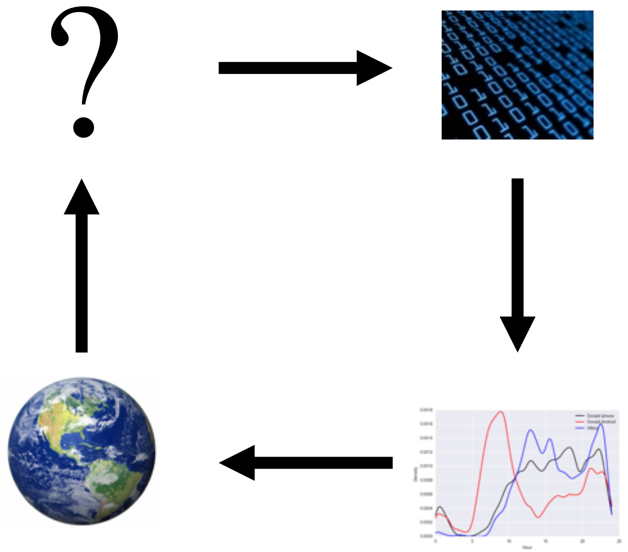
# *Fitting Linear Models, Regularization (revisited) & Cross Validation*

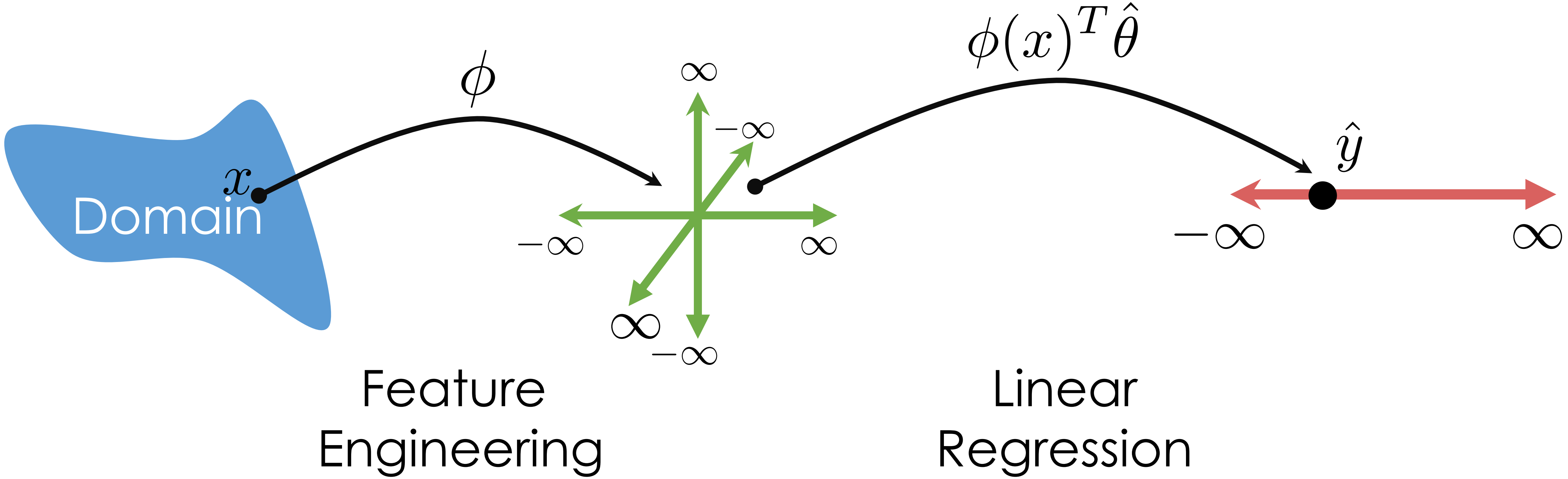
Slides by:

**Joseph E. Gonzalez** jegonzal@cs.berkeley.edu



# Previously

# Feature Engineering and Linear Regression



# **Recap:** *Feature Engineering*

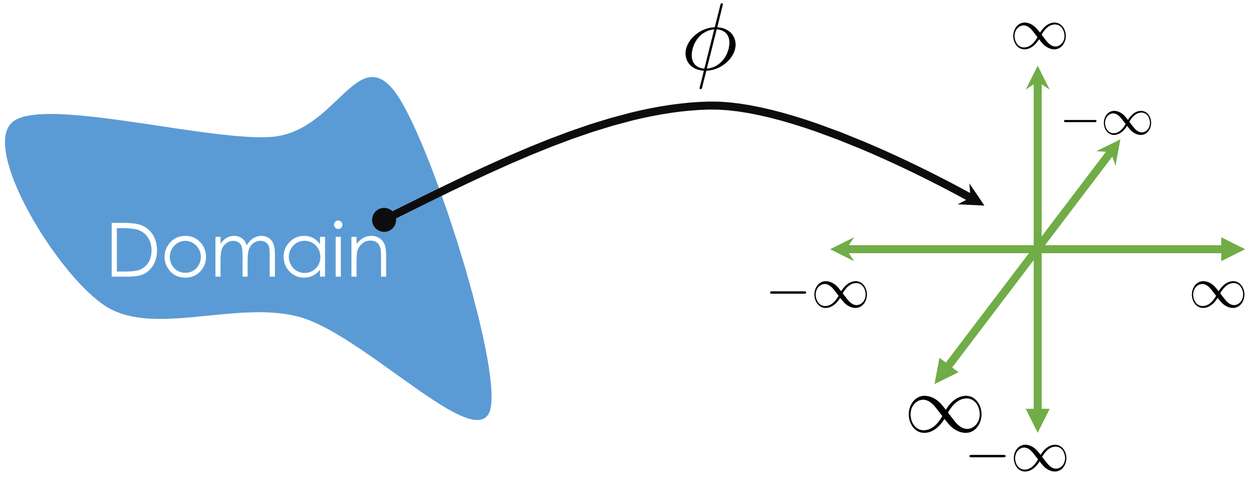
> **Notation:** Computer scientist / ML researchers tend to you *d* (dimensions) and statisticians will use *p* (parameters).

p

- Linear models with feature functions:

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

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



## ➢ 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."*

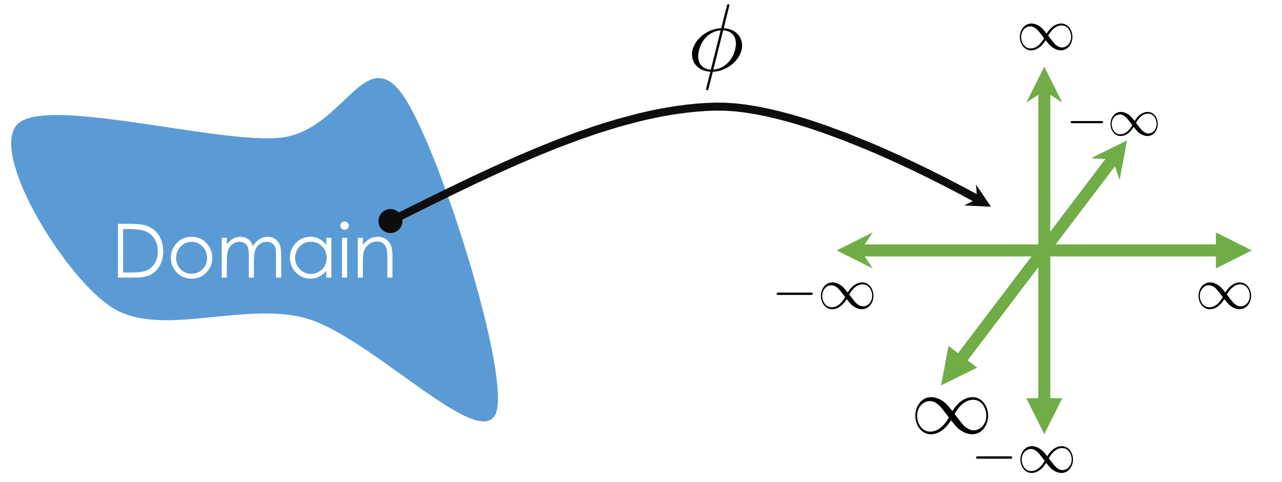
| aardvark | aardwolf | | fun | | learning | | machine | | zyzzyva |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | … | 1 | … | 2 | … | 1 | … | 0 |

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 $\Phi$




$X$ 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:

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

In this case we are assuming ***X*** is the transformed data $\mathbf{\Phi}$. 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 $\Phi$

| 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

$$\Phi \in \mathbb{R}^{n \times d} = \phi\left[X\right] = \begin{bmatrix} \text{---} & \phi\left(x^{(1)}\right) & \text{---} \\ \text{---} & \phi\left(x^{(2)}\right) & \text{---} \\ & \cdots & \\ \text{---} & \phi\left(x^{(n)}\right) & \text{---} \end{bmatrix}$$

DataFrame

n

d

**Rows** of the $\mathbf{\Phi}$ matrix correspond to records.

**Columns** of the $\mathbf{\Phi}$ matrix correspond to features.

Confusing notation!

# The Feature Matrix $\Phi$

| 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

$$\Phi \in \mathbb{R}^{n \times d} = \phi\left[ X \right] = \begin{bmatrix} —\phi(X_{1,\bullet})— \\ —\phi(X_{2,\bullet})— \\ \dots \\ —\phi(X_{n,\bullet})— \end{bmatrix}$$

DataFrame ($X$); $n$ rows, $d$ columns

**Rows** of the $\Phi$ matrix correspond to records.

**Columns** of the $\Phi$ matrix correspond to features.

Notation Guide

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

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

# Making Predictions

$$\Phi \in \mathbb{R}^{n \times d} = \phi\left[X\right] = \begin{bmatrix} \text{——} \ \phi\left(X_{1,\bullet}\right) \ \text{——} \\ \text{——} \ \phi\left(X_{2,\bullet}\right) \ \text{——} \\ \cdots \\ \text{——} \ \phi\left(X_{n,\bullet}\right) \ \text{——} \end{bmatrix}$$

DataFrame

n

d

**Rows** of the $\mathbf{\Phi}$ matrix correspond to records.

**Columns** of the $\mathbf{\Phi}$ matrix correspond to features.

## Prediction

$$\hat{Y} = f_{\hat{\theta}}(X) = \Phi\hat{\theta} = \begin{bmatrix} \text{——} \ \phi\left(X_{1,\bullet}\right) \ \text{——} \\ \text{——} \ \phi\left(X_{2,\bullet}\right) \ \text{——} \\ \cdots \\ \text{——} \ \phi\left(X_{n,\bullet}\right) \ \text{——} \end{bmatrix} \begin{bmatrix} | \\ \hat{\theta} \\ | \end{bmatrix} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \cdots \\ \hat{y}^{(n)} \end{bmatrix}$$

n

d

# 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:

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

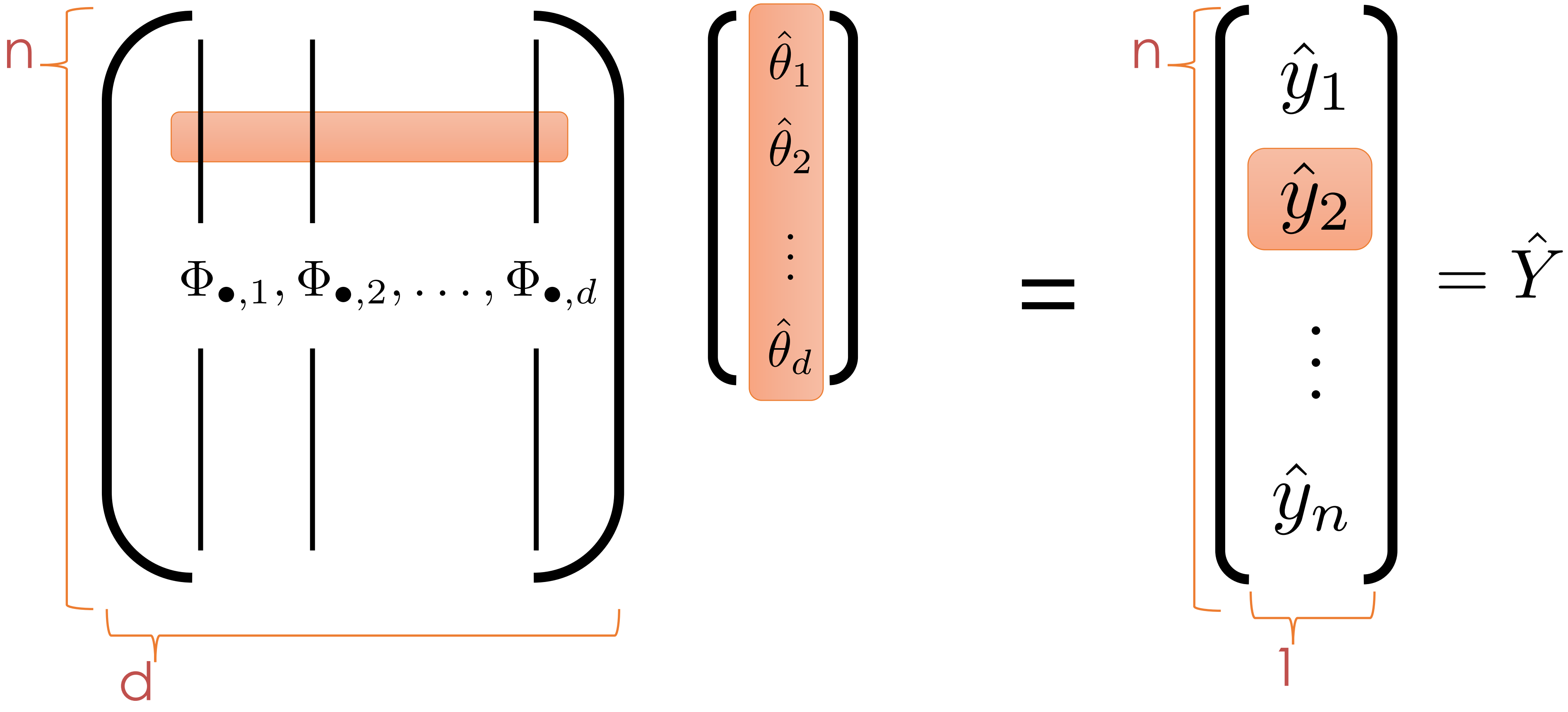
- **You should know this!**
- You do not need to know the calculus based derivation.
- **You should know the geometric derivation …**

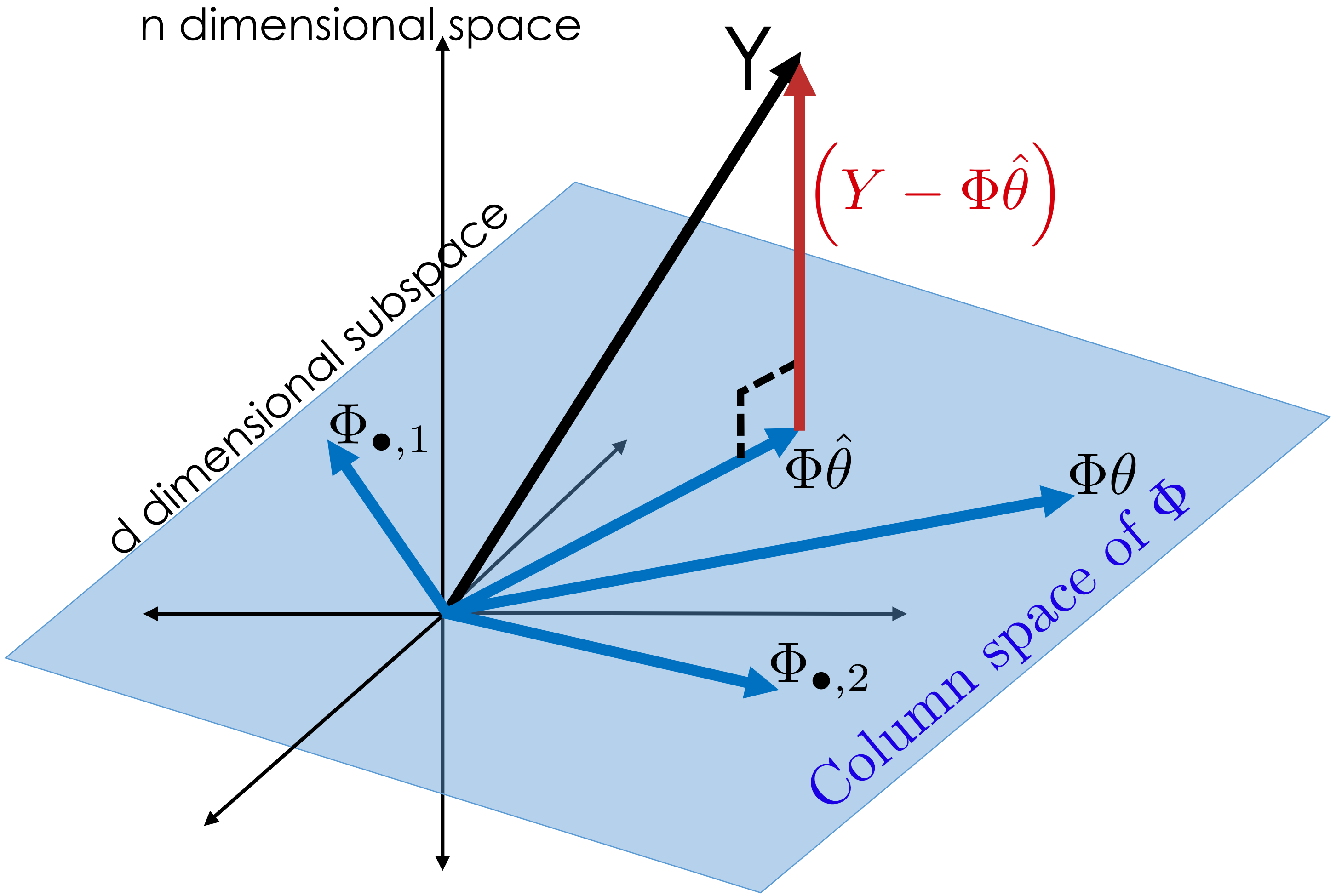
# Geometric Derivation: **Not Bonus Material**

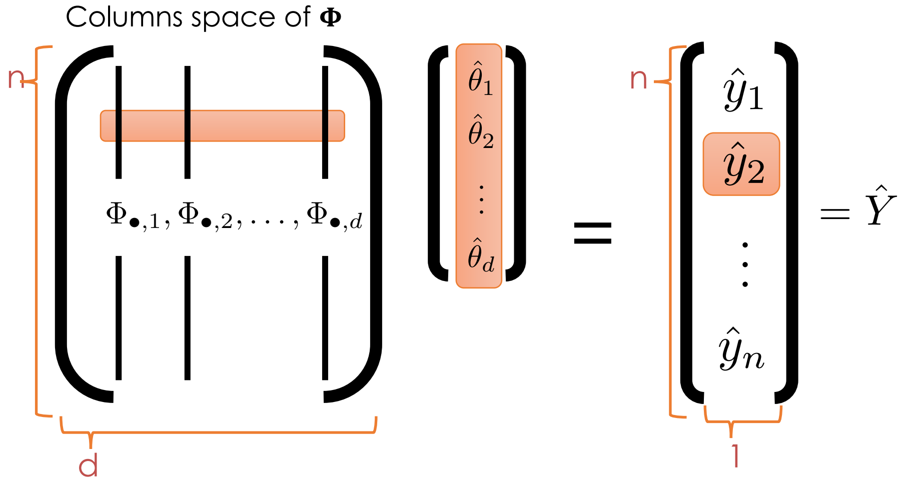
We decided that this is too exciting to not know.

- Examine the column spaces:

Columns space of $\mathbf{\Phi}$






**Derivation**

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

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

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

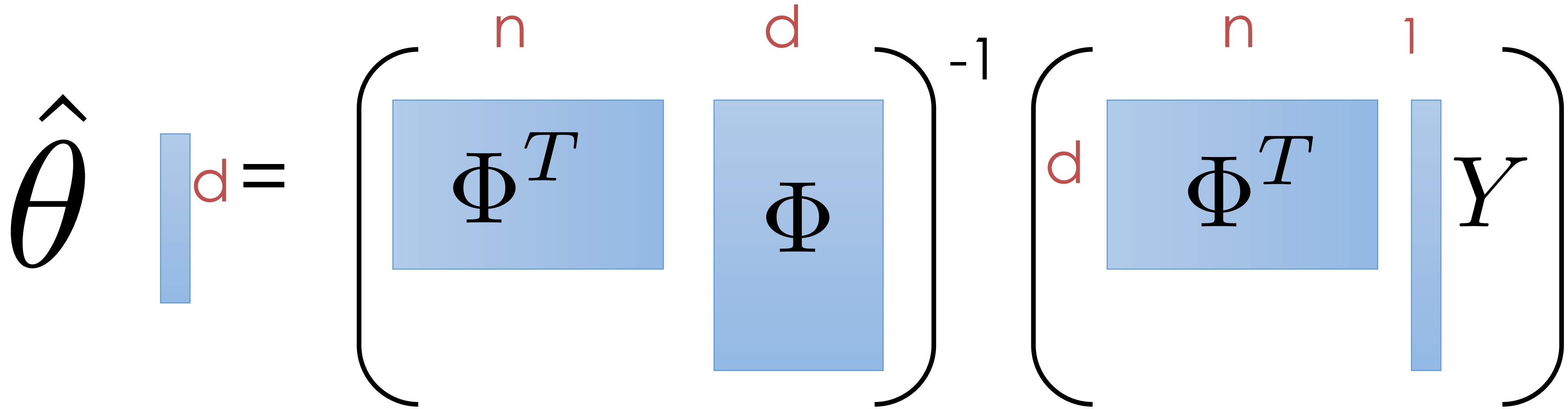
$$\hat{\theta} = \left(\Phi^T \Phi\right)^{-1} \Phi^T Y$$

"Normal Equation"

Definition of orthogonality

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

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




**Note:** For inverse to exist $\mathbf{\Phi}$ 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!

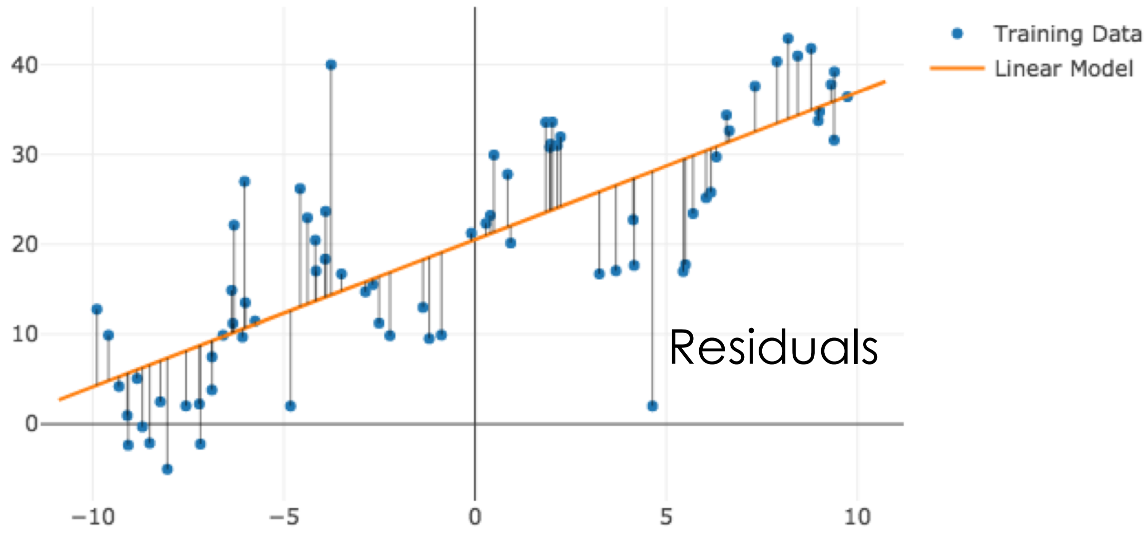
# Scikit Learn Models

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

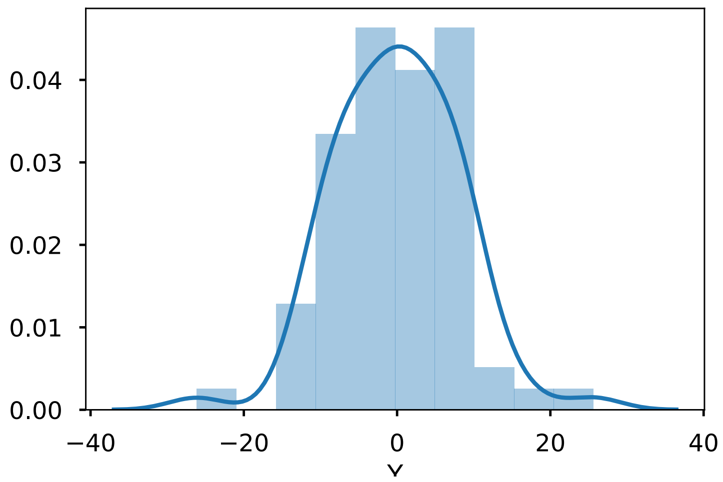
Ordinary Least Squares Regression

```
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']])
```

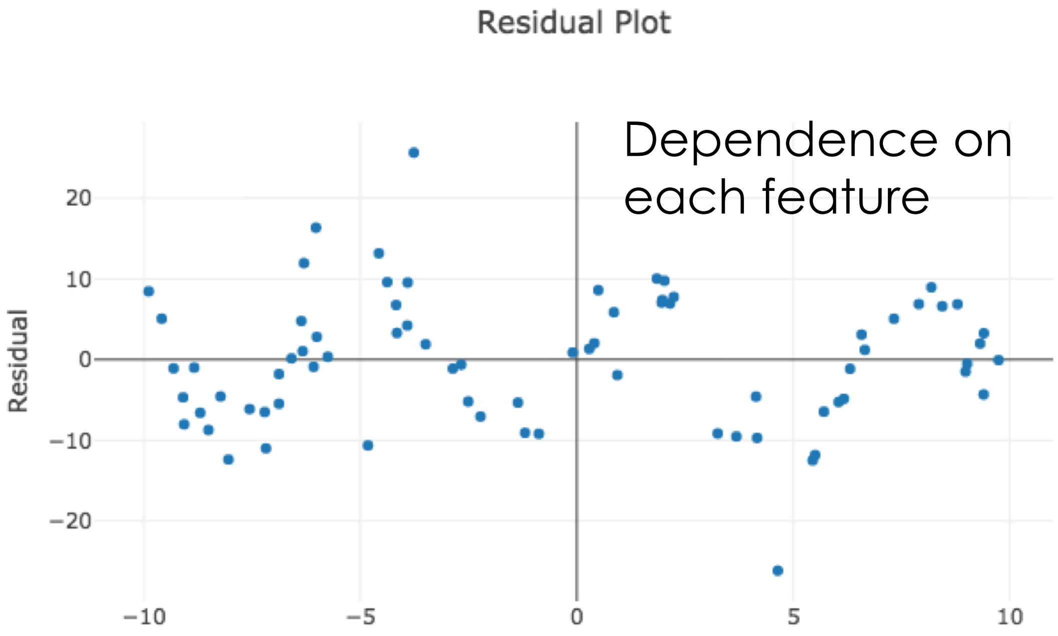
# Diagnosing Fit → The Residuals




Residuals

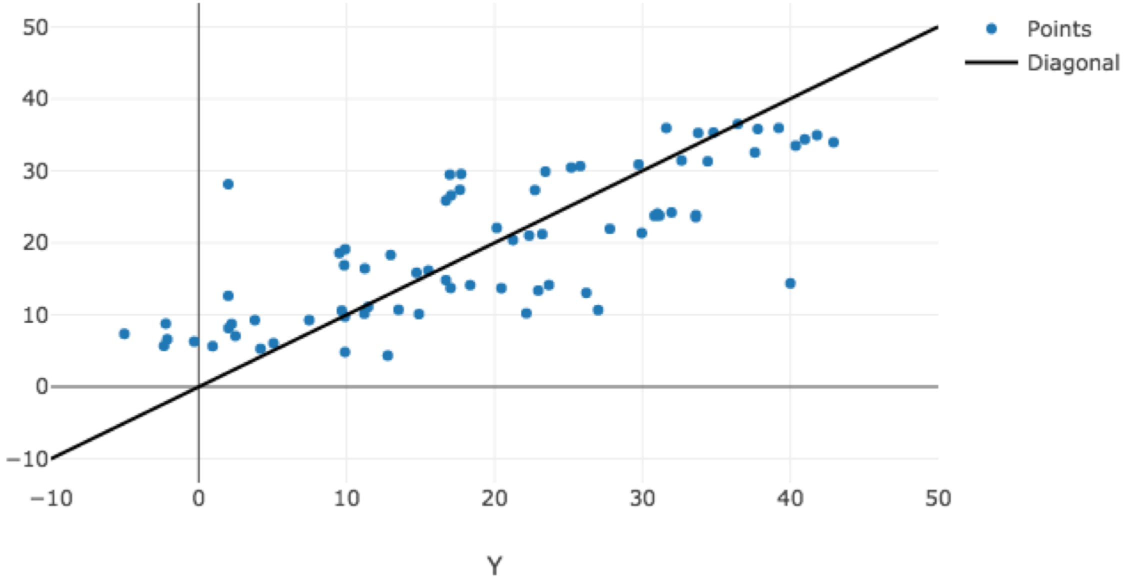



Look at Spread




Dependence on each feature

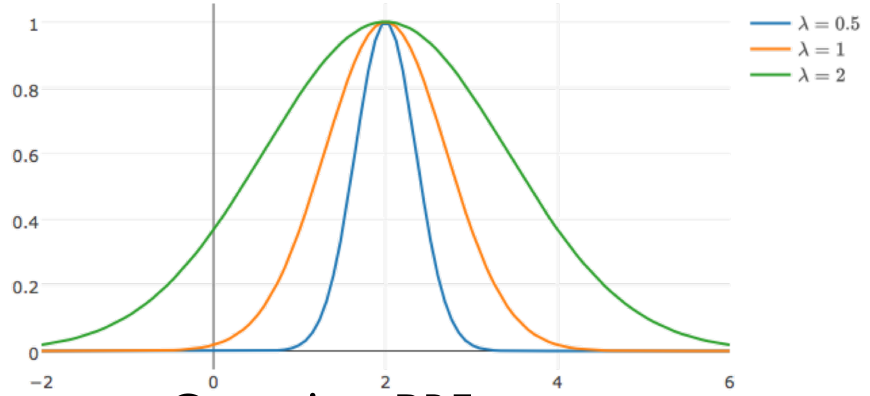
Predicted Y vs True Y



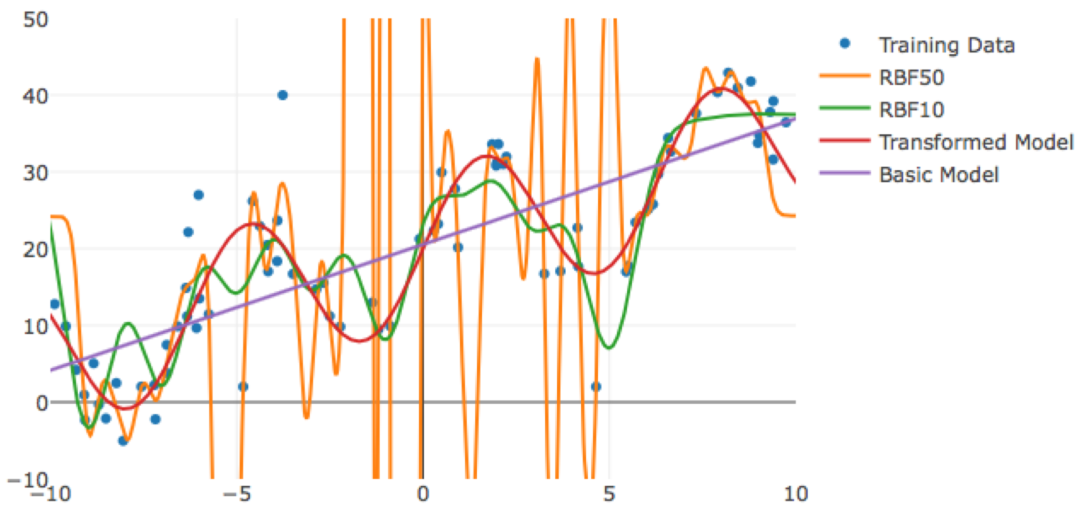
# Notebook Demo

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

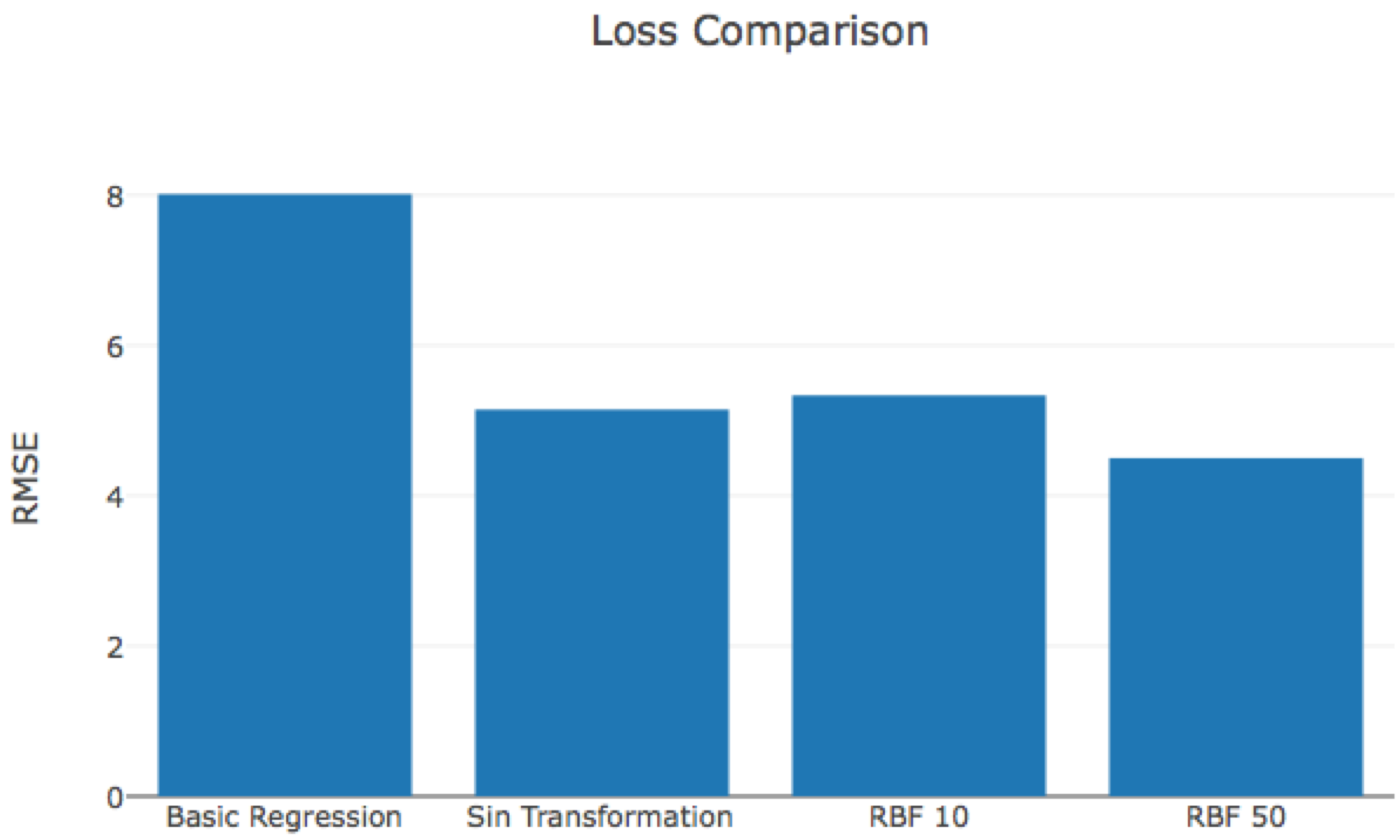
$$\phi_{\lambda_i,\mu_i}(x) = \exp\left(-\frac{||x-\mu_i||_2^2}{\lambda_i}\right)$$



Gaussian RBF

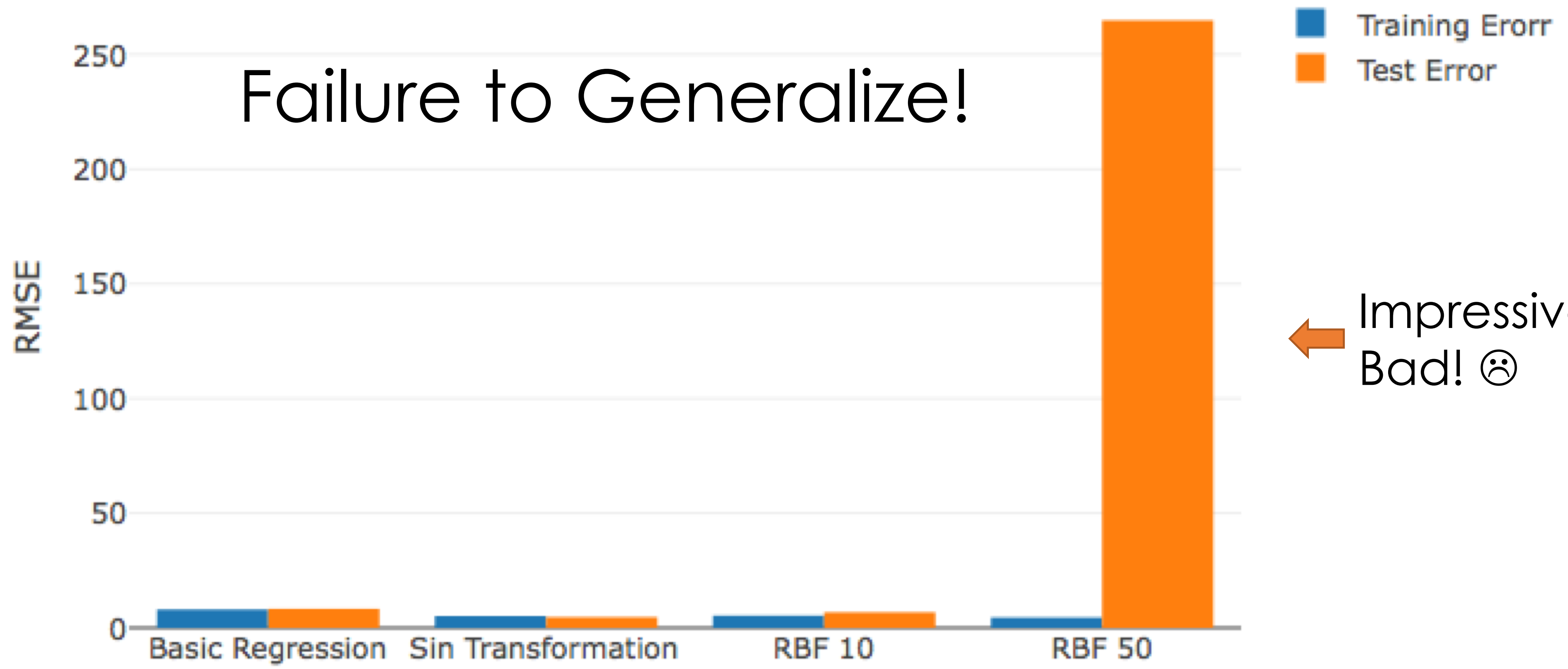


# **Training** Error




Impressive!

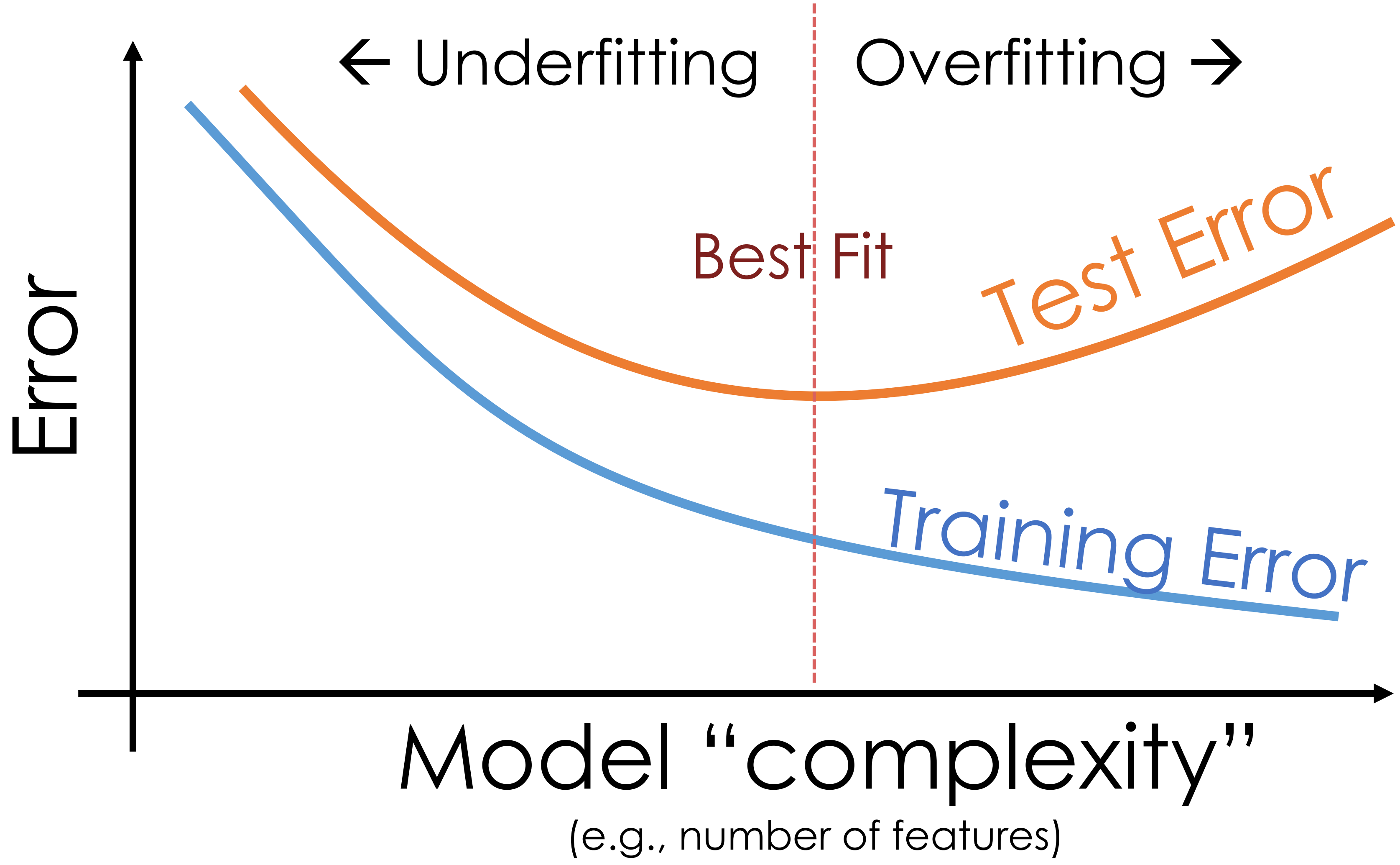
# **Training** vs **Test** Error




Impressively Bad! ☹

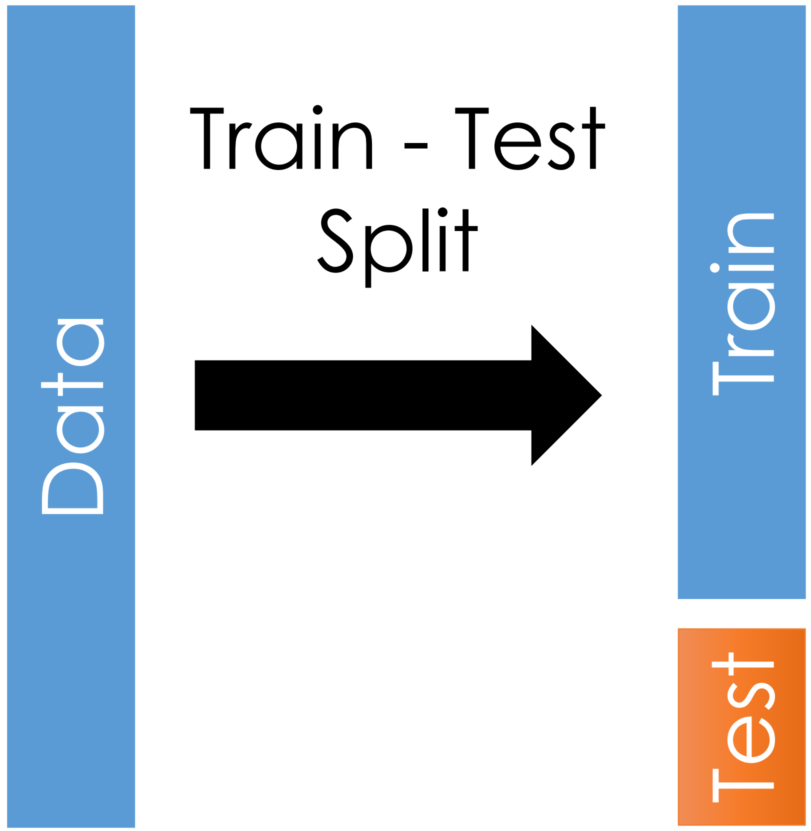
# **Training** vs **Test** Error

**Training error** typically <u>*under estimates*</u> **test error.**



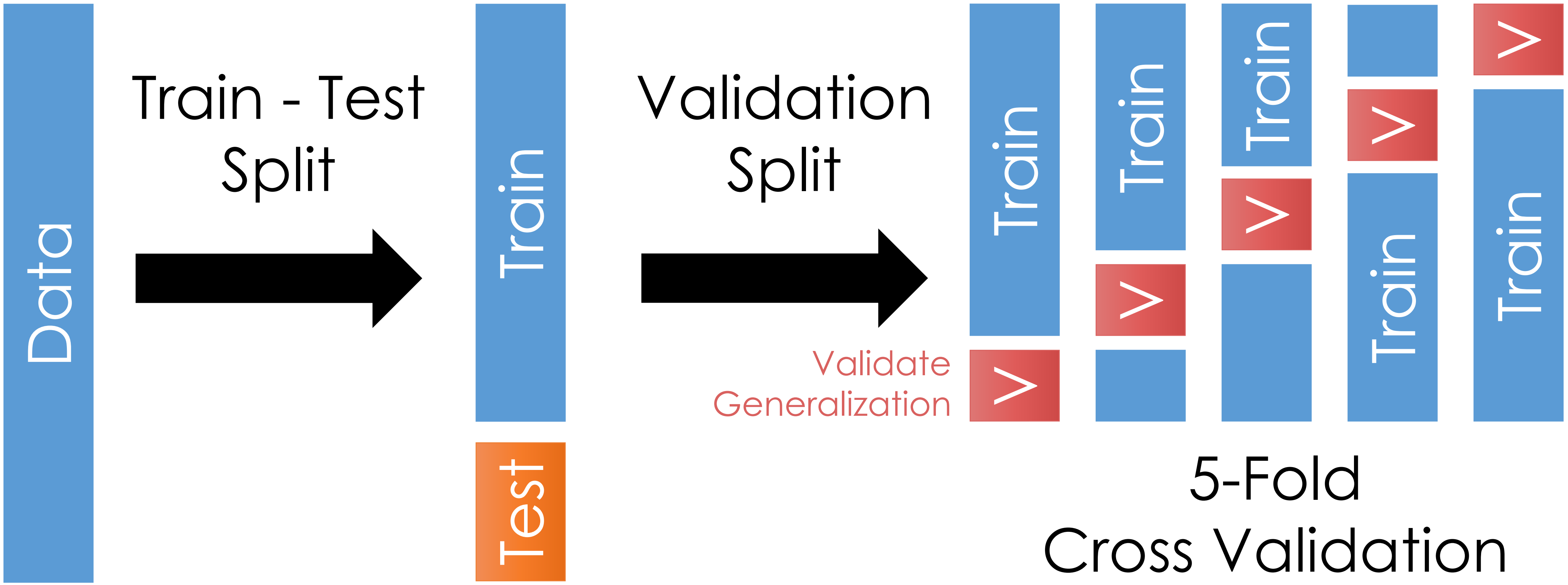
# **Generalization:** *The Train-Test Split*

- **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.*

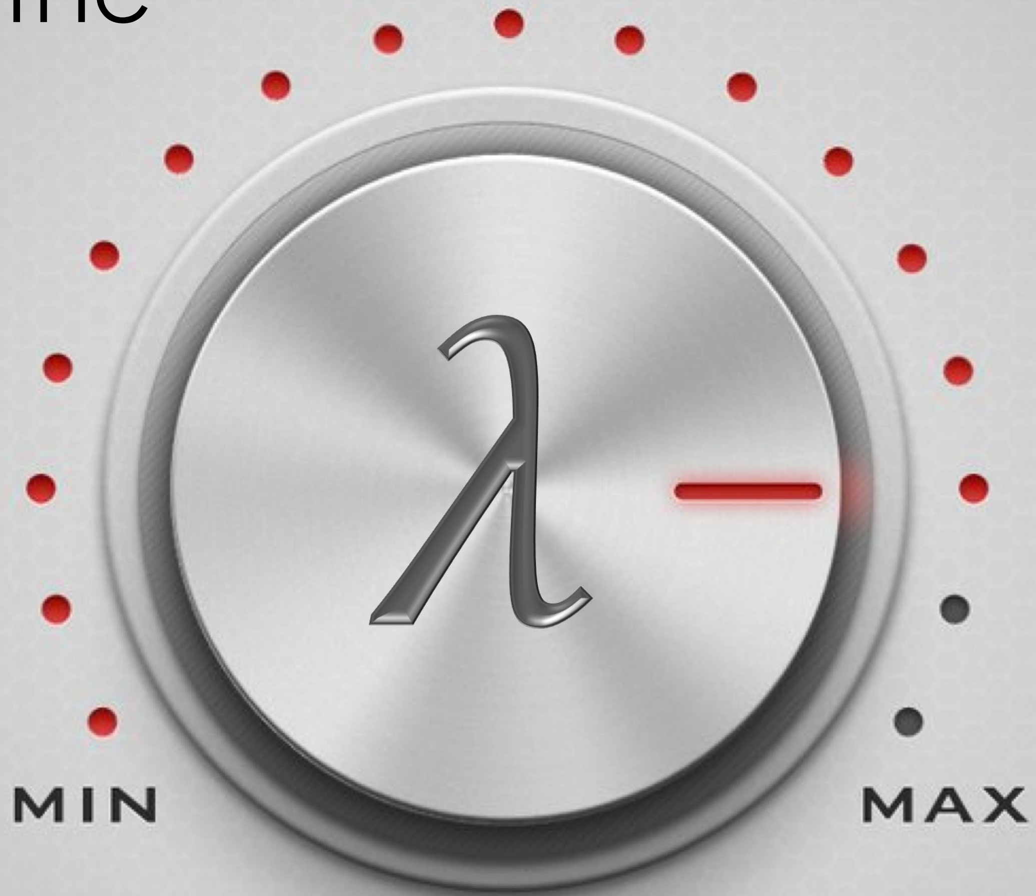
# Recipe for Successful Generalization

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:**

$f_\theta$ is not too "complicated"

Can we make this more formal?

# Basic Idea

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

**Such that:**

$$\text{Complexity}\left( f_\theta \right) \leq \beta$$

Regularization Parameter

How do we define this?

# Idealized Notion of Complexity

$$\text{Complexity}(\ f_\theta\ ) \leq \beta$$

- 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.

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

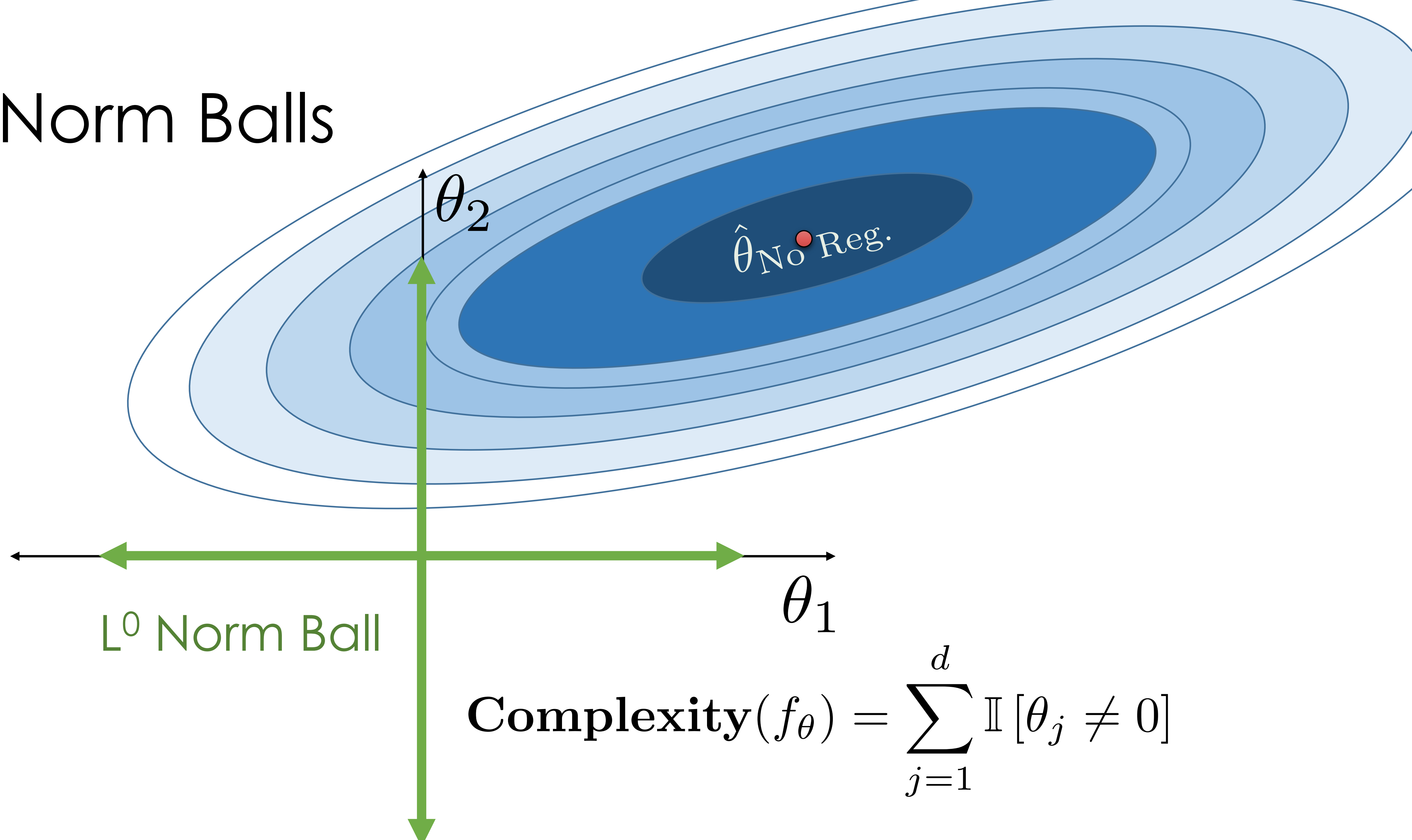
<u>**Such that:**</u>

Need an approximation!

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

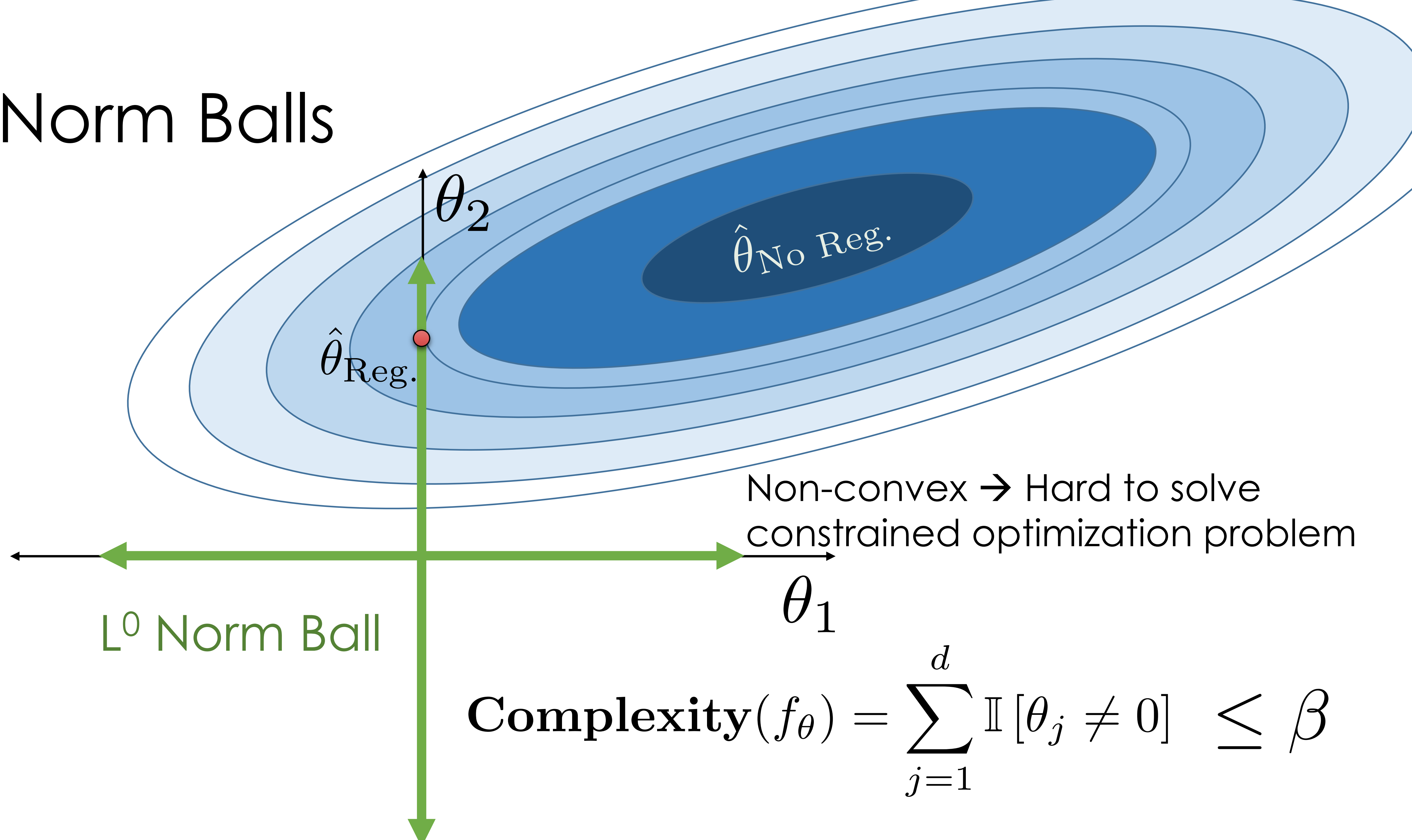
Combinatorial search problem → NP-hard to solve in general.

# Norm Balls




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

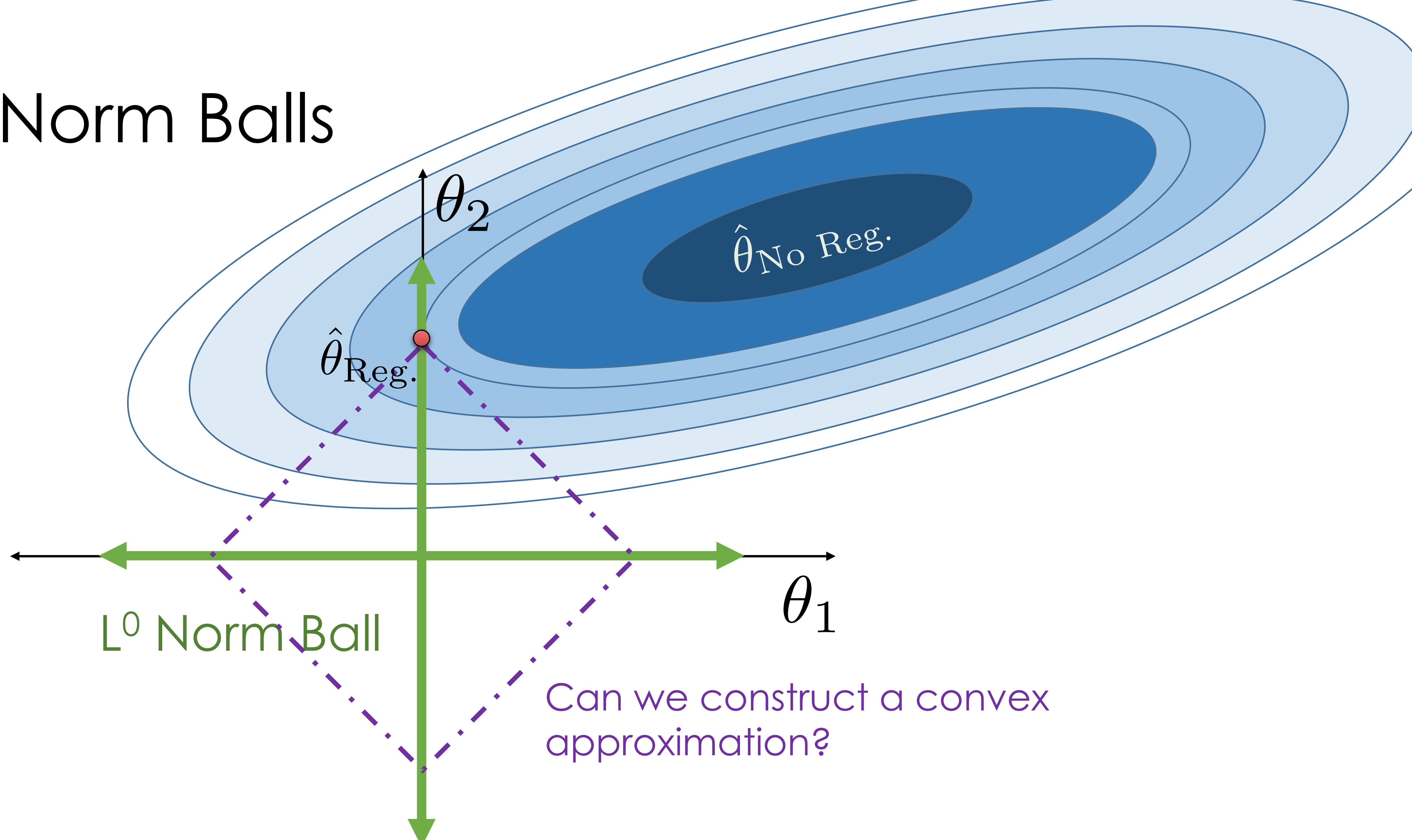
# Norm Balls




Non-convex → Hard to solve constrained optimization problem

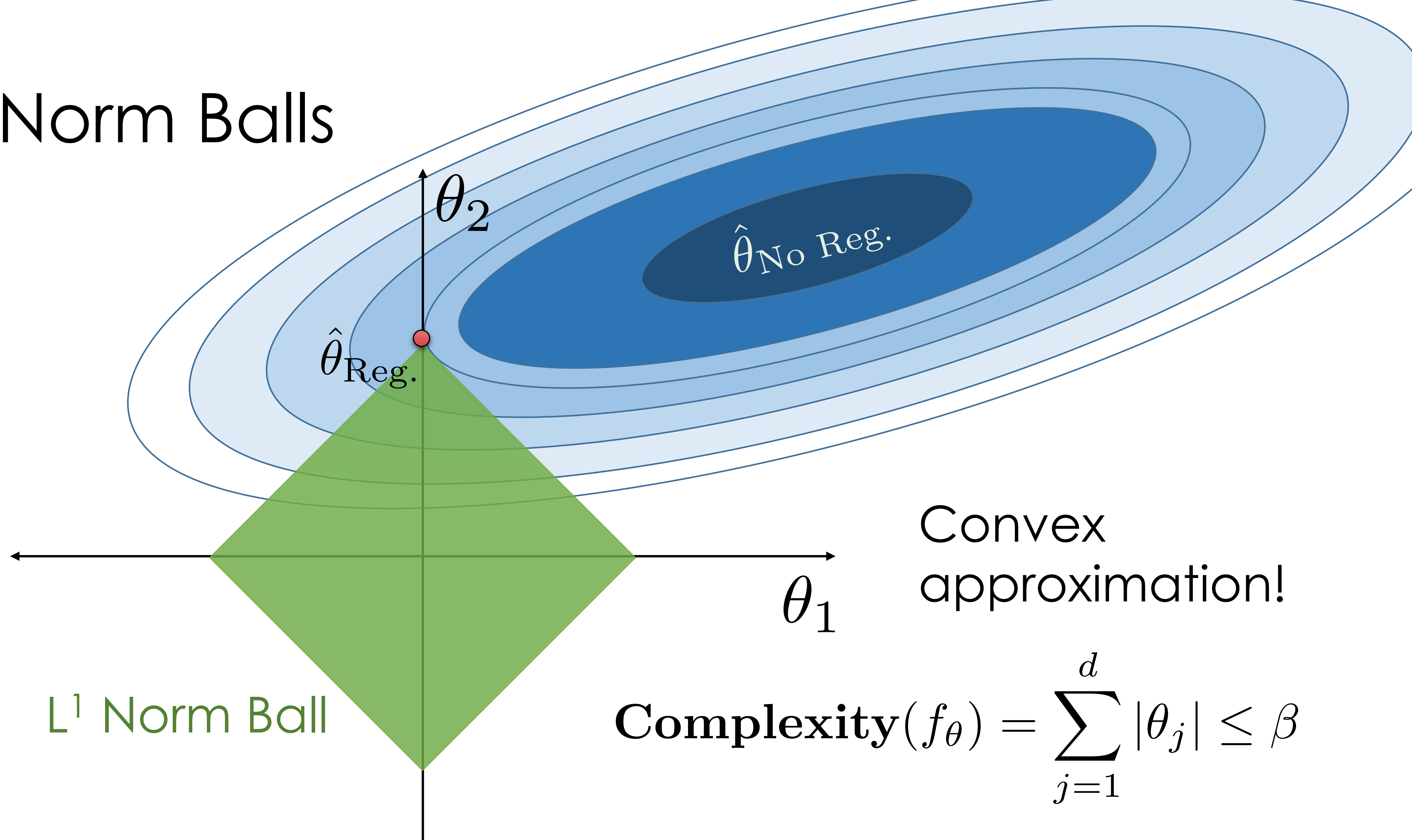
$$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} \mathbb{I}\left[\theta_j \neq 0\right] \leq \beta$$

# Norm Balls




Can we construct a convex approximation?

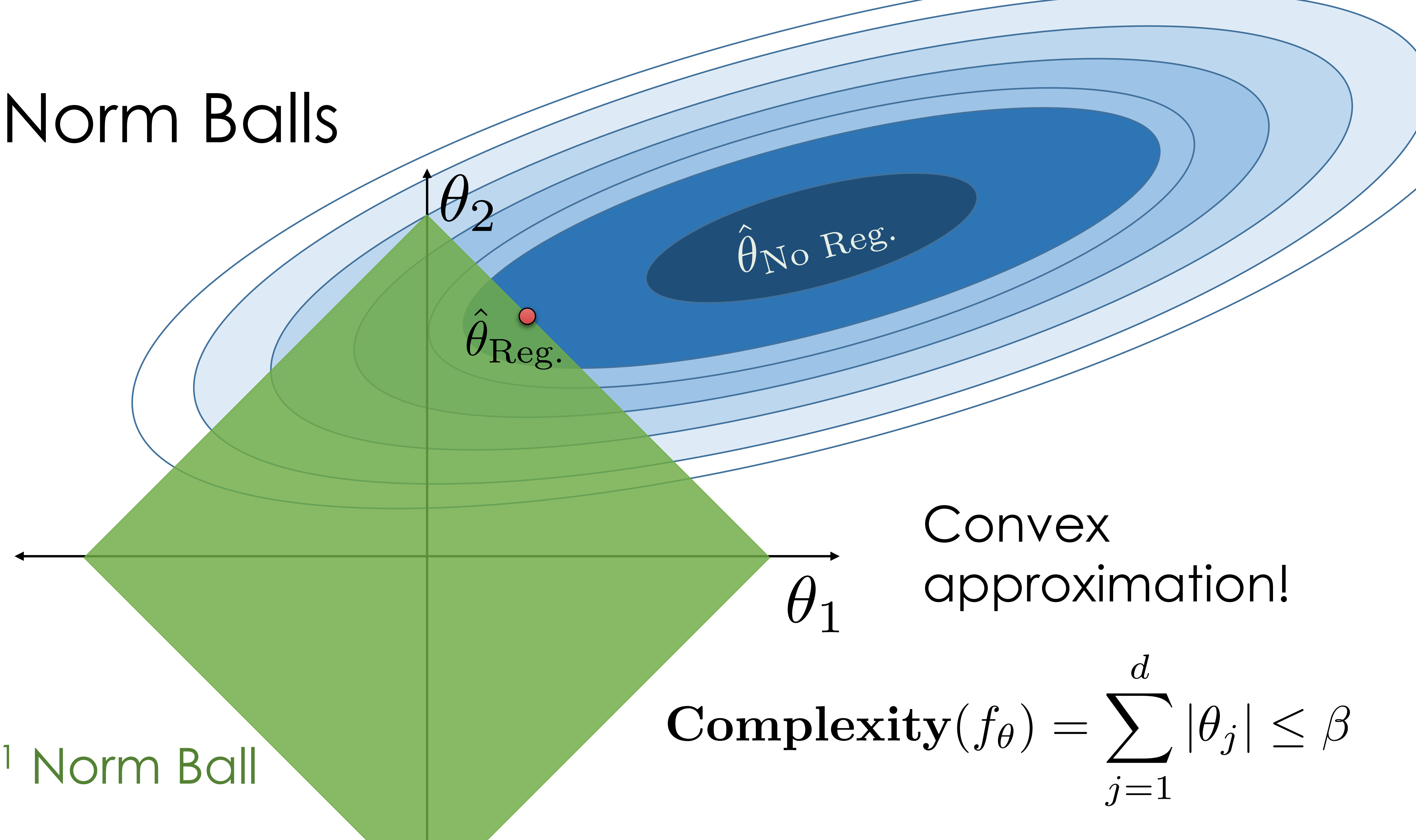
# Norm Balls




Convex approximation!

$$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} |\theta_j| \leq \beta$$

# Norm Balls

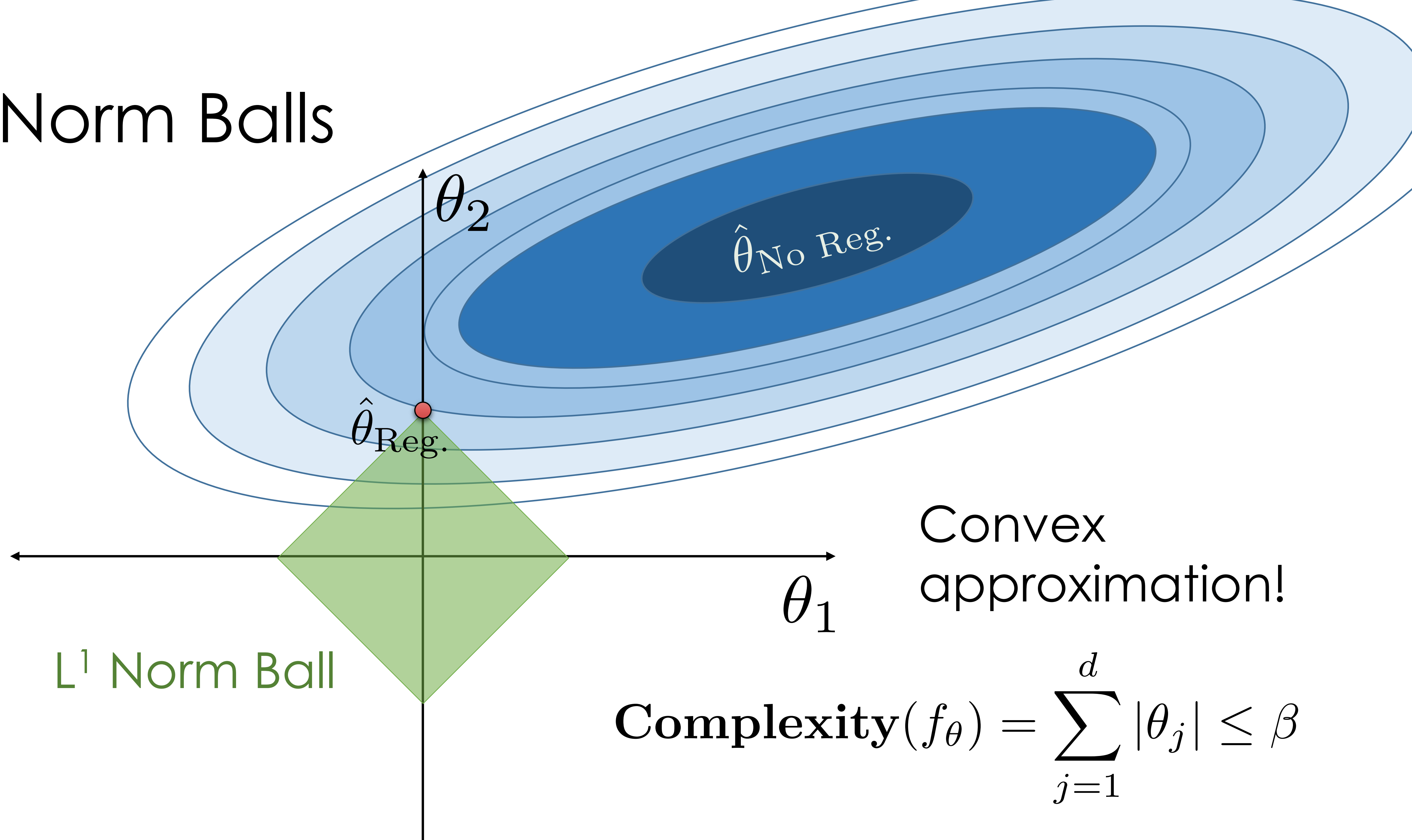



Convex approximation!

$$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} |\theta_j| \leq \beta$$

L[1] Norm Ball

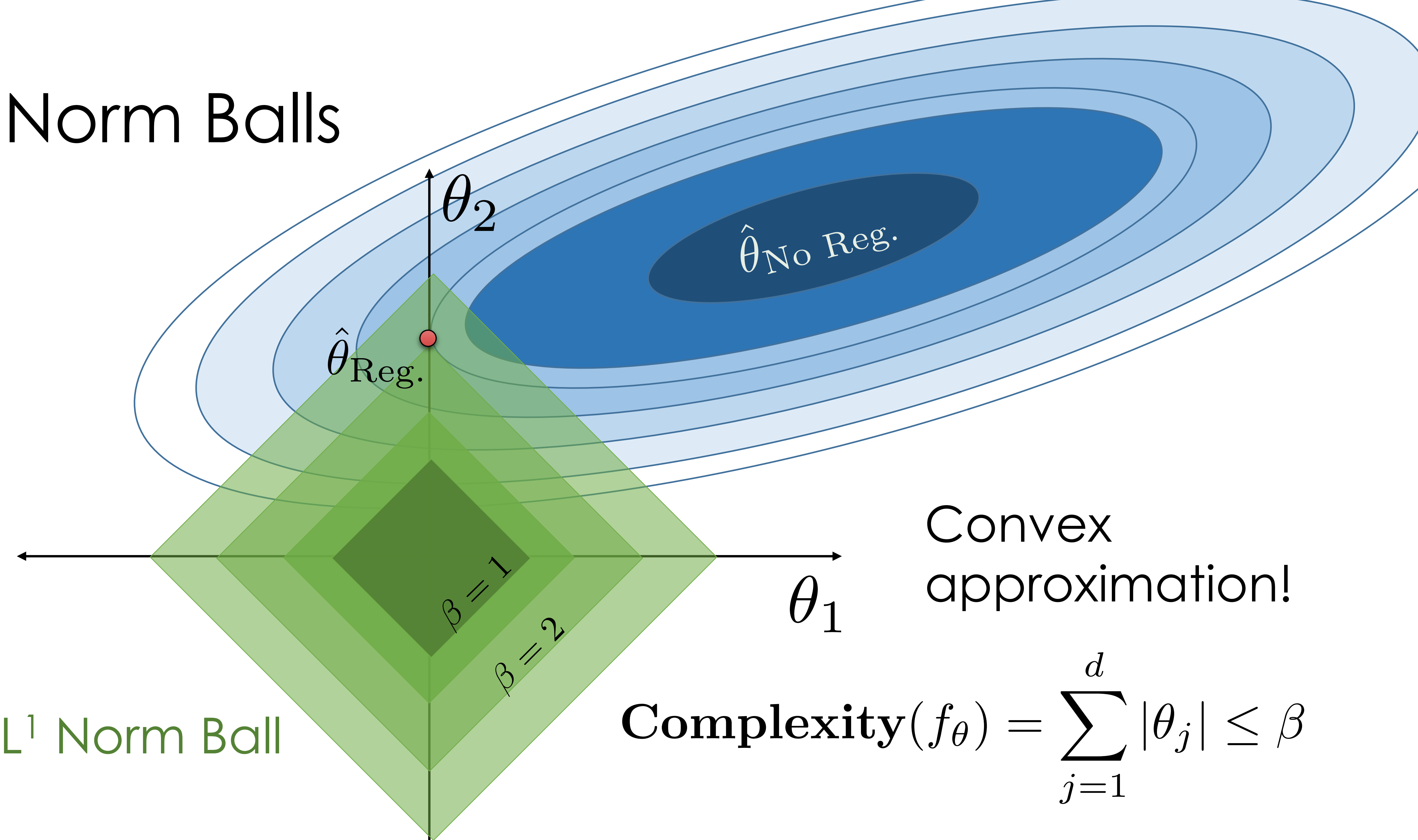
# Norm Balls




Convex approximation!

$$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} |\theta_j| \leq \beta$$

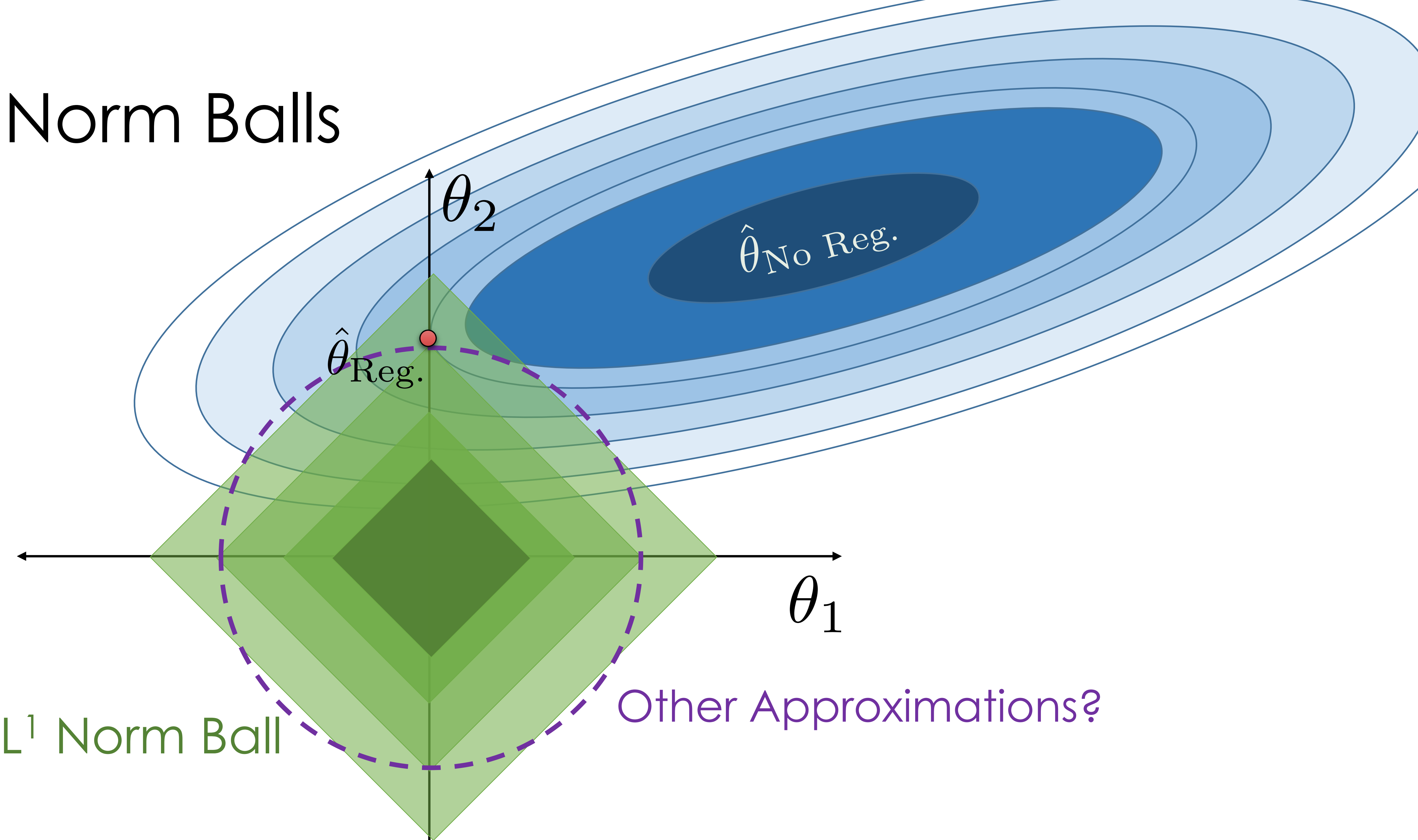
# Norm Balls




Convex approximation!

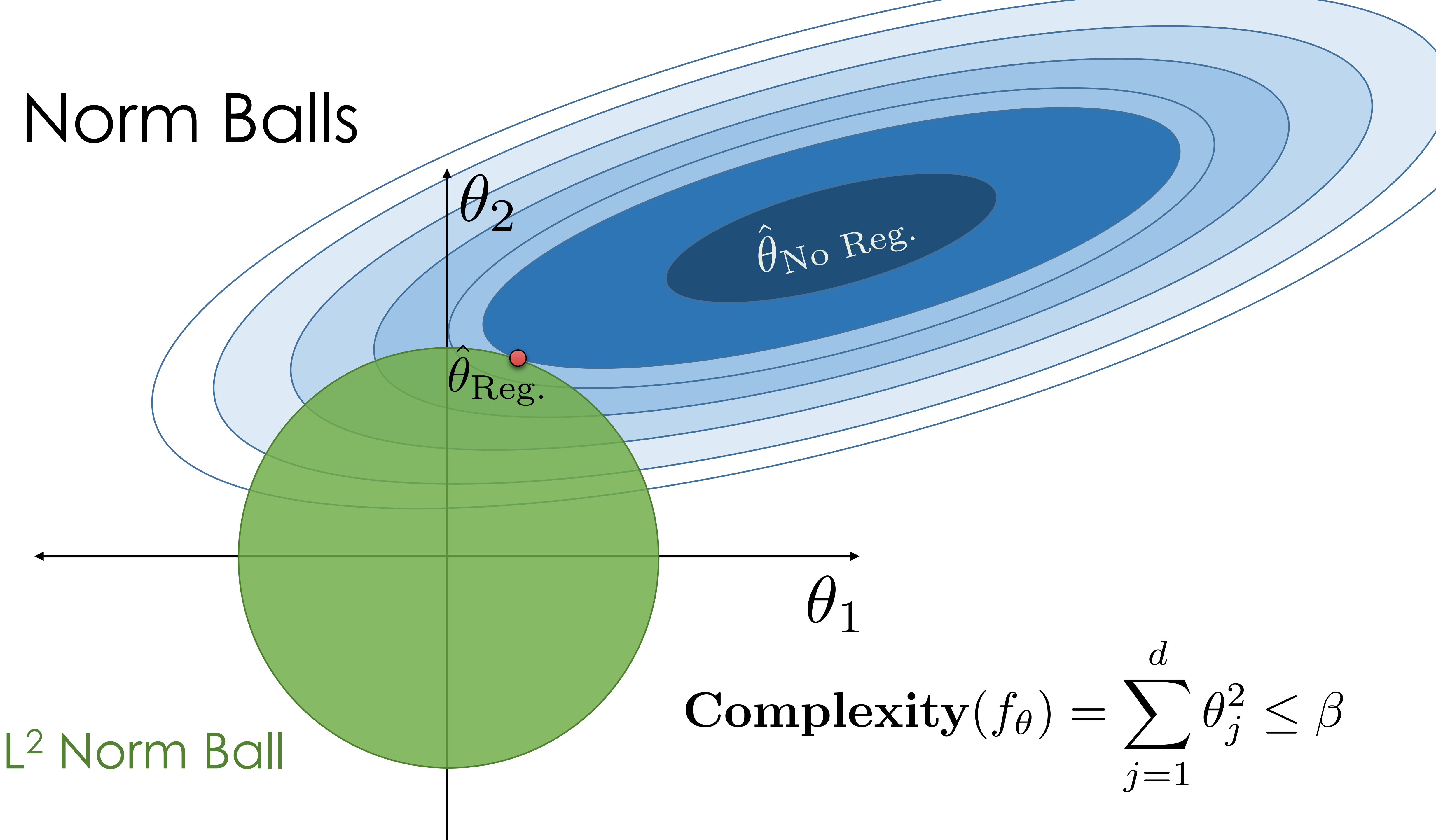
$$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} |\theta_j| \leq \beta$$

# Norm Balls



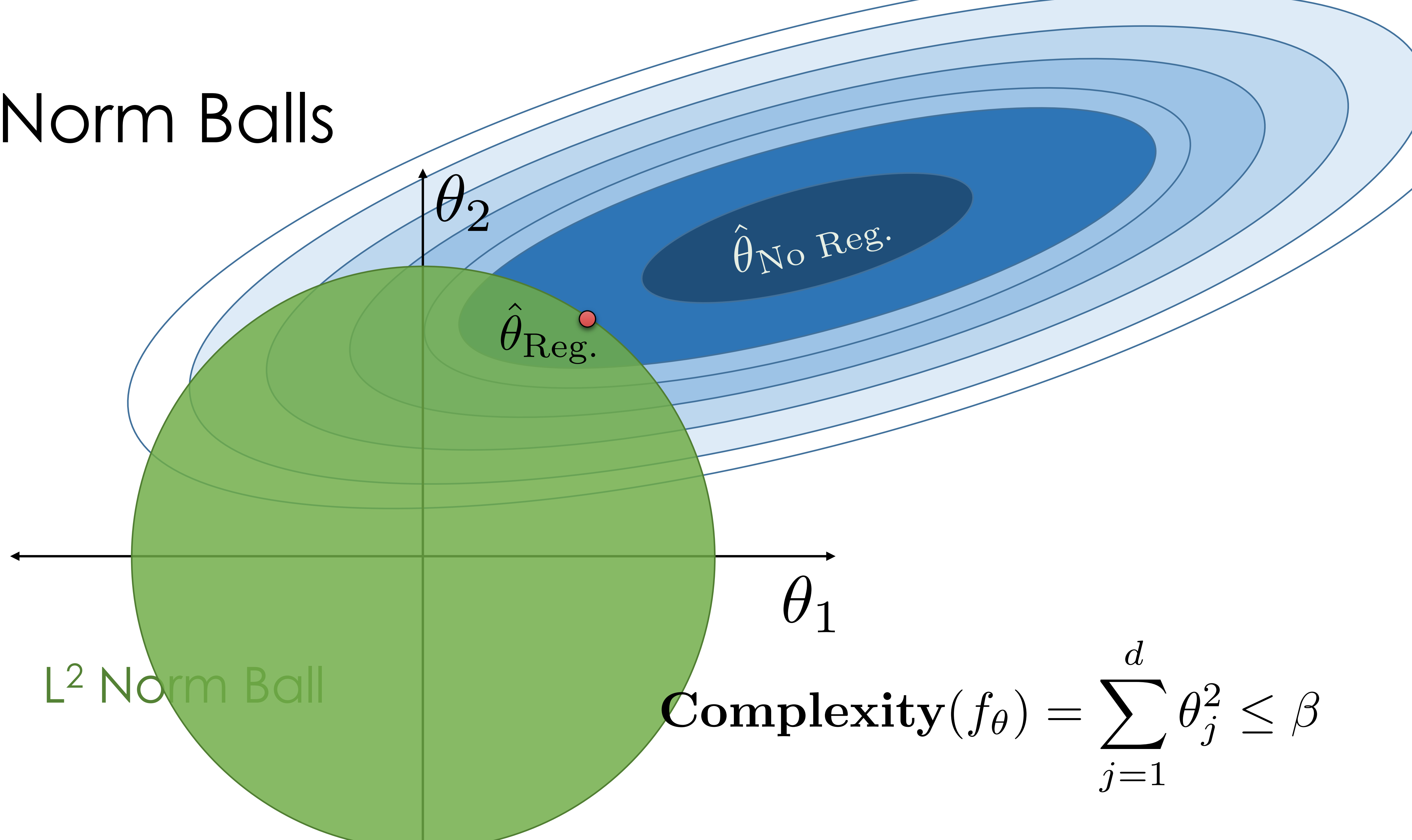
$\theta_2$
$\hat{\theta}_{\text{No Reg.}}$
$\hat{\theta}_{\text{Reg.}}$
$\theta_1$
Other Approximations?
$L^1$ Norm Ball

# Norm Balls



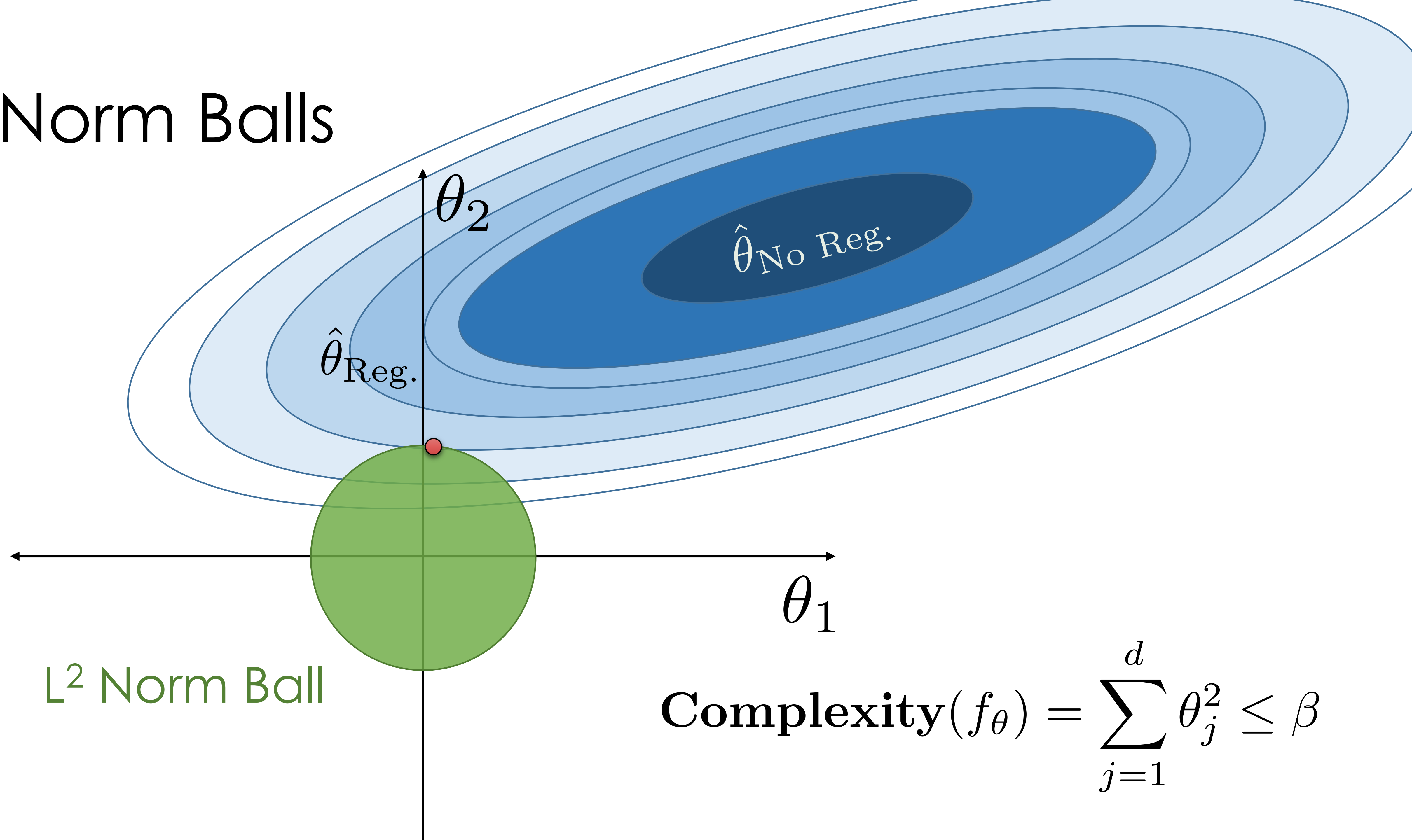

$$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} \theta_j^2 \leq \beta$$

# Norm Balls



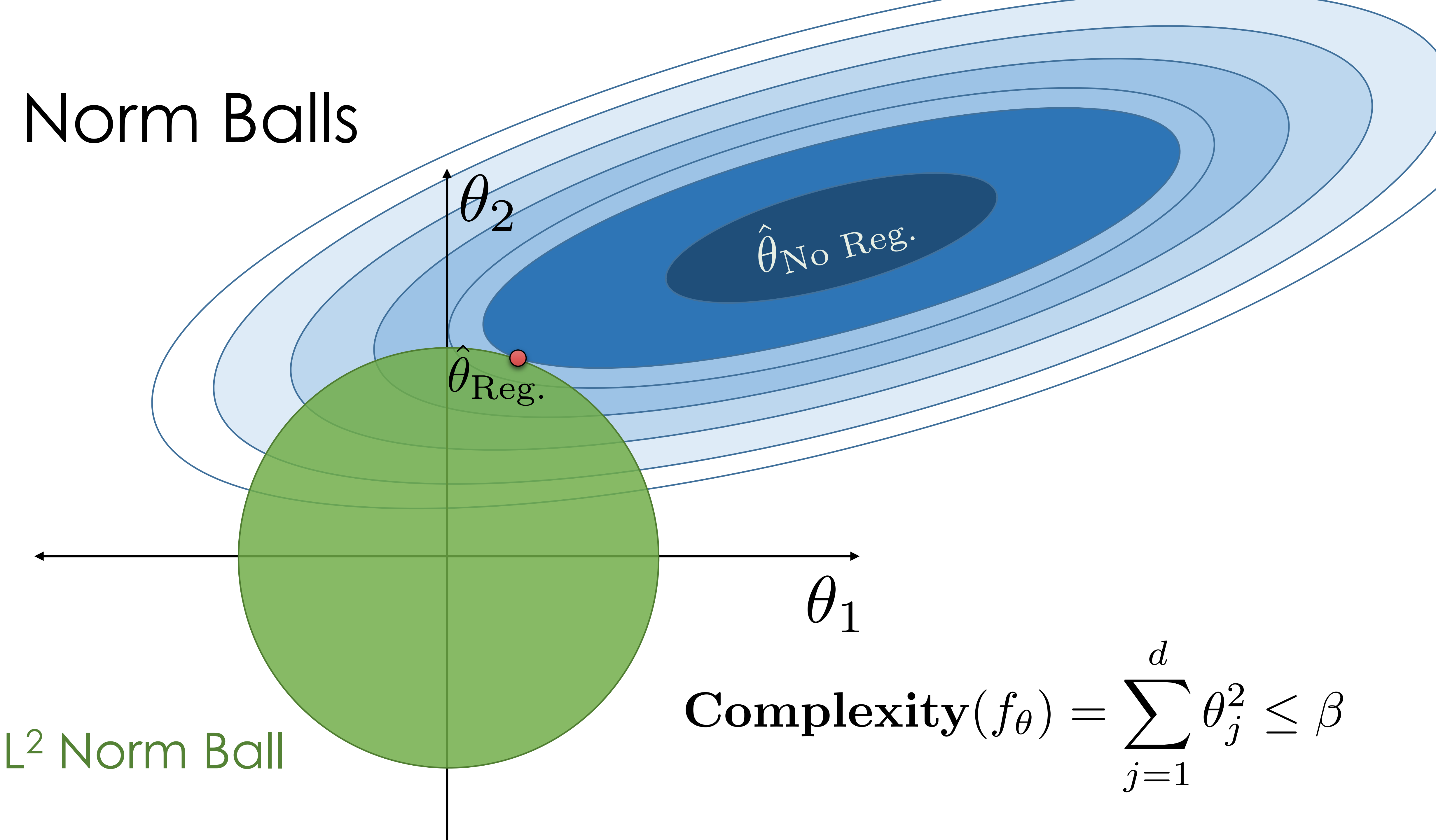

$$\textbf{Complexity}(f_\theta) = \sum_{j=1}^{d} \theta_j^2 \leq \beta$$

# Norm Balls




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

# Norm Balls




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

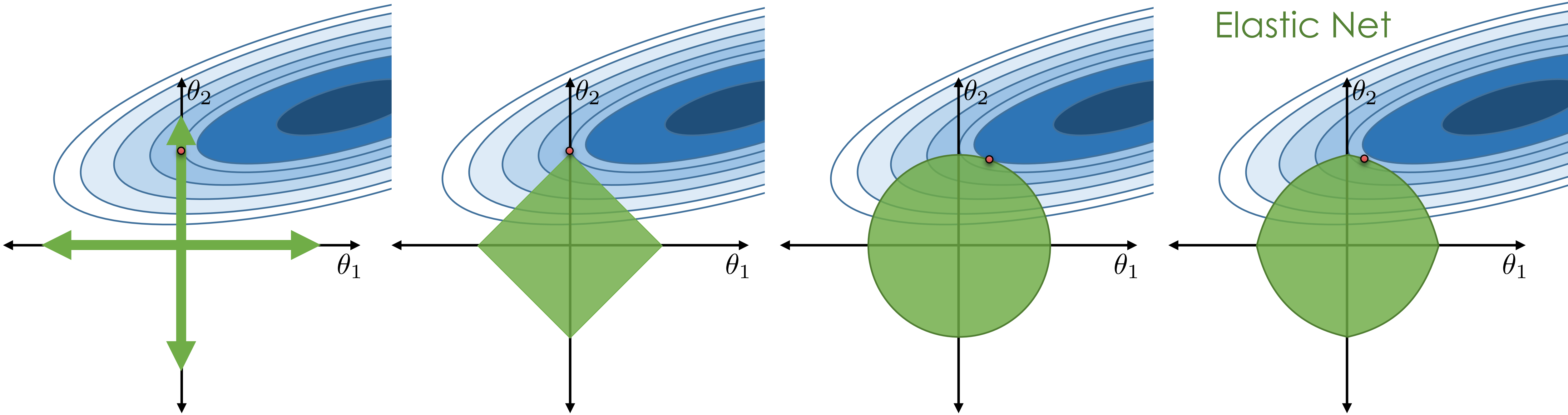
## $L^0$ Norm Ball

## $L^1$ Norm Ball

## $L^2$ Norm Ball

## $L^1 + L^2$ Norm

Elastic Net




**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 $\textbf{Complexity}(f_\theta) = R(\theta)$

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

**Such that:** $R(\theta) \leq \beta$

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

# Generic Regularization (Constrained)

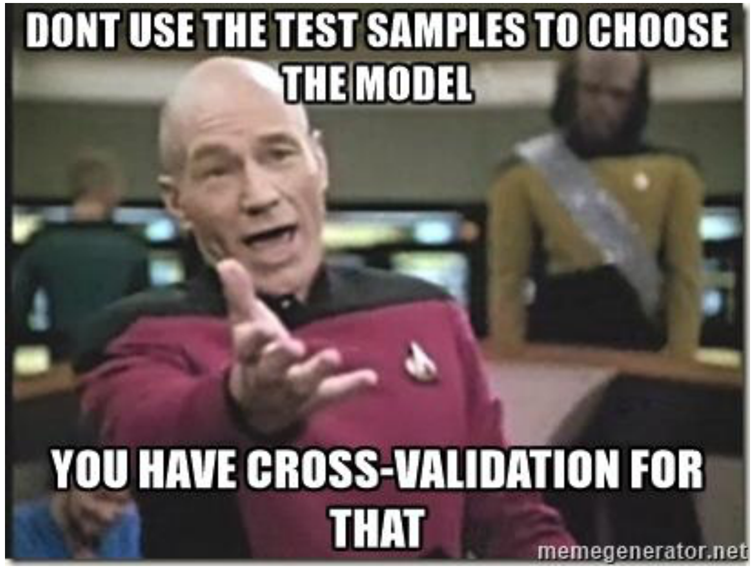
- Defining $\textbf{Complexity}(f_\theta) = R(\theta)$

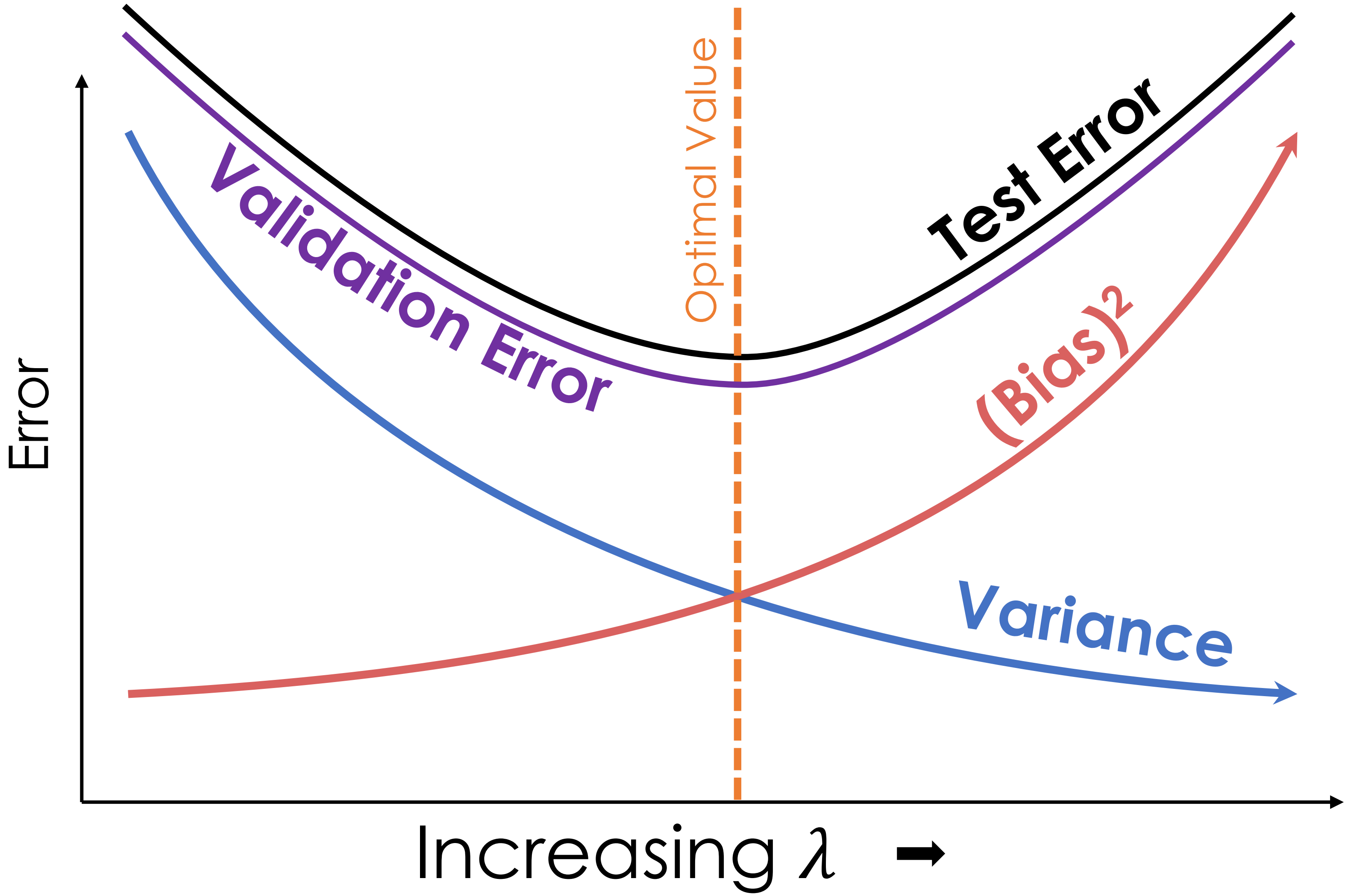
$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \textbf{Loss}\left(y_i, f_\theta(x_i)\right) \;+ \lambda R(\theta)$$

Regularization Parameter

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

# Determining the Optimal $\lambda$




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

# **Standardization** and the **Intercept** Term

$$\texttt{Height} = \theta_1 \, \texttt{age\_in\_seconds} + \theta_2 \, \texttt{weight\_in\_tons}$$

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*:

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