic regularization is **not** required for your project, but L1 is an option)

- Same mechanism: minimizing (**negation** of maximum likelihood) via Lagrangian interpretation:

$$\mathcal{L}(\beta, \lambda) = - \sum_{i=1}^n \left\{ y^{(i)} \mathbf{x}'^{(i)} \beta - \log(1 + e^{\mathbf{x}' \beta}) \right\} + \lambda \sum_{j=1}^p \beta_j^2$$

$$\mathcal{L}(\beta, \lambda) = - \sum_{i=1}^n \left\{ y^{(i)} \mathbf{x}'^{(i)} \beta - \log(1 + e^{\mathbf{x}' \beta}) \right\} + \lambda \sum_{j=1}^p |\beta_j|$$

Note: β_0 is NOT included in penalty, but is used in $\mathcal{L}(\beta)$ term

Fitting L1 logistic regression β_i 's and picking λ

- ESLII book p125 says to find L1 β_0 and $\beta_{1..p}$ that maximize:

$$\max_{\beta_0, \beta} \left\{ \sum_{i=1}^N \left[y_i(\beta_0 + \beta^T x_i) - \log(1 + e^{\beta_0 + \beta^T x_i}) \right] - \lambda \sum_{j=1}^p |\beta_j| \right\}. \quad (4.31)$$

- We minimize the negative of that to find β of max likelihood
- Means we must find β_0 differently than for $\beta_{1..p}$
- As before, find λ by computing minimum loss for different λ values, pick λ that gets min loss on validation set
- L1 Logistic gradient is tricky to get right; see gradient descent lecture

L1 Logistic gradient is tricky to get right

(See derivation of L1 gradients in appendix of project description)

Algorithm: $L1NegLogLikelihood(\mathbf{X}', \mathbf{y}, \beta')$

$$err = \mathbf{y} - \sigma(\mathbf{X}' \cdot \beta')$$

(error vector is $n \times 1$ column vector)

$$\frac{\partial}{\partial \beta_0} = mean(err)$$

(usual log-likelihood gradient; use current β')

$$r = \lambda \text{sign}(\beta')$$

(regularization term $p + 1 \times 1$ column vector)

$$r[0] = 0$$

(kill β_0 position but keep as $p + 1 \times 1$ vector)

$$\nabla = \frac{1}{n} \left\{ \mathbf{X}'^T err - r \right\}$$

$$\text{return} - \begin{bmatrix} \frac{\partial}{\partial \beta_0} \\ \nabla_1 \\ \vdots \\ \nabla_p \end{bmatrix}$$

$$\frac{\partial}{\partial \beta_0} \mathcal{L}(\beta, \lambda) = mean(\mathbf{y} - \sigma(\mathbf{X}' \cdot \beta'))$$

gradients

$$\nabla_{\beta_{1..p}} \mathcal{L}(\beta, \lambda) = \frac{1}{n} \left\{ \mathbf{X}^T (\mathbf{y} - \sigma(\mathbf{X}' \cdot \beta')) - \lambda \text{sign}(\beta) \right\}$$



UNIVERSITY OF SAN FRANCISCO

Resources with code

- <https://aimotion.blogspot.com/2011/11/machine-learning-with-python-logistic.html>
- <https://stackoverflow.com/questions/38853370/matlab-regularized-logistic-regression-how-to-compute-gradient>
- You may look at but not cut-paste from these examples for your project (and I think there are some bugs too)

Key takeaways

- Regularization increases generality at cost of some bias
- Does so by restricting size of coefficients with hard constraint (conceptually) or soft constraint using penalty term (impl.)
- For hard constraint, min loss is inside safe zone or on zone border
- Soft constraint penalty discourages bigger parameters
- L2 discourages any β_i from getting much bigger than the others
- L1 encourages zero β_i ; in practice, we see L1 zero out unpredictive vars

Key implementation details

- Minimize $\mathcal{L}(\beta, \lambda)$ by trying multiple λ and choosing parameters from fitted model getting lowest **validation** error
- Always normalize numeric variables in X before fitting models
- If 0-centered x_i then $\beta_0 = \text{mean}(y)$ for L1 and L2 regression
- Logistic regression has no closed form for β_0
- OLS & L2 regularized linear regression have symbolic solutions
- L1 linear regression and L1/L2 logistic regression require iterative solution: usually some gradient descent variation

Interview questions (L1 vs L2)

- Both L1 and L2 increase bias (reduce "accuracy" of fit, predictions)
- L1 useful for variable / feature selection as some β_i go to 0
- L2 much less likely to get zero coefficients
- L2 useful when you have collinear features (e.g., both tumor radius and tumor circumference features)
 - Collinearity increases coefficient variance, making them unreliable
 - And L2 reduces variance of β_i estimate which counteracts effect of collinearity
- With regularization, we don't get unbiased β_i estimates
 - Expected value of β_i estimate is no longer β_i since we've constrained β_i 's
 - Consequence: we no longer know effect of x_i on target y via β_i
- In my experiments, regularization improves regression much more than classification in terms of R^2 and simple accuracy; (due to sigmoid constraining the solutions to the loss function?)

Lab time

- Regularization for logistic regression

<https://github.com/parrt/msds621/blob/master/labs/linear-models/regularization-logi.ipynb>