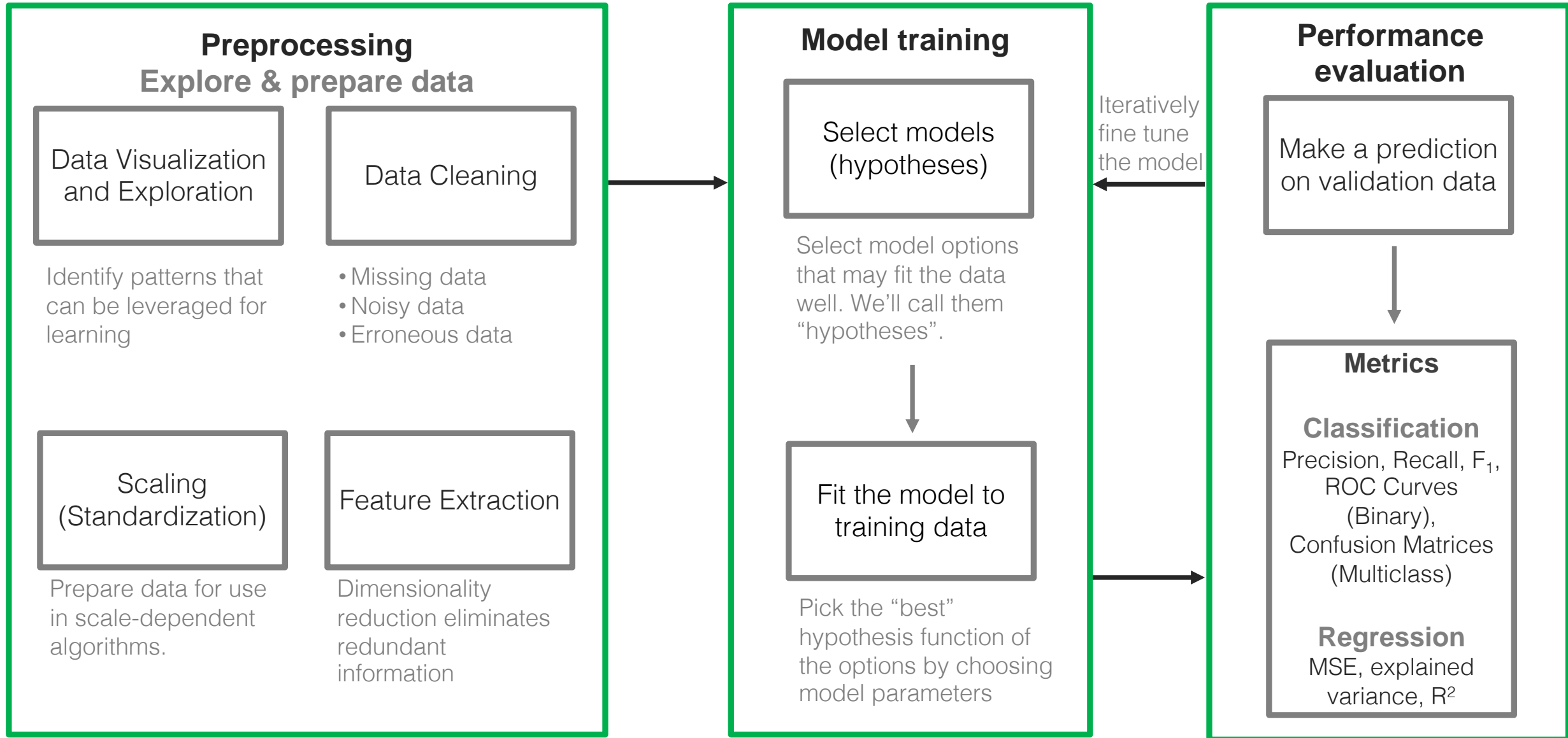
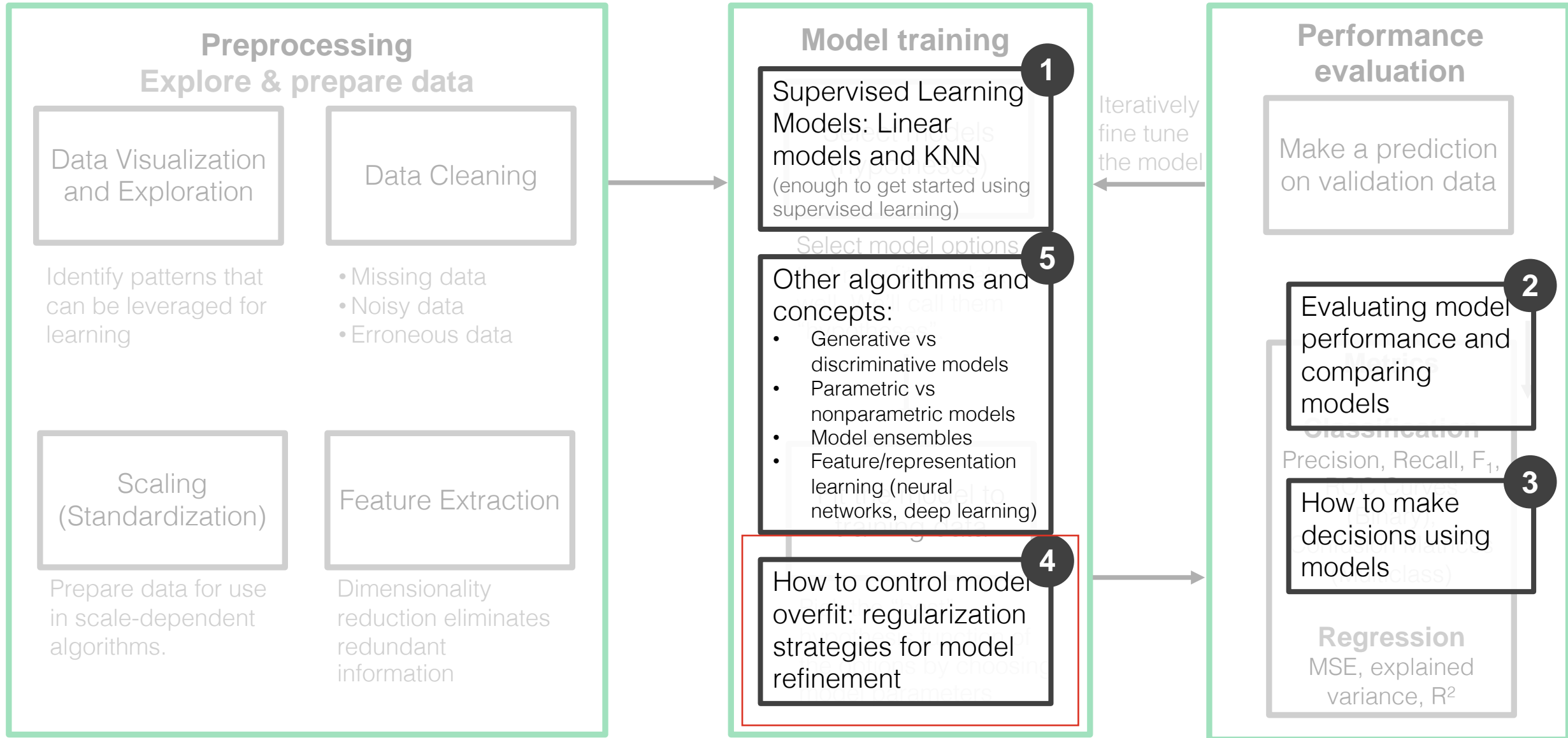


Reducing Overfit

Supervised learning in practice



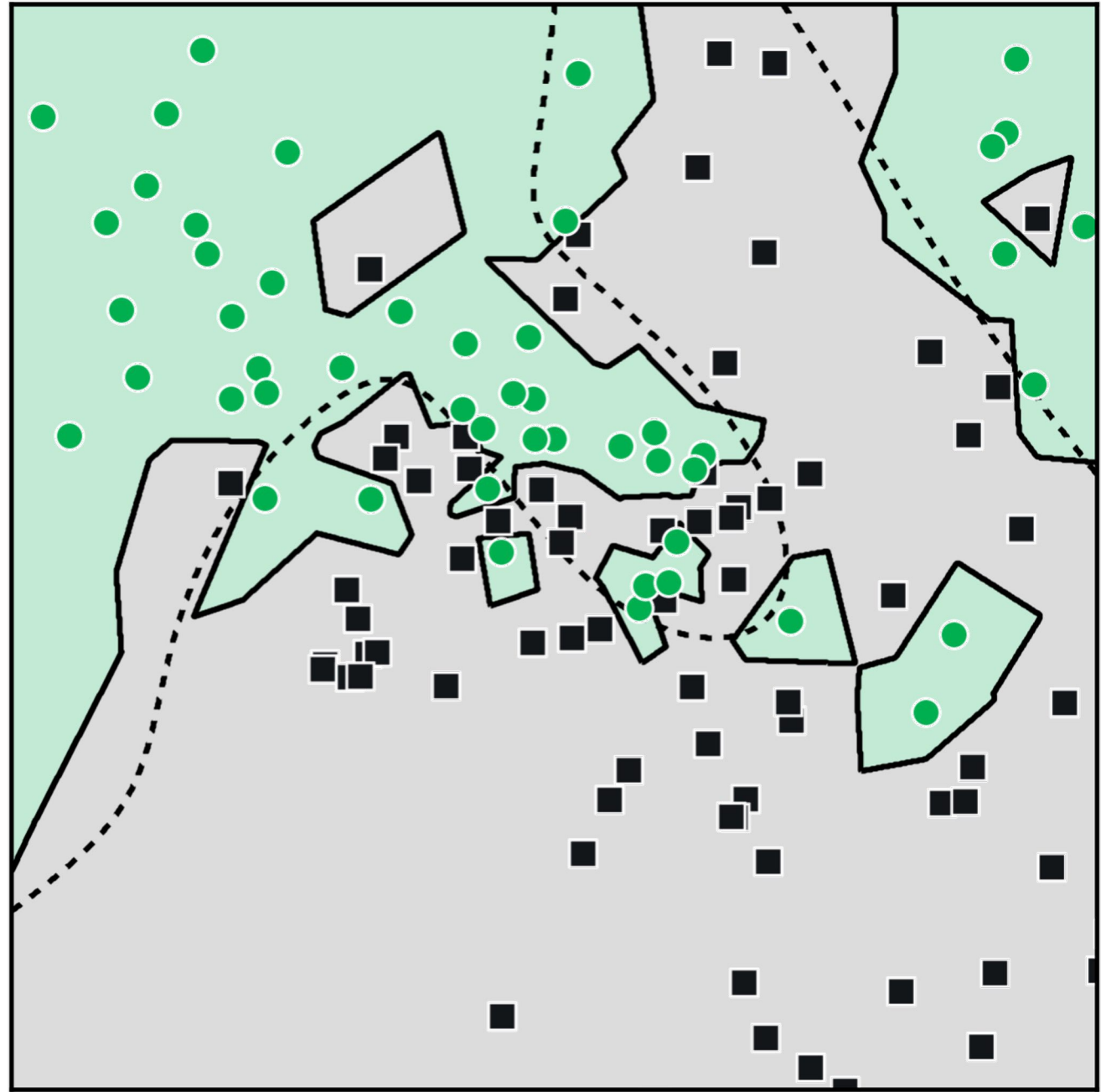
Supervised learning in practice



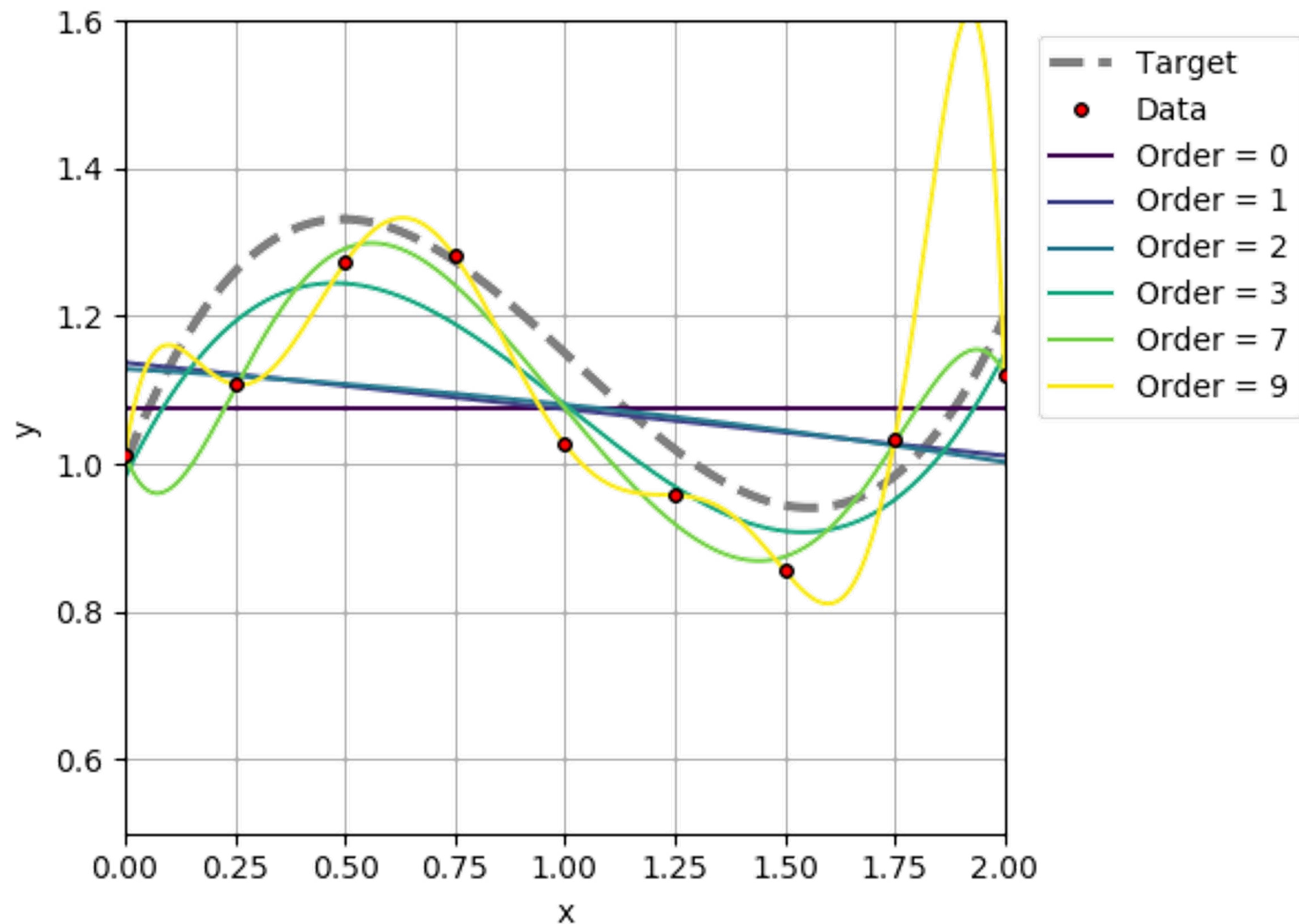
We've seen
overfit in
classification...

Overfitting to the
training data

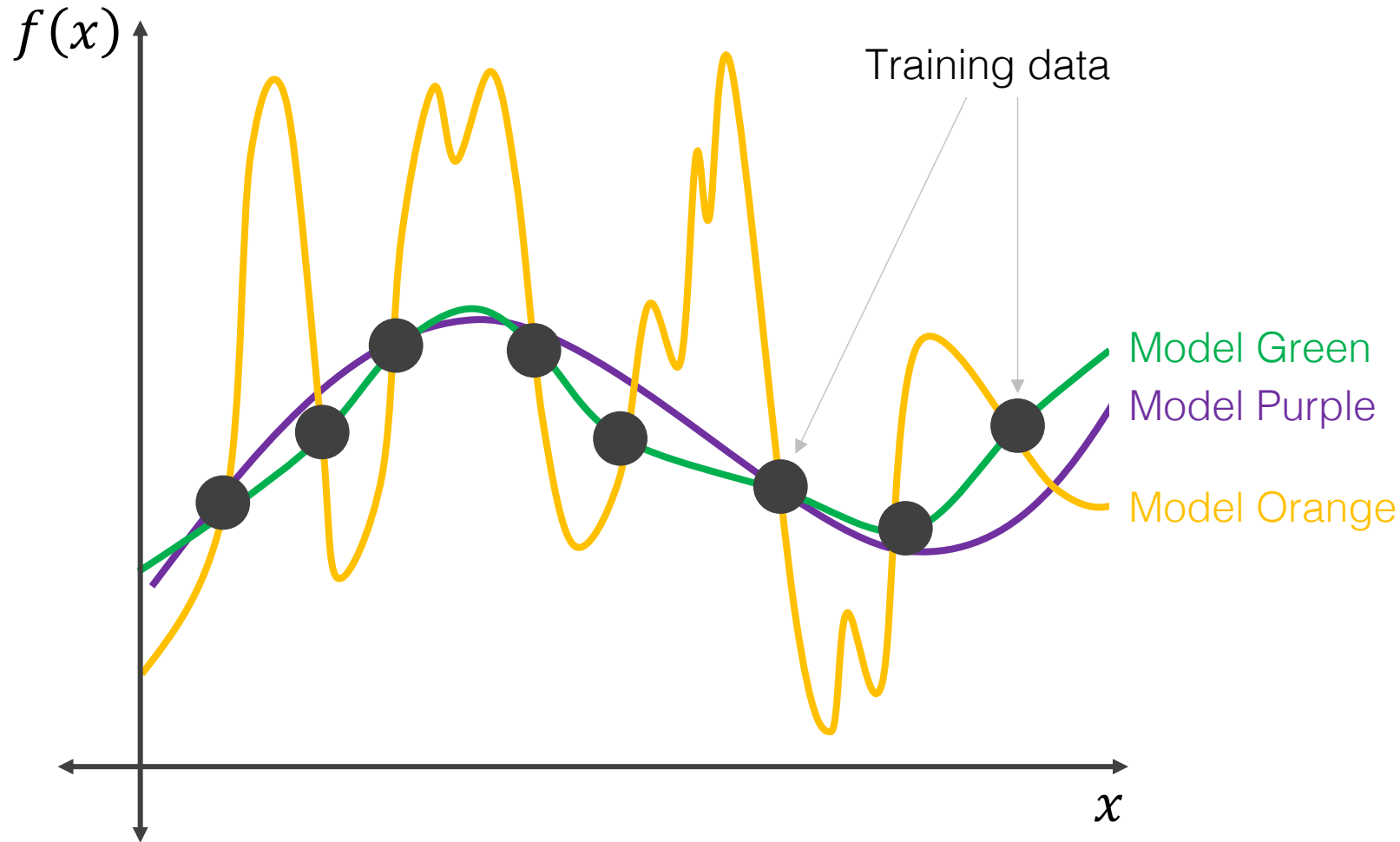
High model **variance**



...and have seen overfit in regression...



How do we limit overfitting?



How do we know which solution is best?

- Models **orange** and **green** both perfectly fit the training data
- Use which model **generalizes best** on held out data

How do enable the algorithm to find solutions that generalize better?

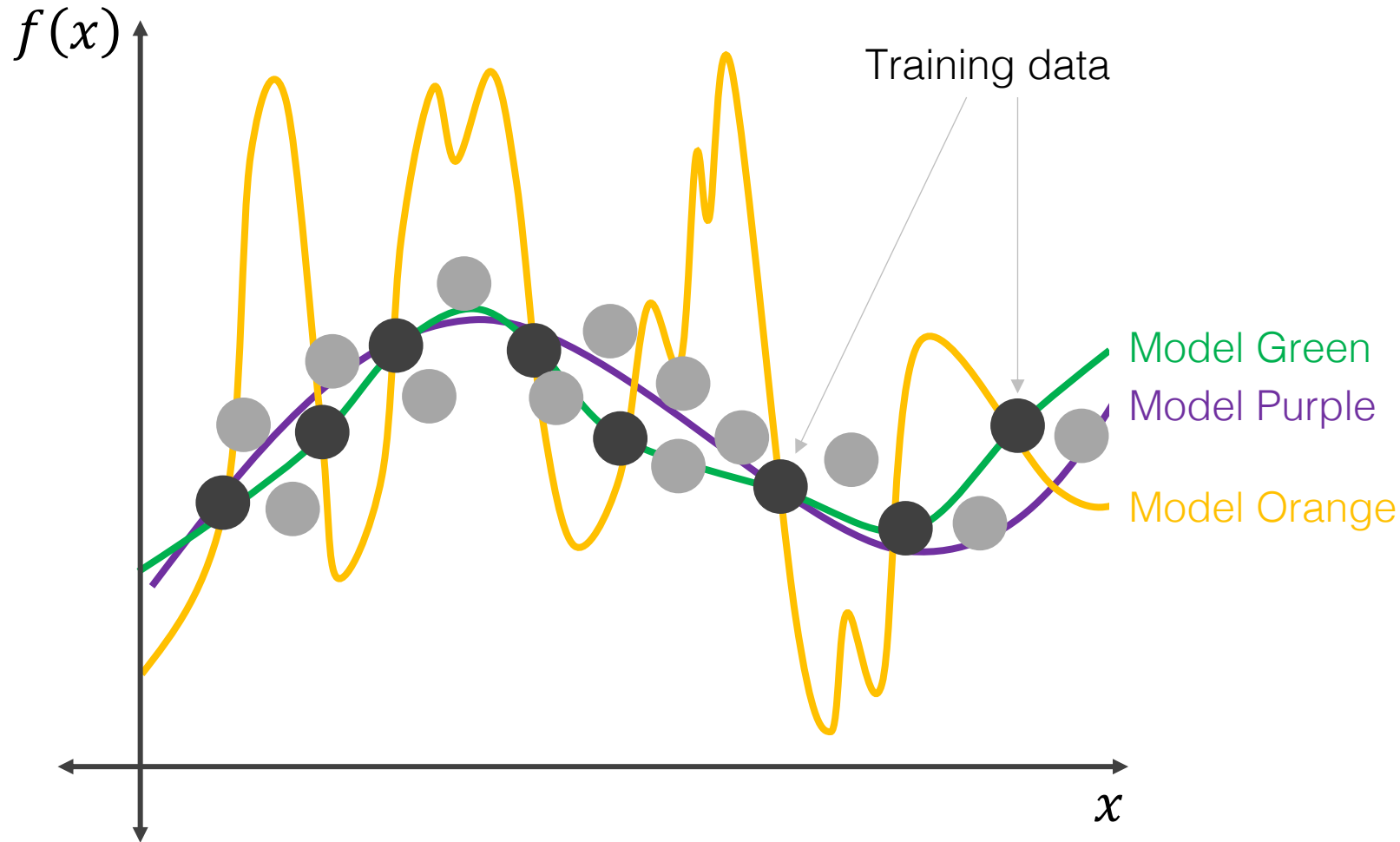
Option #1: Add more data!

(Not always possible)

cost is the biggest problem

**Option #2: Limit model flexibility to
reduce overfit**

Adding representative training data typically helps



Adding more data...

Reduces spurious correlations

“fills in” the feature space

Constrains the model to perform well on a broader set of examples

...is not always an option

How do we reduce overfit?

Option #1: Add more data!
(Not always possible)

**Option #2: Limit model flexibility to
reduce overfit**

Options for limiting model flexibility

1. Variable/feature subset selection

2. Regularization/shrinkage

3. Dimensionality reduction
(in a lecture coming soon!)

**These all reduce
the number of
features modeled
and/or model
flexibility**

Our conceptual tool...



Image from Speckyboy.com

Occam's Razor / Law of Parsimony

All else being equal, choose the **simpler** solution

Options for limiting model flexibility

1. Variable/feature subset selection

2. Regularization/shrinkage

3. Dimensionality reduction

What's the problem with adding features?

Binary classification with one feature



● Class 1
● Class 0

What's the problem with adding features?



Features that are not meaningful make the problem harder

Additional features increase flexibility in most models
(e.g. a linear model with an extra feature will have an extra parameter)

Example: what if x_2 is random noise? The model may still use it in its predictions.

Feature (variable) selection

Manual feature engineering

(e.g. use domain knowledge to remove less informative features)

Filter methods

(e.g. remove highly correlated features)

Wrapper methods

(e.g. subset selection)

try different combinations of feature selection and decide which combination works the best

Embedded methods

(e.g. LASSO regularization)

Wrapper methods for variable subset selection

Search for subsets of features that perform well

- Exhaustive search best result
- Forward selection the rest are trying to approximate exhaustive search
- Backwards selection
- Simulated annealing
- Genetic algorithms
- Particle swarm optimization

Challenge: requires rerunning the training algorithm (computationally expensive)

Forward selection

- Start with no features
- Greedily include the one feature that most improves performance
- Stop when a desired number of features is reached

Backward selection

- Start with all features included
- Greedily remove the feature that decreases performance least
- Stop when a desired number of features is reached

Challenge: requires rerunning the training algorithm (computationally expensive)

Options for limiting model flexibility

1. Variable/feature subset selection

2. Regularization/shrinkage

3. Dimensionality reduction

Regularization

Constraining a model to prevent overfitting or solve an ill-posed problem
(does not have a unique solution)

Regularization

a.k.a. shrinkage

by adding the regularization penalty term

Adjust the **cost/loss function** to penalize larger parameters

$$C(\mathbf{w}) = E(\mathbf{w}, \mathbf{X}, \mathbf{y}) + \lambda R(\mathbf{w})$$

E is our cost / loss function:

- MSE for regression
- Cross entropy/log loss for classification

For regression:

$$C(\mathbf{w}) = \sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \lambda \sum_{j=1}^p w_j^2$$

penalty minimized when abs weights are minimized

Here \hat{f} is our model.
For linear regression,

$$\hat{f}(\mathbf{x}_i, \mathbf{w}) = \mathbf{w}^T \mathbf{x}_i$$

Square error

regression --> Least squared error
classification --> cross-entropy or least

L_2 regularization penalty

This term causes the estimated parameter values to “shrink”

Regularization

a.k.a. shrinkage

Here we assume a regression example

Adjust the **cost/loss function** to penalize larger parameter values

a.k.a....

L_2 regularization	$C(\mathbf{w}) = \sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \lambda \sum_{j=1}^p w_j^2$	ridge regression or weight decay (Tikhonov regularization)
L_1 regularization	$C(\mathbf{w}) = \sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \lambda \sum_{j=1}^p w_j $	least absolute shrinkage and selection operator (LASSO)
L_2 & L_1 regularization	$C(\mathbf{w}) = \sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \lambda_1 \sum_{j=1}^p w_j + \lambda_2 \sum_{j=1}^p w_j^2$	elastic net regularization

To explain how regularization works, we need to know our Norms



Normal Rockwell (painter)



Norm Macdonald (comedian)

...other norms



Norman Borlaug (agronomist)



Norm Peterson (character on Cheers)

Images from Wikipedia, Norm MacDonald photo by playerx licensed under CC BY 2.0

Norm

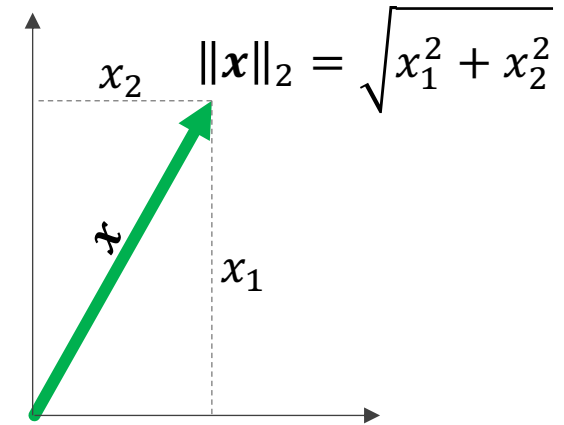
A function that assigns a positive **length or size** to a vector

The most familiar is likely the **Euclidean**, or L_2 norm:

$$\|\mathbf{x}\|_2 \triangleq \sqrt{x_1^2 + \dots + x_n^2} = \left(\sum_{i=1}^n x_i^2 \right)^{\frac{1}{2}} = \sqrt{\mathbf{x}^T \mathbf{x}}$$

You'll often see this in its squared form:

$$\|\mathbf{x}\|_2^2 \triangleq x_1^2 + \dots + x_n^2 = \sum_{i=1}^n x_i^2 = \mathbf{x}^T \mathbf{x}$$

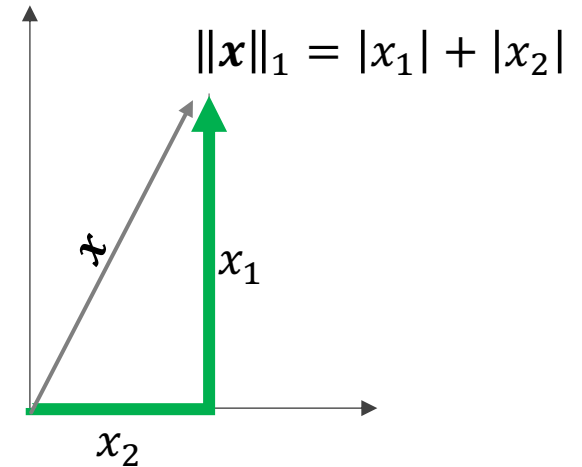


Norms

There's also the **L_1 norm**

(a.k.a taxicab or Manhattan distance)

$$\|\mathbf{x}\|_1 \triangleq |x_1| + \cdots + |x_n| = \sum_{i=1}^n |x_i|$$



The general **L_p norm**:

$$\|\mathbf{x}\|_p \triangleq \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}}$$

In the limit, the **infinity norm** is the maximum entry of the vector \mathbf{x} :

$$\|\mathbf{x}\|_\infty \triangleq \max_i |x_i|$$

Norms for a vector

Assume a 2-D vector: $\mathbf{w} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$

$$\|\mathbf{w}\|_1 = |w_1| + |w_2|$$

$$= |1| + |3|$$

$$= 4$$

$$\|\mathbf{w}\|_2 = \sqrt{w_1^2 + w_2^2}$$

$$= \sqrt{1^2 + 3^2}$$

$$= \sqrt{10} \approx 3.2$$

$$\|\mathbf{w}\|_\infty = \max_i |w_i|$$

$$= 3$$

Unit Norms

Assume a 2-D vector: $\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$

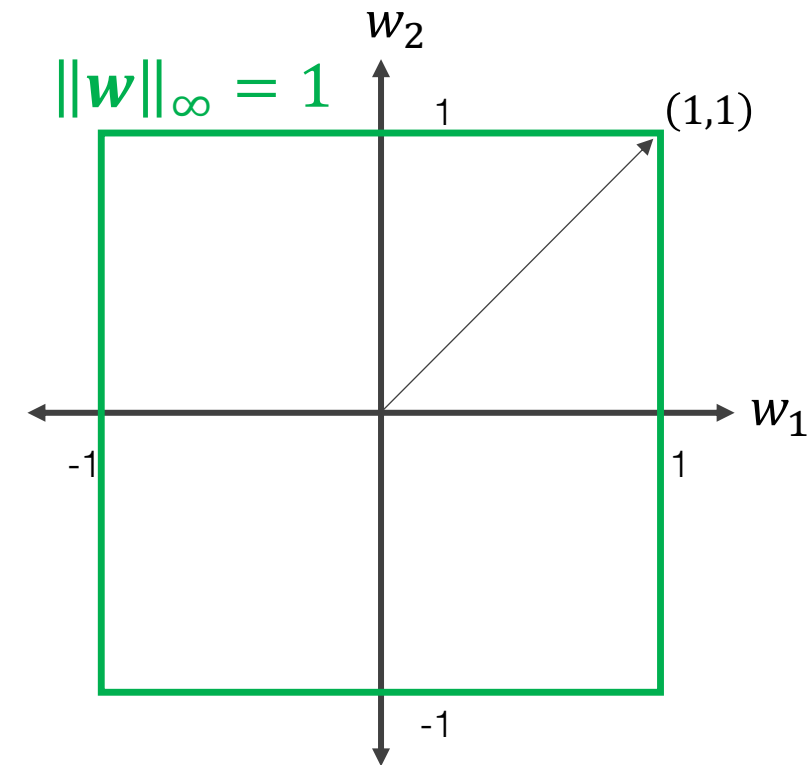
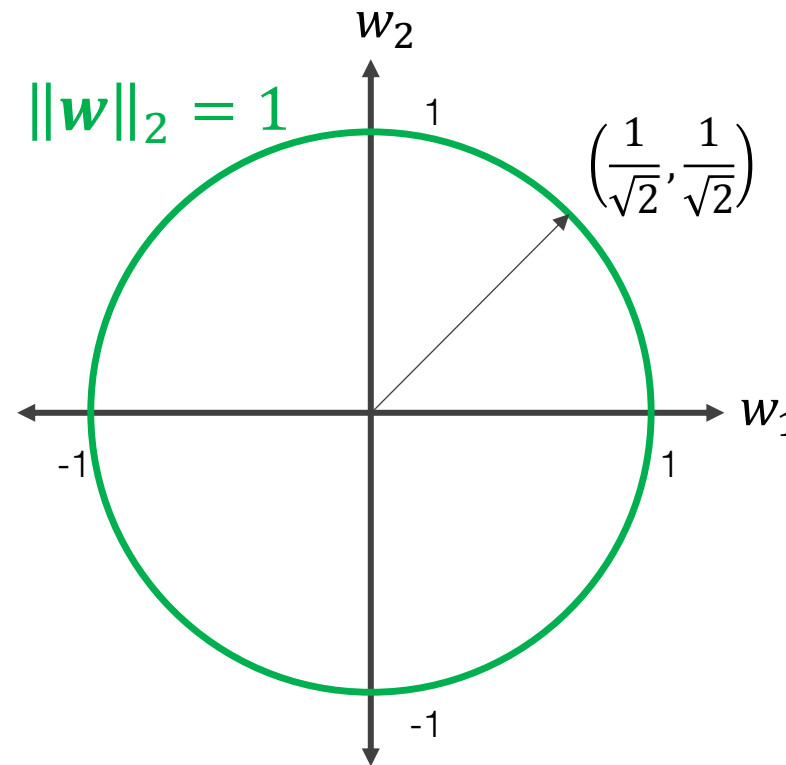
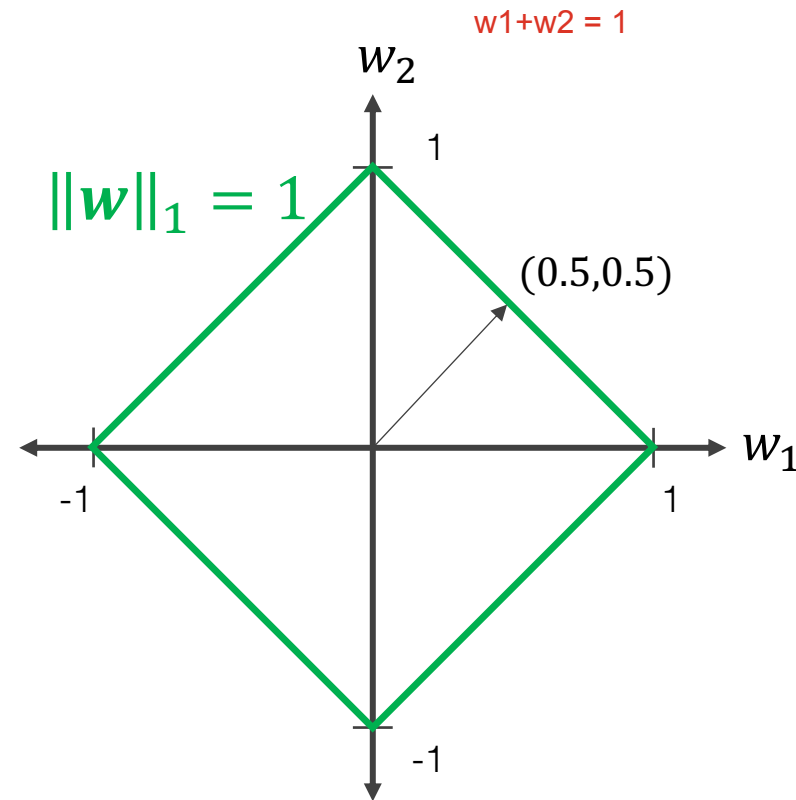
All possible values of \mathbf{w} that have a norm of 1
(Plotted as the green lines below)

$$\|\mathbf{w}\|_1 = \sum_{i=1}^n |w_i|$$

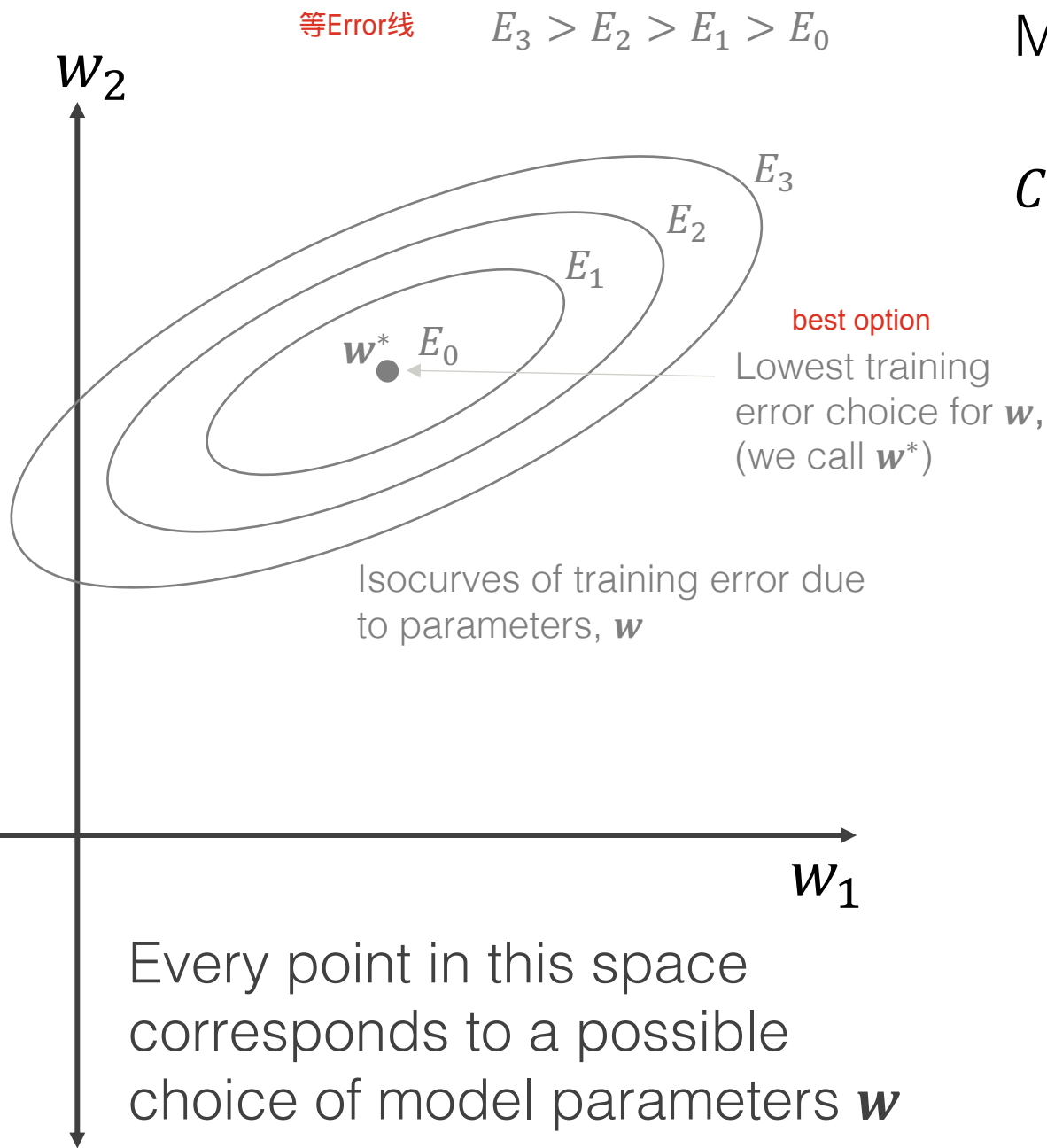
focus on these two only

$$\|\mathbf{w}\|_2 = \sqrt{w_1^2 + w_2^2}$$

$$\|\mathbf{w}\|_\infty = \max_i |w_i|$$



$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

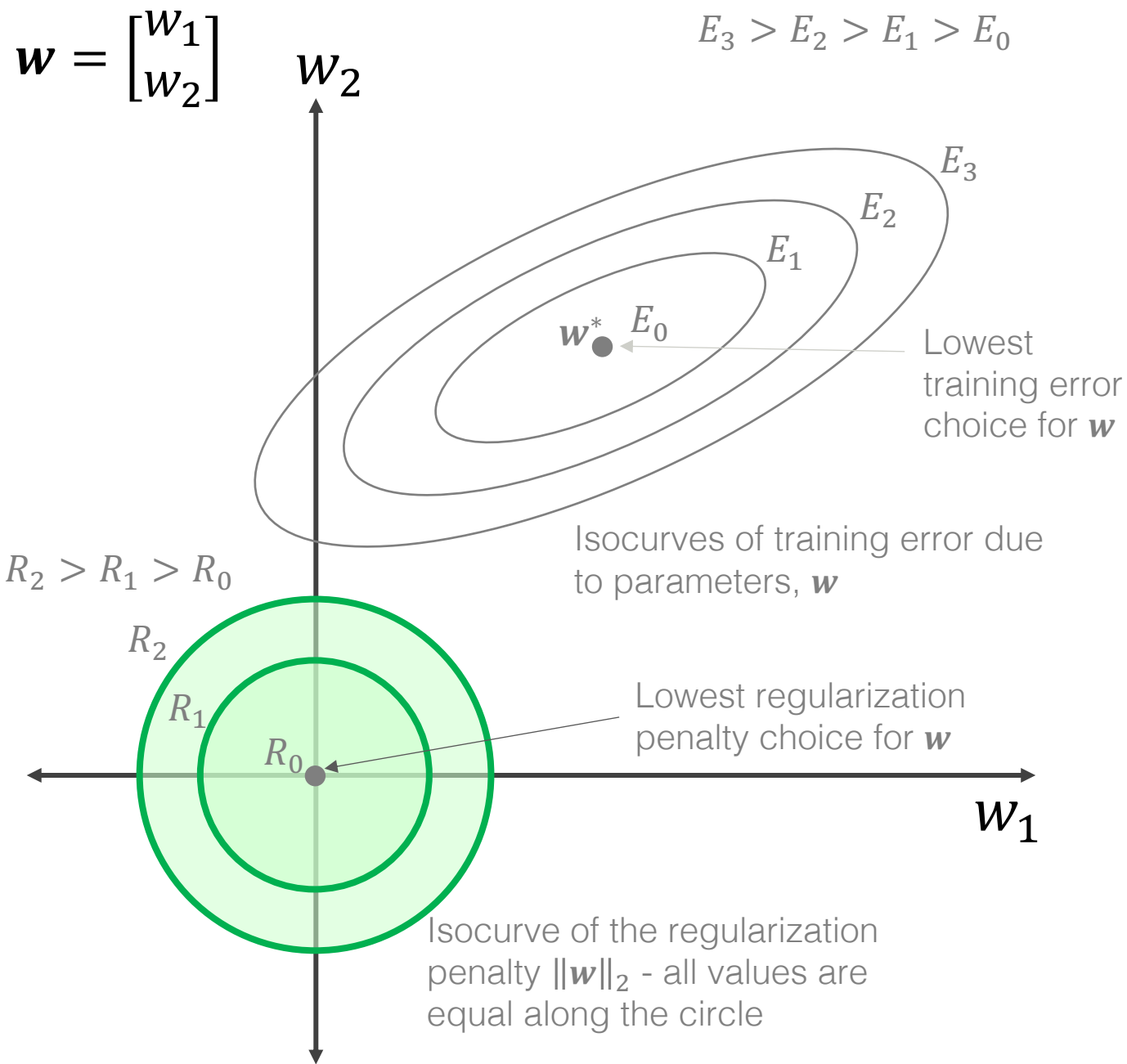


Minimize cost function:

$$C(\mathbf{w}) = \sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

Training error term (E)

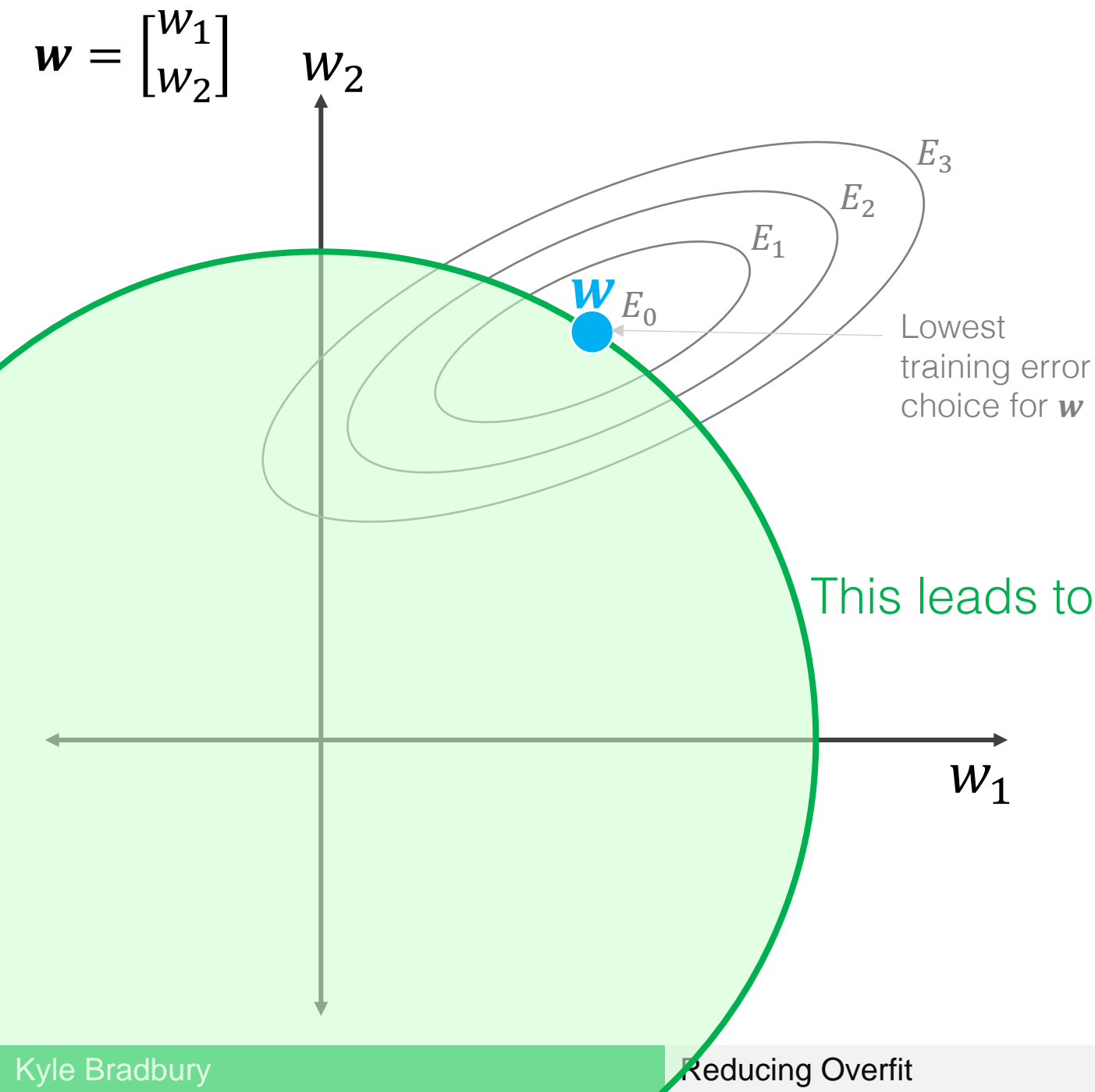
First, let's just minimize training error



Minimize cost function:

$$C(\mathbf{w}) = \underbrace{\sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2}_{\text{Training error term (E)}} + \underbrace{\lambda \sum_{j=1}^p w_j^2}_{\text{Regularization penalty (R)}}$$

Next, let's add a regularization penalty

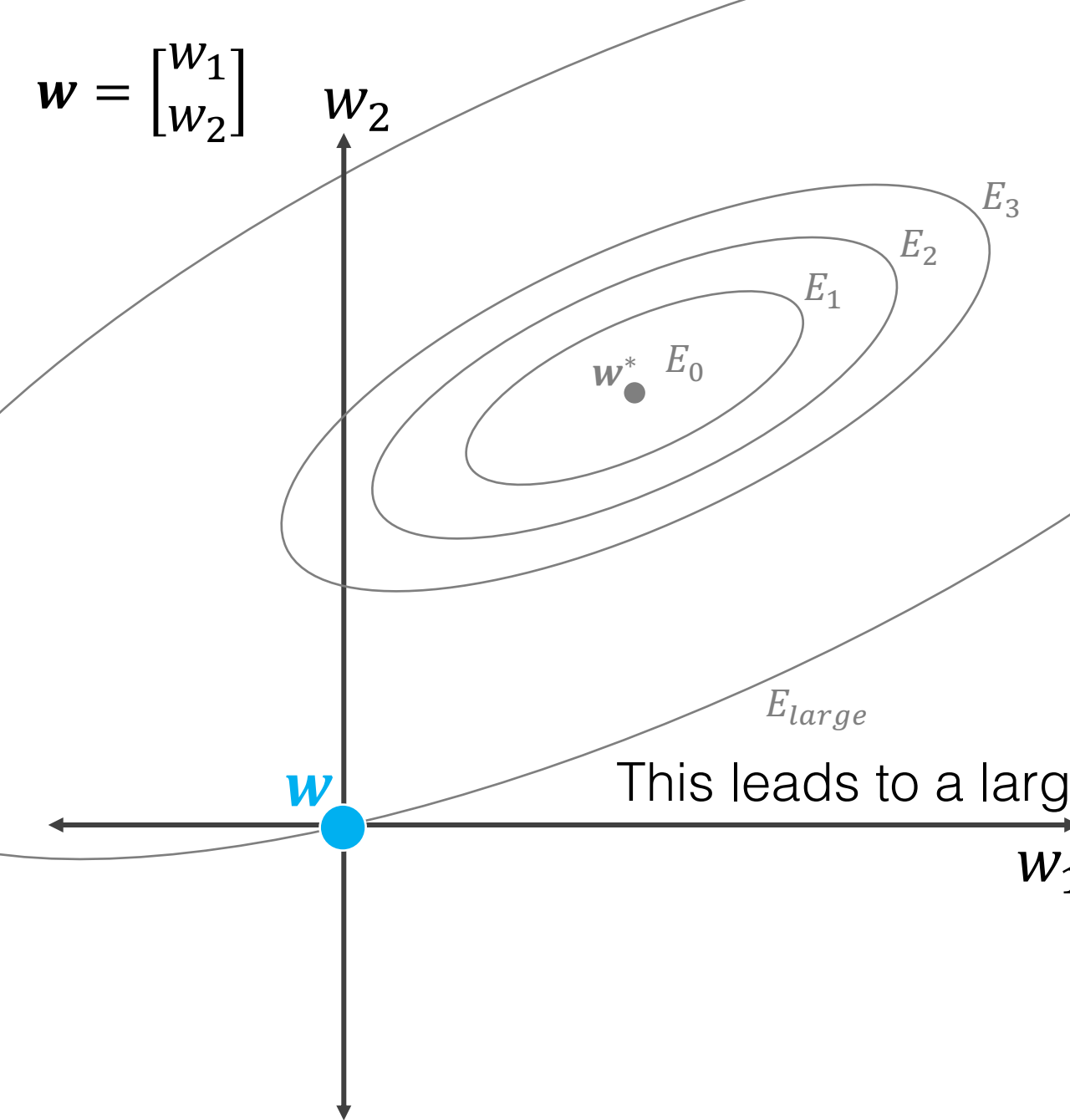


Minimize cost function:

$$C(\mathbf{w}) = \underbrace{\sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2}_{\text{Training error}} + \underbrace{\lambda \sum_{j=1}^p w_j^2}_{\text{Regularization penalty}}$$

Small **Large**

Now if we **only minimize training error**, the regularization penalty is large



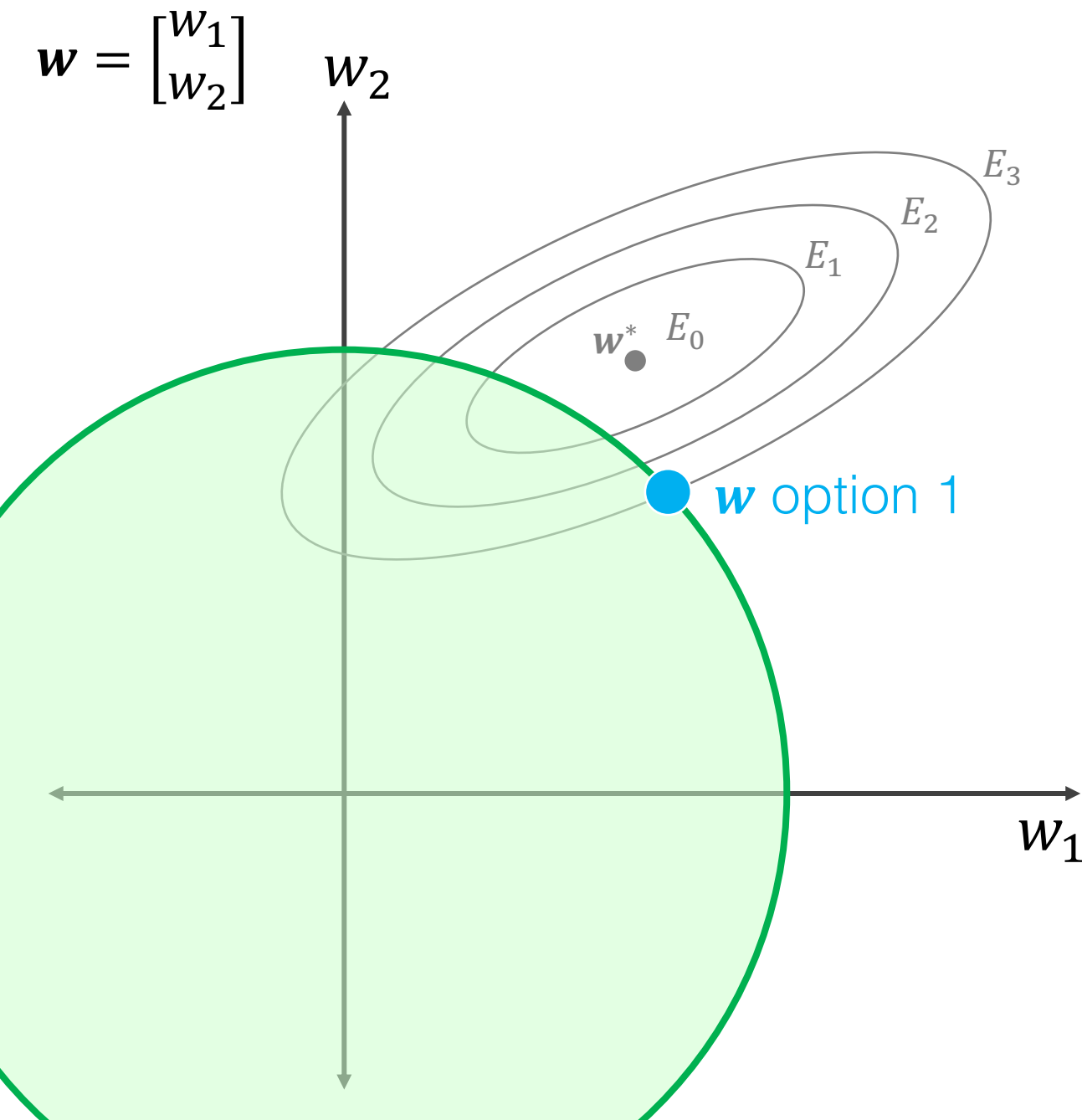
Minimize cost function:

$$C(\mathbf{w}) = \underbrace{\sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2}_{\text{Training error}} + \underbrace{\lambda \sum_{j=1}^p w_j^2}_{\text{Regularization penalty}}$$

Large **Small**

This leads to a large training error term...

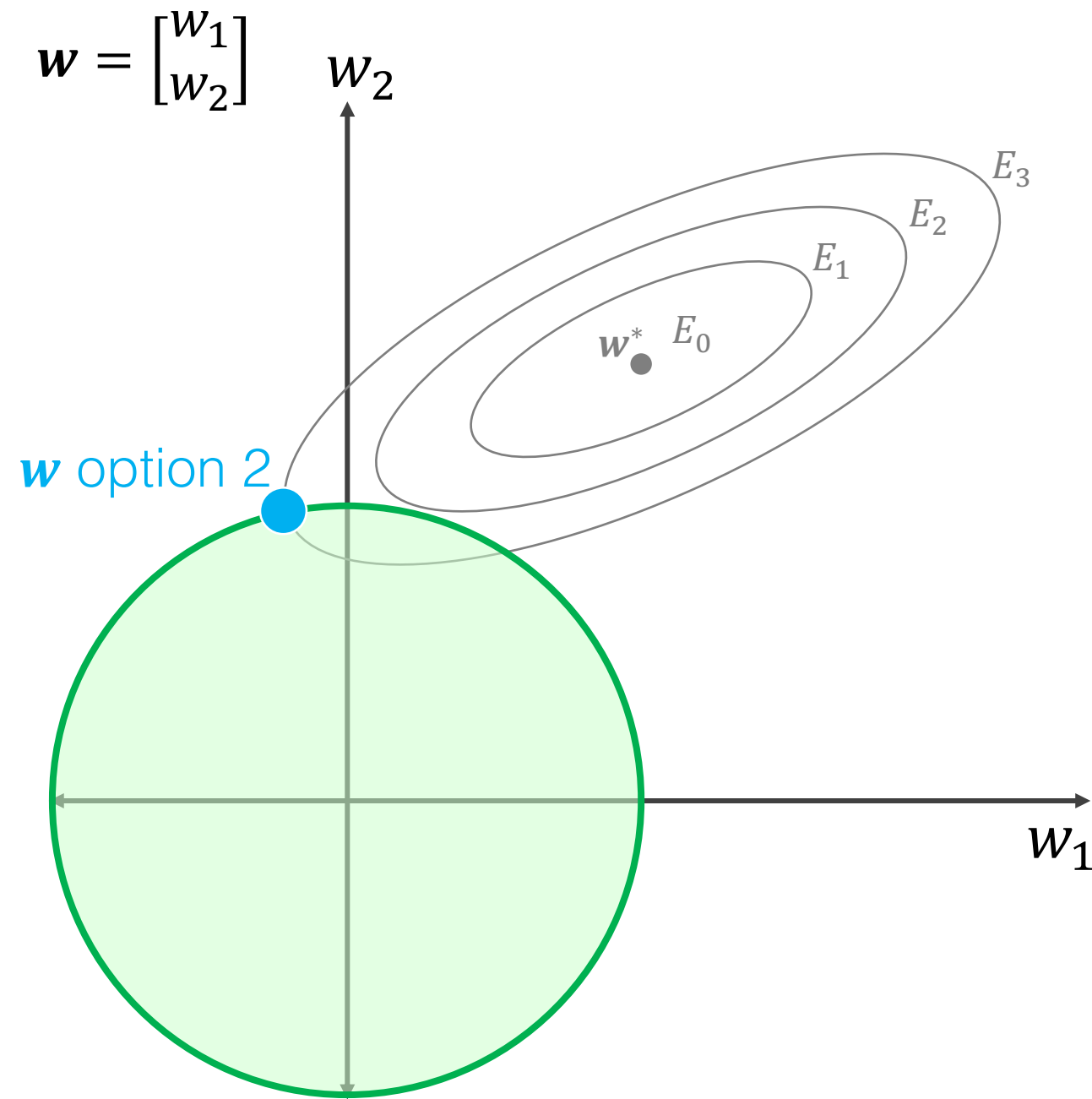
Now if we **only minimize regularization penalty**, the training error is large



Minimize cost function:

$$C(\mathbf{w}) = \underbrace{\sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2}_{\text{Training error}} + \underbrace{\lambda \sum_{j=1}^p w_j^2}_{\text{Regularization penalty}}$$

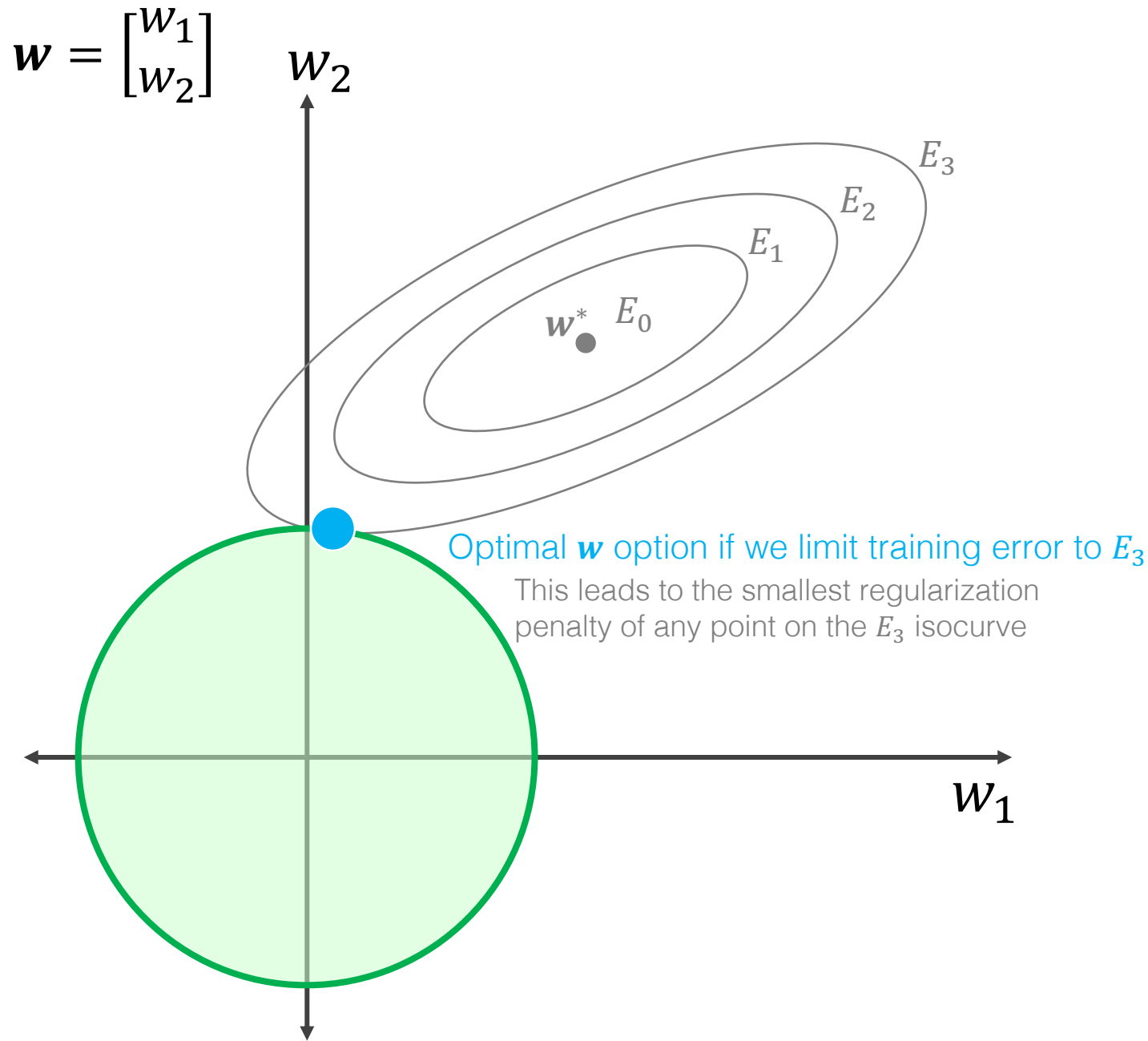
For any level of training error (assume E_3 here), there may be many parameter values that result in an equivalent training error



Minimize cost function:

$$C(\mathbf{w}) = \underbrace{\sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2}_{\text{Training error}} + \underbrace{\lambda \sum_{j=1}^p w_j^2}_{\text{Regularization penalty}}$$

For any level of training error (assume E_3 here), there may be many parameter values that result in an equivalent training error

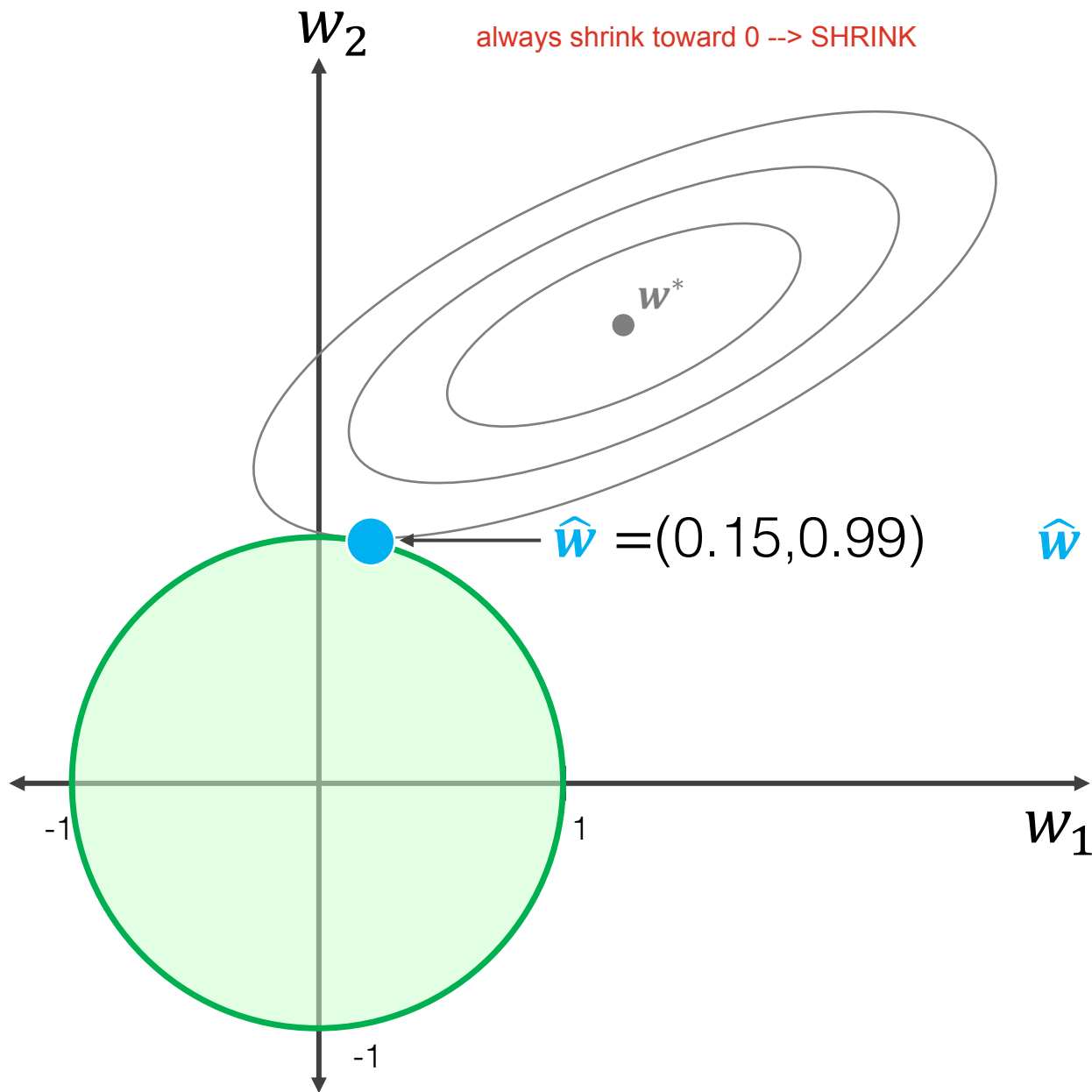


Minimize cost function:

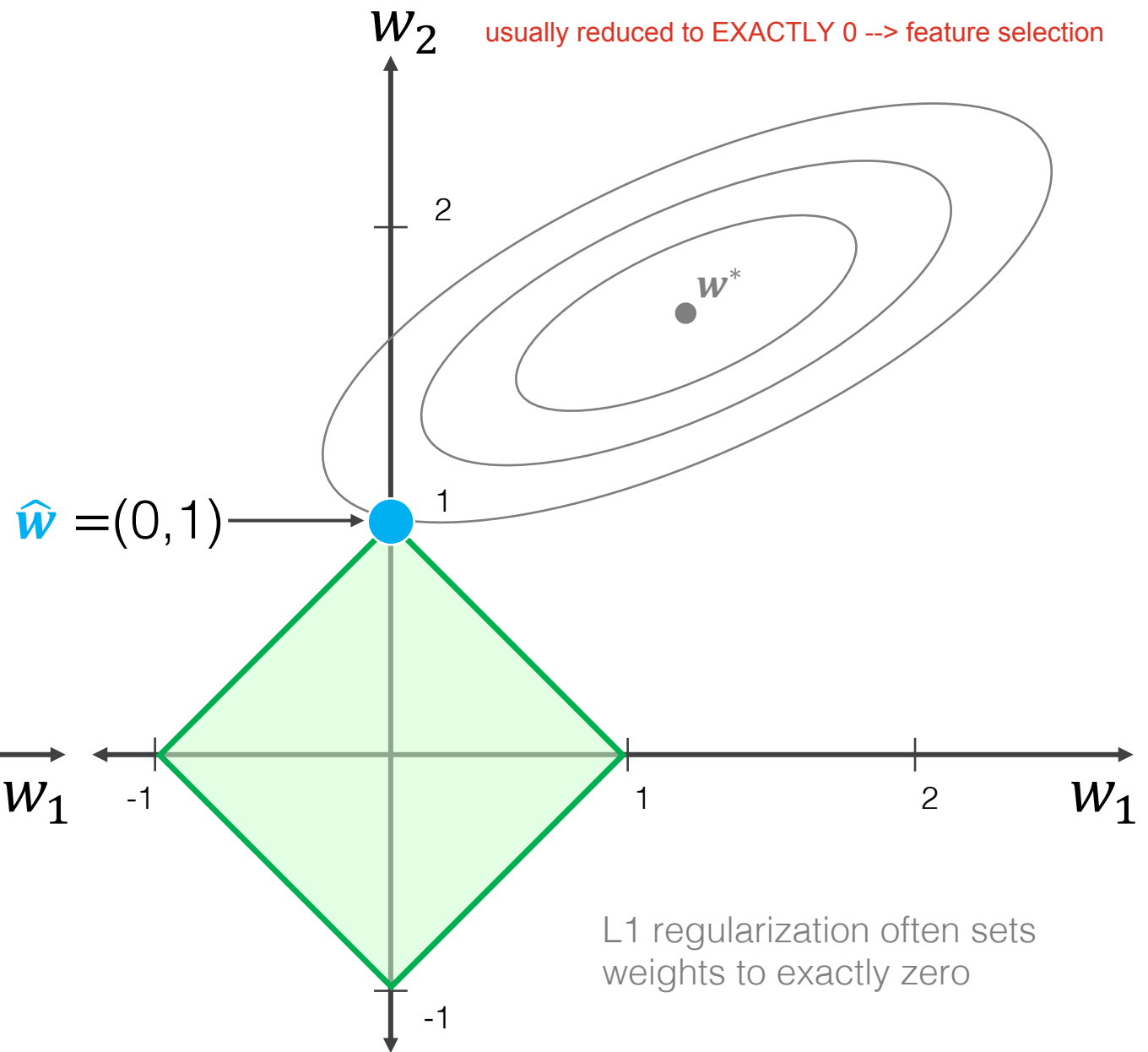
$$C(\mathbf{w}) = \underbrace{\sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2}_{\text{Training error}} + \underbrace{\lambda \sum_{j=1}^p w_j^2}_{\text{Regularization penalty}}$$

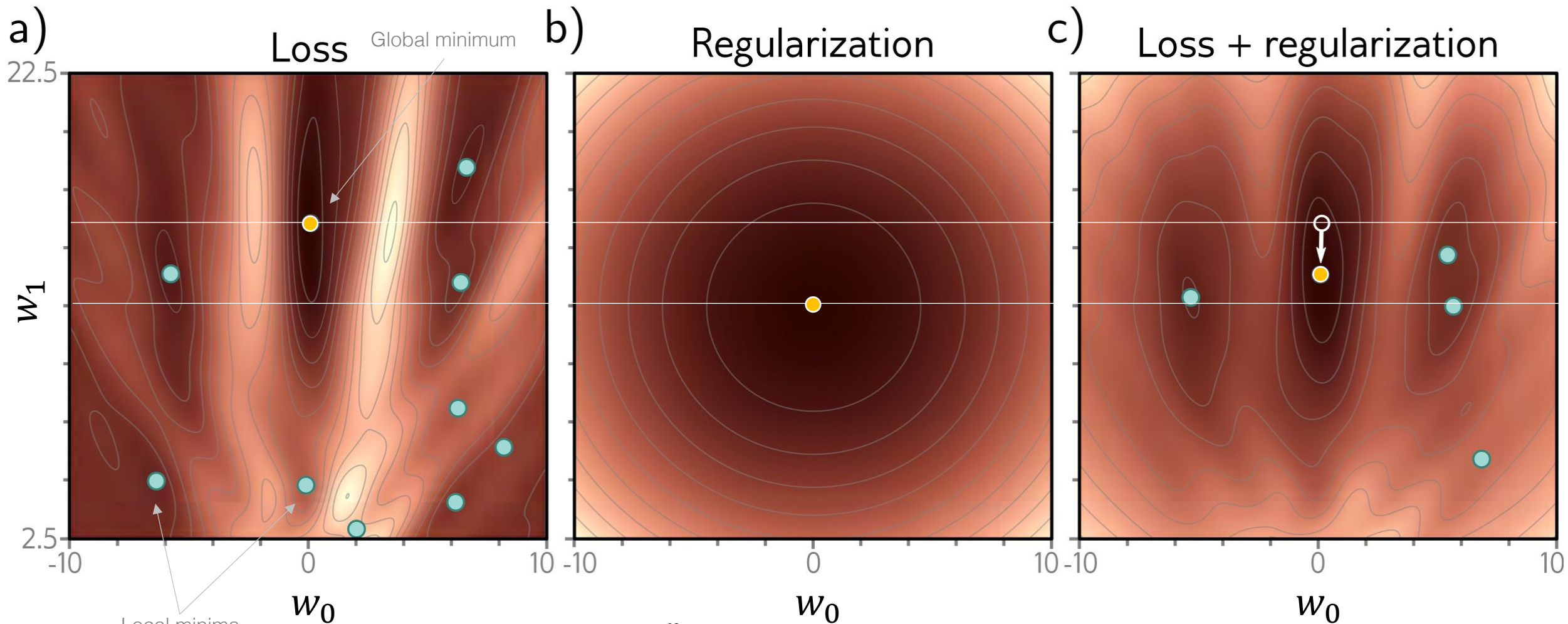
However, we can choose between the options by minimizing the regularization penalty

Ridge: L_2 regularization



LASSO: L_1 regularization





$$C(\mathbf{w}) = \underbrace{\sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2}_{\text{Training error (a.k.a. loss)}} + \underbrace{\lambda \sum_{j=1}^p w_j^2}_{\text{Regularization penalty}}$$

**Visualizing
Regularization in
Action**

c

a

b

Regularization reduces variance

Leads to smaller model parameters

L_1 regularization also performs variable selection

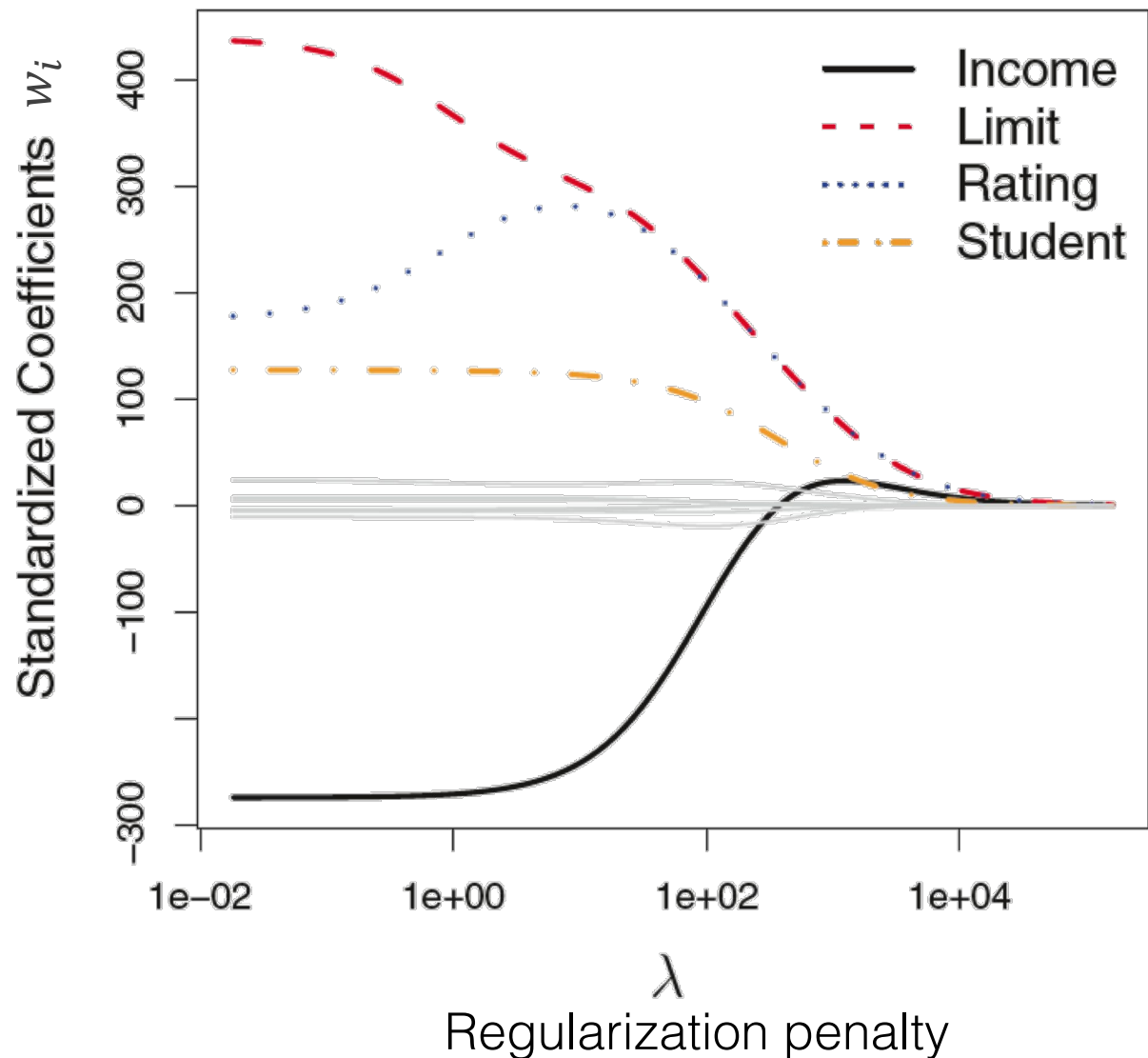
Example: predicting credit default

11 features to use to predict default:

- Income
- Credit limit
- Credit rating
- Credit balance
- Number of credit cards
- Age
- Education
- Gender
- Student status
- Ethnicity
- Marriage status

L_2 regularization

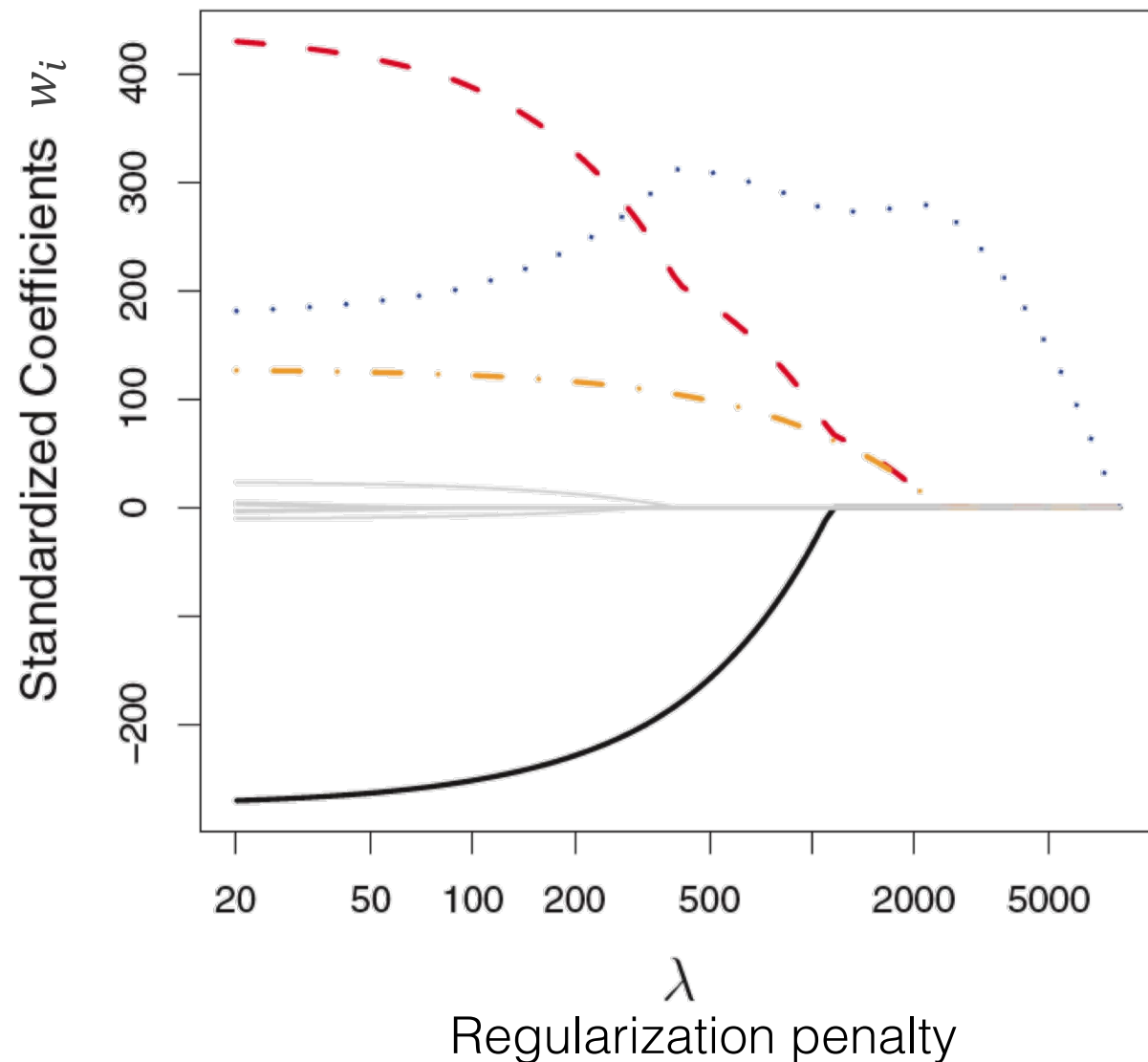
Ridge regression



everything goes towards 0

L_1 regularization

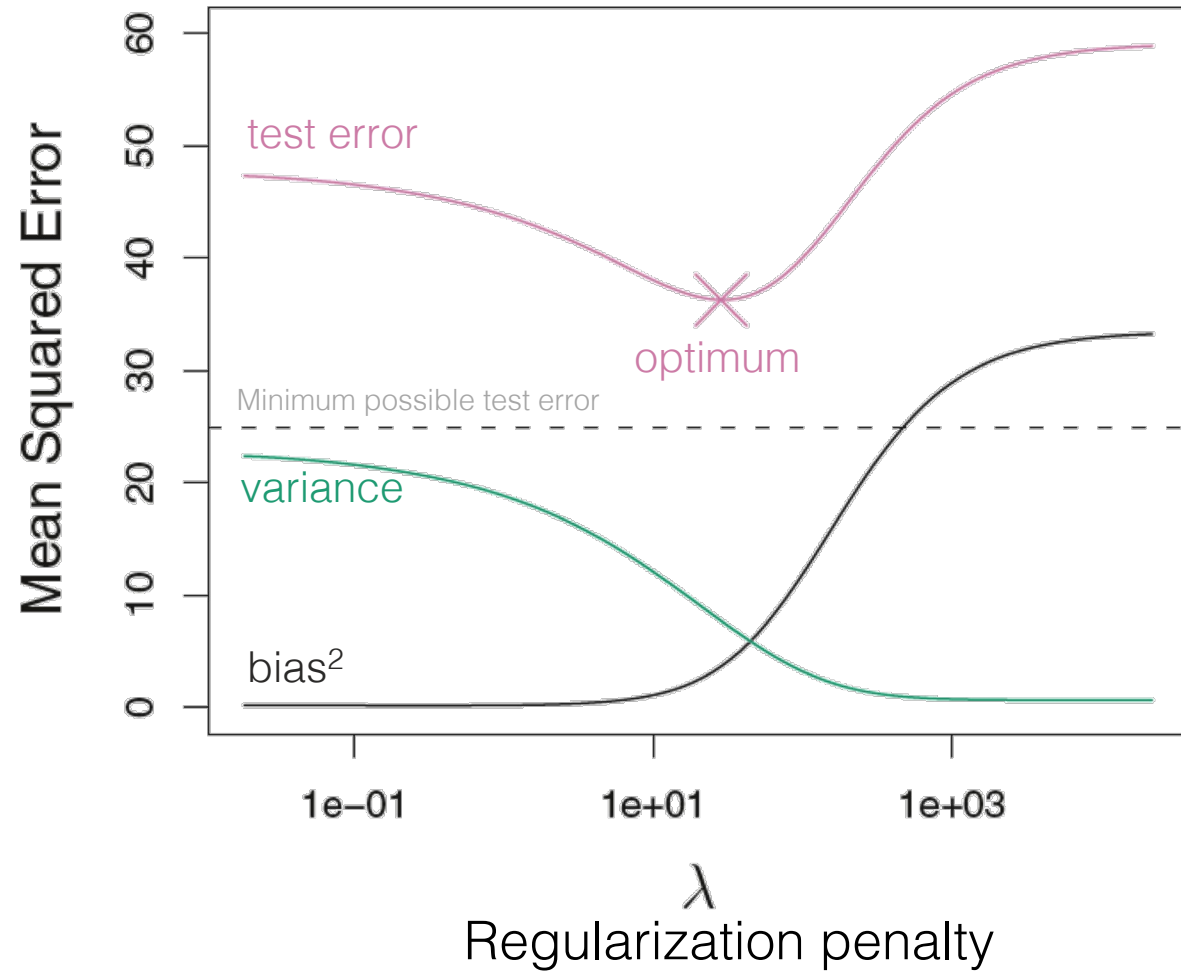
LASSO regularization



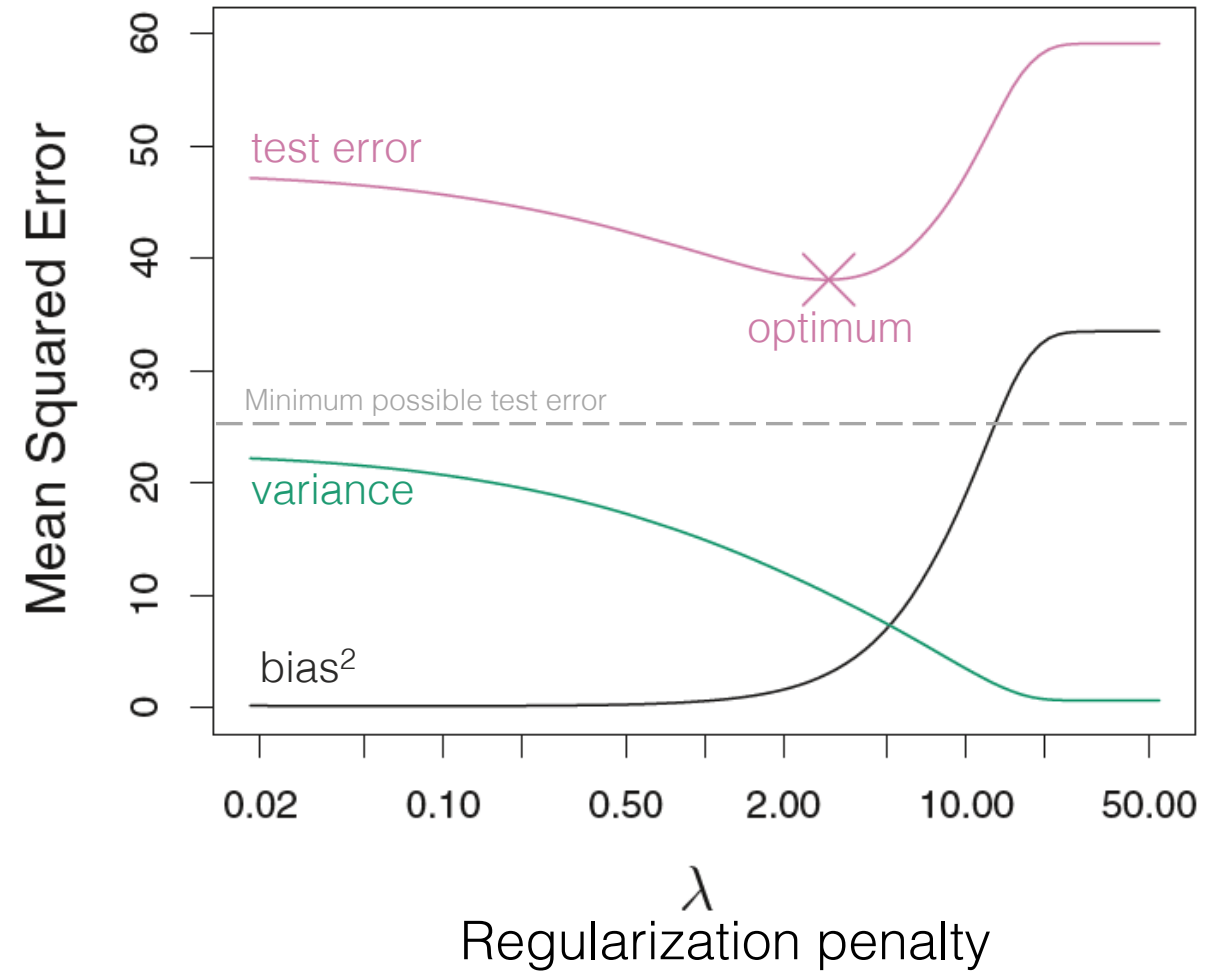
some may hit exactly 0

Images from James et al., An Introduction to Statistical Learning

L_2 regularization



L_1 regularization



Underdetermined systems and OLS

$$X = \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix}$$

Number of features p

N
Number of samples

If $p > N$, then the system is **underdetermined**

Often means there are infinitely many solutions

Ridge regression makes this problem solvable

Choosing the regularization parameter λ

- λ is a hyperparameter
- Use a training, validation, and test set
- Can also apply nested cross validation

Train

Used for model training / fitting

Validation

Used to optimize
hyperparameters

Test

Used to evaluate
generalization
performance

Strengths of L_1 and L_2 regularization

Ridge regression (L_2 regularization) handles **multicollinearity** well

LASSO regularization (L_1 regularization) reduces the number of predictors in a model (yields **sparse** models)

You can use a little of both via elastic net regularization

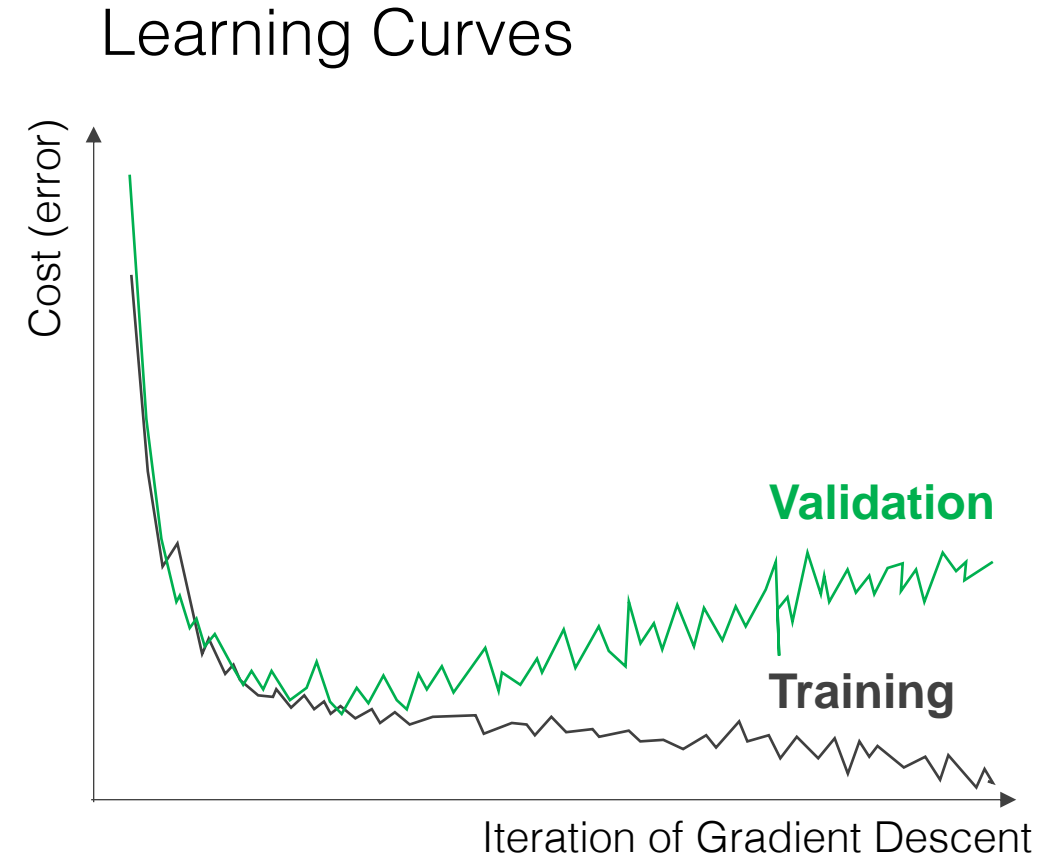
These approaches can be easily added to many cost functions

One more approach: Early Stopping

Iterative learning (training) methods, (e.g. gradient descent) tend to learn more complex models over time

Stop the fitting process earlier, before overfit has occurred

Common in neural network training



Takeaways

Reducing the number of features in a model may improve generalization error by reducing overfit

Overly flexible models can be regularized to reduce overfit (reducing variance)

L_1 and L_2 regularization are effective tools for battling overfit