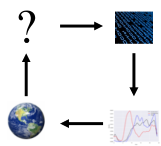
Fitting Linear Models, Regularization (revisited) & Cross Validation

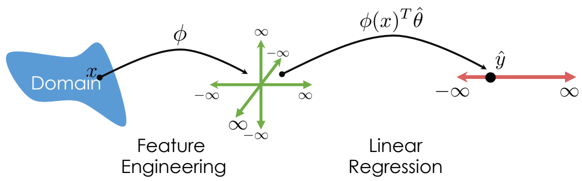
Slides by:

Joseph E. Gonzalez jegonzal@cs.berkeley.edu



Previously

Feature Engineering and Linear Regression



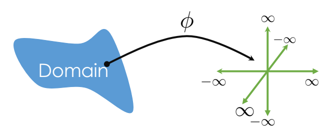
Recap: *Feature Engineering*

- Linear models with feature functions:

$$f_\theta(x) = \sum_{j=1}^{d} \theta_j \phi_j(x)$$

- Feature Functions: $\phi : \mathcal{X} \rightarrow \mathbb{R}^d$

Notation: Computer scientist / ML researchers tend to you *d* (dimensions) and statisticians will use *p* (parameters).



- **One-hot encoding:** *Categorical Data*

state
NY
WA
CA

AL	...	CA	...	NY	...	WA	...	WY
0	...	0	...	1	...	0	...	0
0	...	0	...	0	...	1	...	0
0	...	1	...	0	...	0	...	0

- **Bag-of-words & N-gram:** *Text Data*

"Learning about machine learning is fun."

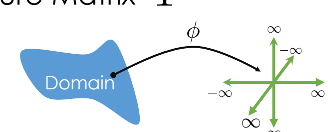
aardvark	aardwolf		fun		learning		machine		zyzzyva
0	0	...	1	...	2	...	1	...	0

Vector

- **Custom Features:** *Domain Knowledge*

$$\phi(\text{lat}, \text{lon}, \text{amount}) = \frac{\text{amount}}{\textbf{Stores}[\textbf{ZipCode}[\text{lat}, \text{lon}]]}$$

The Feature Matrix Φ



X DataFrame

uid	age	state	hasBought	review
0	32	NY	True	"Meh."
42	50	WA	True	"Worked out of the box ..."
57	16	CA	NULL	"Hella tots lit..."

$\xrightarrow{\phi}$

$\Phi \in \mathbb{R}^{n \times d}$

AK	...	NY	...	WY	age	hasBought	hasBought missing
0	...	1	...	0	32	1	0
0	...	0	...	0	50	1	0
0	...	0	...	0	16	0	1

Entirely **Quantitative** Values

X DataFrame

uid	age	state	hasBought	review
0	32	NY	True	"Meh."
42	50	WA	True	"Worked out of the box …"
57	16	CA	NULL	"Hella tots lit…"

$\xrightarrow[\phi]{}$

$\Phi \in \mathbb{R}^{n \times d}$

AK	…	NY	…	WY	age	hasBought	hasBought missing
0	…	1	…	0	32	1	0
0	…	0	…	0	50	1	0
0	…	0	…	0	16	0	1

Entirely **Quantitative** Values

Another quick note on confusing notation.

In many textbooks and even in the class notes and discussion you will see:

$$X \in \mathbb{R}^{n \times d} \quad \text{and} \quad \hat{\theta} = \left(X^T X\right)^{-1} X^T Y$$

In this case we are assuming ***X*** is the transformed data **Φ**. This can be easier to read but hides the feature transformation process.

Capital Letter: *Matrix* or *Random Variable?*

- Both tend to be capitalized
- Unfortunately, there is no common convention … you will have to use context.

The Feature Matrix Φ

AK	…	NY	…	WY	age	hasBought	hasBought missing
0	…	1	…	0	32	1	0
0	…	0	…	0	50	1	0
0	…	0	…	0	16	0	1

Entirely **Quantitative** Values

$$\Phi \in \mathbb{R}^{n \times d} = \phi\left(\underset{\text{DataFrame}}{X}\right) = \underbrace{n\left\{\begin{bmatrix} \text{---}\ \phi\left(x^{(1)}\right)\ \text{---} \\ \text{---}\ \phi\left(x^{(2)}\right)\ \text{---} \\ \cdots \\ \text{---}\ \phi\left(x^{(n)}\right)\ \text{---} \end{bmatrix}\right.}_{d}$$

Rows of the **Φ** matrix correspond to records.

Columns of the **Φ** matrix correspond to features.

Confusing notation!

The Feature Matrix Φ

AK	…	NY	…	WY	age	hasBought	hasBought missing
0	…	1	…	0	32	1	0
0	…	0	…	0	50	1	0
0	…	0	…	0	16	0	1

Entirely **Quantitative** Values

$$\Phi \in \mathbb{R}^{n \times d} = \phi\left(\underset{\text{DataFrame}}{X}\right) = \underbrace{n\left\{\begin{bmatrix} \text{---}\ \phi\left(X_{1,\bullet}\right)\ \text{---} \\ \text{---}\ \phi\left(X_{2,\bullet}\right)\ \text{---} \\ \cdots \\ \text{---}\ \phi\left(X_{n,\bullet}\right)\ \text{---} \end{bmatrix}\right.}_{d}$$

Rows of the **Φ** matrix correspond to records.

Columns of the **Φ** matrix correspond to features.

Notation Guide

$A_{i,\bullet}$: row i of matrix A.

$A_{\bullet,j}$: column j of matrix A.

Making Predictions

$$\Phi \in \mathbb{R}^{n \times d} = \phi\left(\underset{\text{DataFrame}}{X}\right) = \underbrace{n\left\{\begin{bmatrix} \text{---}\ \phi\left(X_{1,\bullet}\right)\ \text{---} \\ \text{---}\ \phi\left(X_{2,\bullet}\right)\ \text{---} \\ \cdots \\ \text{---}\ \phi\left(X_{n,\bullet}\right)\ \text{---} \end{bmatrix}\right.}_{d}$$

Rows of the **Φ** matrix correspond to records.

Columns of the **Φ** matrix correspond to features.

<u>Prediction</u>

$$\hat{Y} = f_{\hat{\theta}}(X) = \Phi\hat{\theta} = \underbrace{n\left\{\begin{bmatrix} \text{---}\ \phi\left(X_{1,\bullet}\right)\ \text{---} \\ \text{---}\ \phi\left(X_{2,\bullet}\right)\ \text{---} \\ \cdots \\ \text{---}\ \phi\left(X_{n,\bullet}\right)\ \text{---} \end{bmatrix}\right.}_{d} \begin{bmatrix} \mid \\ \hat{\theta} \\ \mid \end{bmatrix} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \cdots \\ \hat{y}^{(n)} \end{bmatrix}$$

Normal Equations

- Solution to the least squares model:

$$\hat{\theta} = \arg\min \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{d} \theta_j \phi_j(x_i) \right)^2$$

- Given by the normal equation:

$$\hat{\theta} = \left(\Phi^T \Phi\right)^{-1} \Phi^T Y$$

- **You should know this!**
- You do not need to know the calculus based derivation.
- **You should know the geometric derivation …**

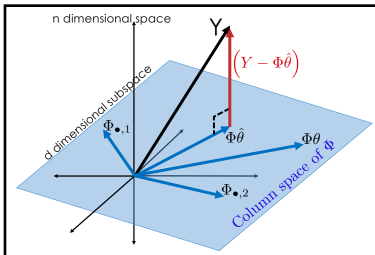
Geometric Derivation: <u>**Not**</u> **Bonus Material**

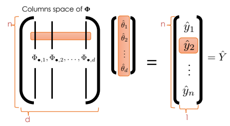
We decided that this is too exciting to not know.

- Examine the column spaces:

Columns space of **Φ**

$$\underbrace{n\left\{\begin{bmatrix} \mid & \mid & & \mid \\ \Phi_{\bullet,1}, & \Phi_{\bullet,2}, & \ldots, & \Phi_{\bullet,d} \\ \mid & \mid & & \mid \end{bmatrix}\right.}_{d} \begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ \vdots \\ \hat{\theta}_d \end{bmatrix} = \underbrace{n\left\{\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix}\right.}_{1} = \hat{Y}$$





Definition of orthogonality

$$0 = \Phi^T (Y - \Phi\hat{\theta})$$

Derivation

$$0 = \Phi^T \left(Y - \Phi\hat{\theta}\right)$$

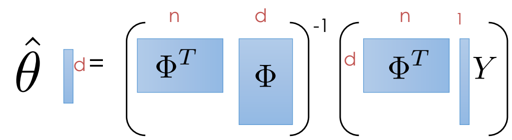
$$0 = \Phi^T Y - \Phi^T \Phi\hat{\theta}$$

$$\Phi^T \Phi\hat{\theta} = \Phi^T Y$$

$$\hat{\theta} = \left(\Phi^T \Phi\right)^{-1} \Phi^T Y$$

"Normal Equation"

The Normal Equation $\hat{\theta} = \left(\Phi^T \Phi\right)^{-1} \Phi^T Y$



Note: For inverse to exist **Φ** needs to be full column rank.

→ cannot have co-linear features

This can be addressed by adding regularization …

In practice we will use regression software (e.g., scikit-learn) to estimate θ

Least Squares Regression in Practice

- Use optimized software packages
  - Address numerical issues with matrix inversion
- Incorporate some form of regularization
  - Address issues of collinearity
  - Produce more robust models
- We will be using scikit-learn:
  - http://scikit-learn.org/stable/modules/linear_model.html
  - See Homework 6 for details!

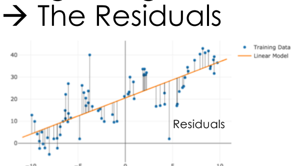
Scikit Learn Models

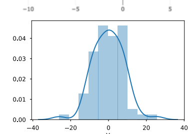
- Scikit Learn has a wide range of models
- Many of the models follow a common pattern:

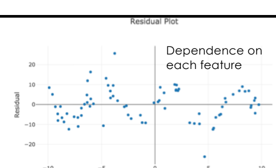
Ordinary Least Squares Regression

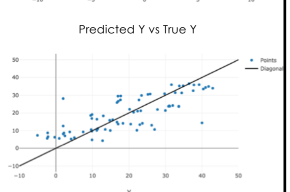
```
from sklearn import linear_model
f = linear_model.LinearRegression(fit_intercept=True)
f.fit(train_data[['X']], train_data['Y'])
Yhat = f.predict(test_data[['X']])
```

Diagnosing Fit → The Residuals





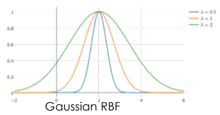


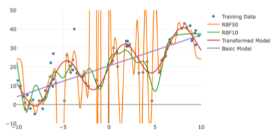


Notebook Demo

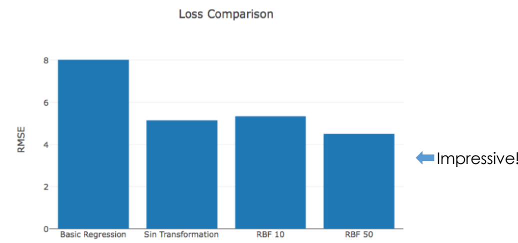
- **Generic Features:** *increase model expressivity*
  - **Gaussian Radial Basis Functions:**

$$\phi_{\lambda_i,\mu_i}(x) = \exp\left(-\frac{||x-\mu_i||_2^2}{\lambda_i}\right)$$

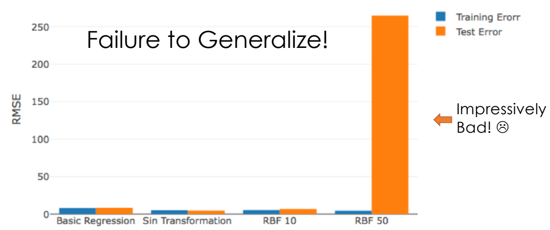




Training Error

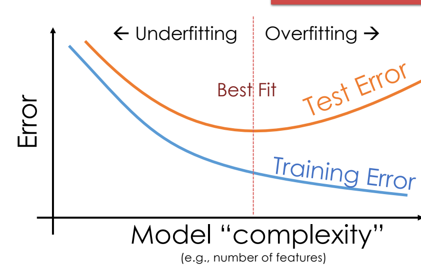


Training vs **Test** Error



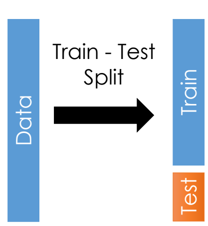
Training vs **Test** Error

Training error typically *under estimates* **test error.**



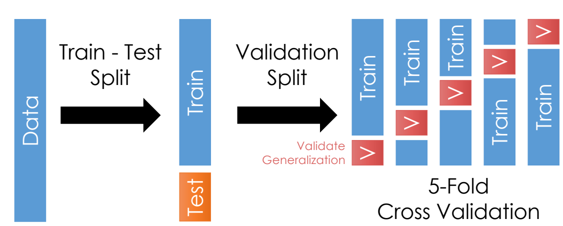
Generalization: *The Train-Test Split*

- **Training Data:** used to fit model
- **Test Data:** check generalization error
- How to split?
  - Randomly, Temporally, Geo…
  - Depends on application (usually randomly)
- What size? (90%-10%)
  - Larger training set → more complex models
  - Larger test set → better estimate of generalization error
  - Typically between 75%-25% and 90%-10%



You can only use the test dataset once after deciding on the model.

Generalization: *Validation Split*



*Cross validation **simulates multiple train test-splits** on the training data.*

Recipe for Successful Generalization

1. Split your data into **training** and **test** sets (90%, 10%)
2. Use **only the training data** when designing, training, and tuning the model
   - Use **cross validation** to test *generalization* during this phase
   - **Do not look at the test data**
3. Commit to your final model and train once more using **only the training data.**
4. Test the final model using the **test data.** If accuracy is not acceptable return to (2). (*Get more test data if possible.*)
5. Train **on all available data** and ship it!

Returning to Regularization

Regularization

Parametrically Controlling the *Model Complexity*

Basic Idea

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \mathbf{Loss}\left(y_i, f_\theta(x_i)\right)$$

Such that:

f_θ is not too "complicated"

Can we make this more formal?

Basic Idea

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \mathbf{Loss}\left(y_i, f_\theta(x_i)\right)$$

Regularization Parameter

Such that:

Complexity$(\ f_\theta\ ) \leq \beta$

How do we define this?

Idealized Notion of Complexity

Complexity$(\ f_\theta\ ) \leq \beta$

- Focus on complexity of **linear models:**
  - Number and kinds of features
- Ideal definition:

$$\mathbf{Complexity}(f_\theta) = \sum_{j=1}^{d} \mathbb{I}\left[\theta_j \neq 0\right]$$

Number of non-zero parameters

- Why?

Ideal "Regularization"

Find the best value of θ which uses fewer than β features.

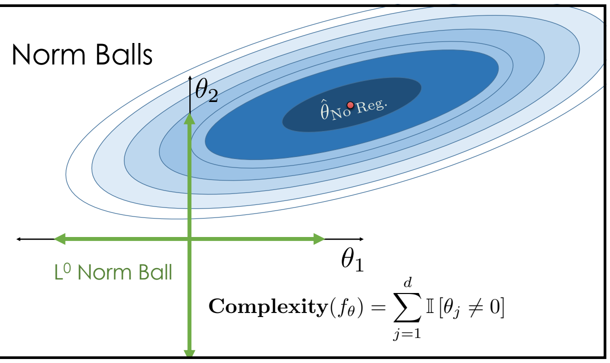
$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \mathbf{Loss}\left(y_i, f_\theta(x_i)\right)$$

Such that: Need an approximation!

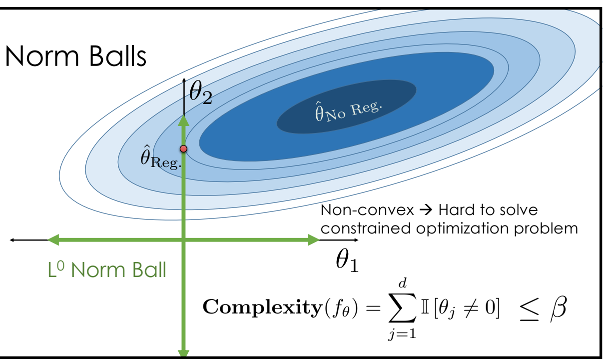
$$\mathbf{Complexity}(f_\theta) = \sum_{j=1}^{d} \mathbb{I}\left[\theta_j \neq 0\right] \;\leq \beta$$

Combinatorial search problem → NP-hard to solve in general.

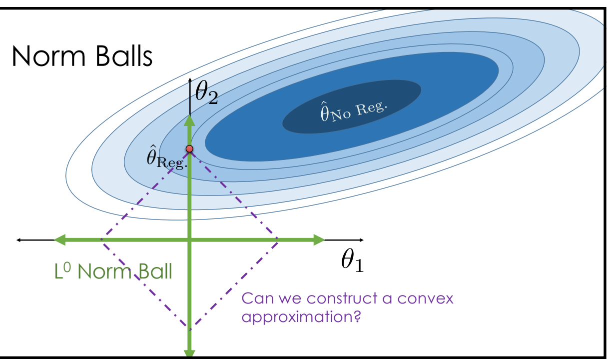
Norm Balls



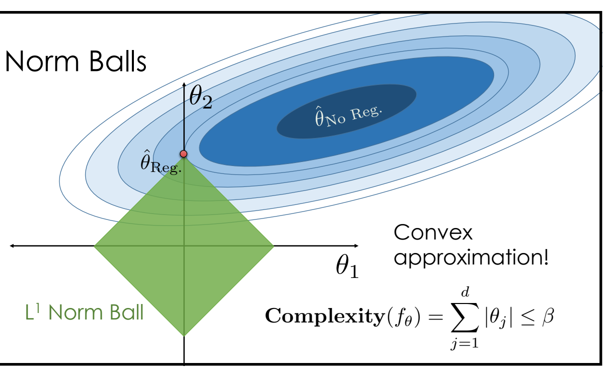
Norm Balls



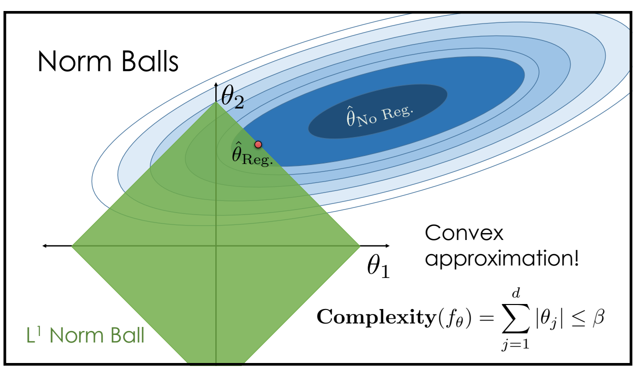
Norm Balls

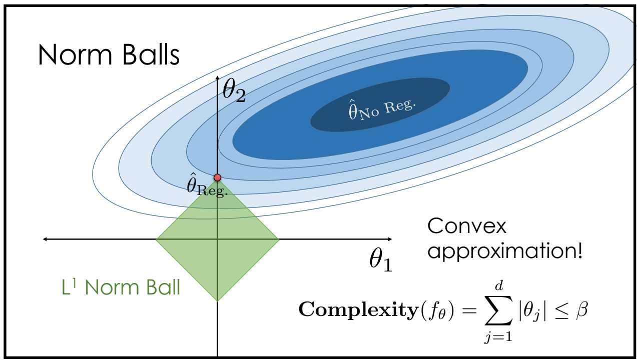


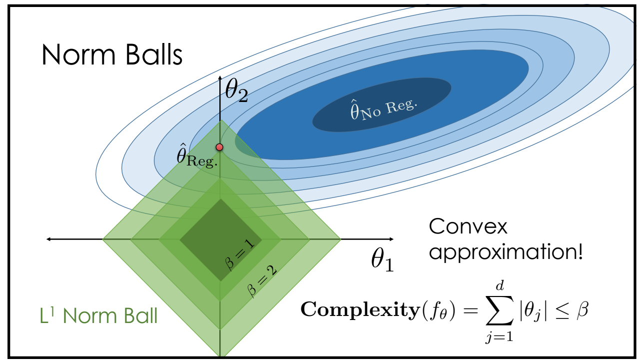
Norm Balls

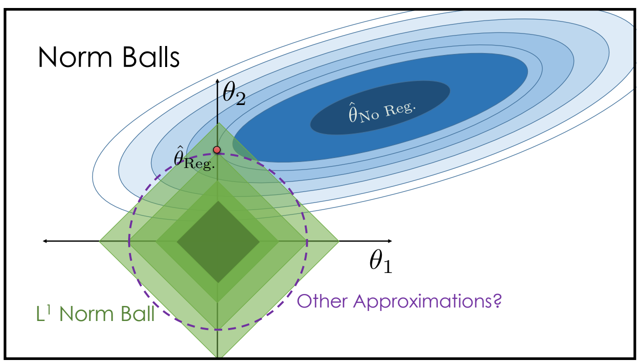


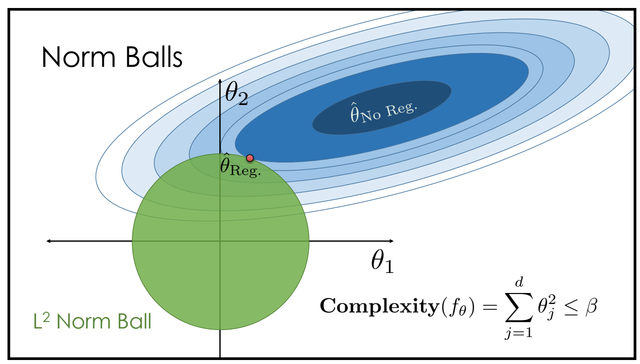
Norm Balls

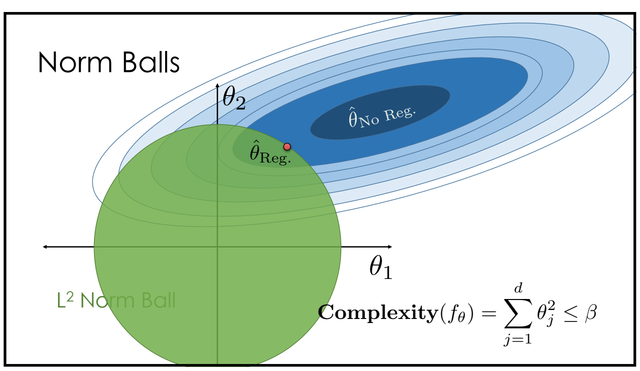


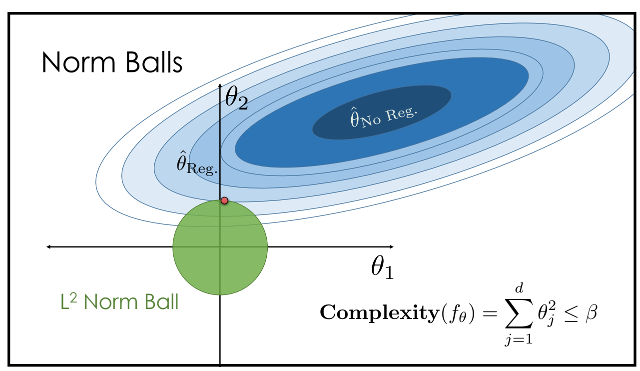

Norm Balls
θ_2
$\hat{\theta}_{\text{No Reg.}}$
$\hat{\theta}_{\text{Reg.}}$
Convex approximation!
θ_1
L¹ Norm Ball
$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} |\theta_j| \leq \beta$


Norm Balls
θ_2
$\hat{\theta}_{\text{No Reg.}}$
$\hat{\theta}_{\text{Reg.}}$
Convex approximation!
$\beta = 1$
$\beta = 2$
θ_1
L¹ Norm Ball
$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} |\theta_j| \leq \beta$

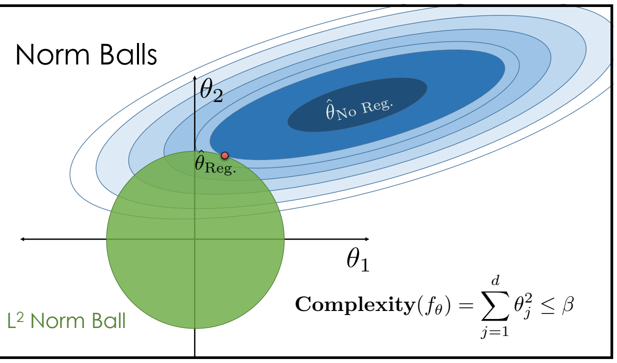

Norm Balls
θ_2
$\hat{\theta}_{\text{No Reg.}}$
$\hat{\theta}_{\text{Reg.}}$
θ_1
L¹ Norm Ball
Other Approximations?


Norm Balls
θ_2
$\hat{\theta}_{\text{No Reg.}}$
$\hat{\theta}_{\text{Reg.}}$
θ_1
L² Norm Ball
$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} \theta_j^2 \leq \beta$

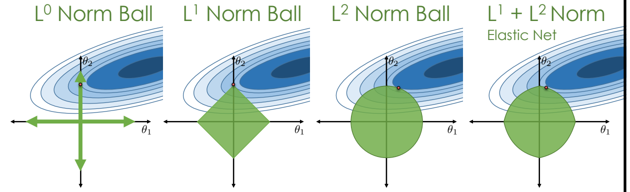

Norm Balls
θ_2
$\hat{\theta}_{\text{No Reg.}}$
$\hat{\theta}_{\text{Reg.}}$
θ_1
L² Norm Ball
$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} \theta_j^2 \leq \beta$


Norm Balls
θ_2
$\hat{\theta}_{\text{No Reg.}}$
$\hat{\theta}_{\text{Reg.}}$
θ_1
L² Norm Ball
$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} \theta_j^2 \leq \beta$

Norm Balls



$$\mathbf{Complexity}(f_\theta) = \sum_{j=1}^{d} \theta_j^2 \le \beta$$



Ideal for Feature Selection but combinatorically difficult to optimize

Encourages Sparse Solutions **Convex!**

Spreads weight over features **(robust)** does not encourage sparsity

Compromise Need to tune two regularization parameters

Generic Regularization (Constrained)

- Defining $\mathbf{Complexity}(f_\theta) = R(\theta)$

$$\hat{\theta} = \arg\min_\theta \frac{1}{n}\sum_{i=1}^{n} \mathbf{Loss}\left(y_i, f_\theta(x_i)\right)$$

<u>**Such that:**</u> $R(\theta) \le \beta$

- There is an equivalent unconstrained formulation (obtained by Lagrangian duality)

Generic Regularization (Constrained)

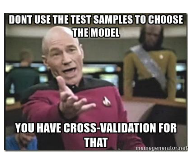
- Defining $\mathbf{Complexity}(f_\theta) = R(\theta)$

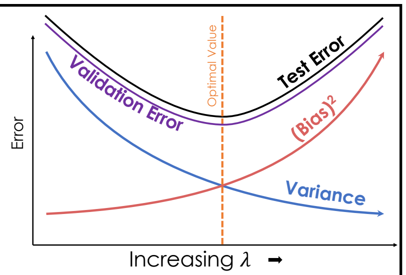
$$\hat{\theta} = \arg\min_\theta \frac{1}{n}\sum_{i=1}^{n} \mathbf{Loss}\left(y_i, f_\theta(x_i)\right) + \lambda R(\theta)$$

Regularization Parameter

- There is an equivalent unconstrained formulation (obtained by Lagrangian duality)

Determining the Optimal λ





- Value of λ determines bias-variance tradeoff
  - Larger values → more regularization → more bias → less variance
- Determined through cross validation

Using Scikit-Learn for Regularized Regression

```
import sklearn.linear_model
```

- Regularization parameter $\alpha = 1/\lambda$
  - larger α → less regularization → greater complexity → overfitting
- Lasso Regression (L1)
  - `linear_model.Lasso(alpha=3.0)`
  - `linear_model.LassoCV()` automatically picks α by cross-validation
- Ridge Regression (L2)
  - `linear_model.Ridge(alpha=3.0)`
  - `linear_model.RidgeCV()` automatically selects α by cross-validation
- Elastic Net (L1 + L2)
  - `linear_model.ElasticNet(alpha=3.0, l1_ratio = 2.0)`
  - `linear_model.ElasticNetCV()` automatically picks α by cross-validation

Standardization and the **Intercept** Term

`Height =` θ_1 `age_in_seconds +` θ_2 `weight_in_tons`

Small

Large

- Regularization penalized dimensions equally
- **Standardization**
  - Ensure that each dimensions has the same scale
  - centered around zero
- **Intercept Terms**
  - Typically don't regularize intercept term
  - Center *y* values (e.g., subtract mean)

Standardization

For each dimension *k*:

$$z_k = \frac{x_k - \mu_k}{\sigma_k}$$