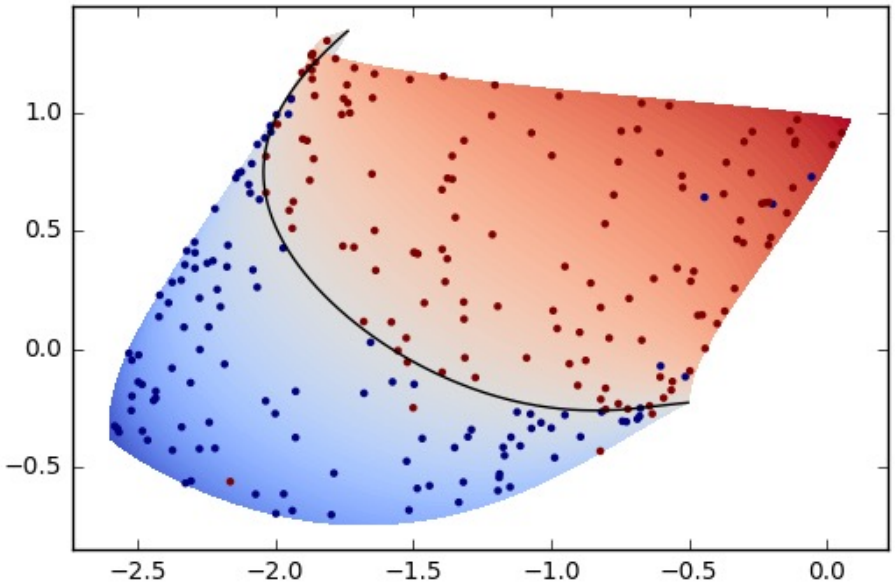
Lecture 5: Model Selection



Gavin Kerrigan

Spring 2023

Some materials courtesy Padhraic Smyth, Alex Ihler.

Announcements

- Discussion sections tomorrow
  - Matplotlib and Scikit-Learn
  - Come prepared with questions on lecture / homework
- HW1 due Friday
  - Due 11:59PM – no late submissions
  - Submit PDF via Gradescope
  - "Statement of Collaboration"
    - If you completed the HW on your own, state this
- HW2 released early next week

Today's Lecture

Model Selection

Cross Validation

Regularization

Reminders on Nonlinear Regression

Consider a feature vector with d features

$$\mathbf{x} = (x_1, x_2, \ldots, x_d)$$

The **polynomial feature expansion** of $\boldsymbol{x}$ in degree p is

$$\boldsymbol{z} = \phi_p(\boldsymbol{x}) = (1, x_1, x_2, \ldots, x_d, x_1^2, x_2^2, \ldots, x_d^2, \ldots, x_1^p, \ldots, x_d^p, \ldots, x_2 x_3^{p-1}, \ldots)$$

New feature vector consisting of all possible feature combinations of degree $\leq p$

Reminders on Nonlinear Regression

Fitting a polynomial to data:

1. Fix a polynomial degree p
2. Compute the feature expansions $\boldsymbol{z}_i = \phi_p(\boldsymbol{x}_i)$ for every datapoint $i = 0, 1, 2, \ldots, n$

sklearn.preprocessing.PolynomialFeatures

class sklearn.preprocessing.**PolynomialFeatures**(*degree=2, *, interaction_only=False, include_bias=True, order='C'*)
[source]

3. Fit a linear regression model with the *new features* $\boldsymbol{z}_i$

sklearn.linear_model.LinearRegression

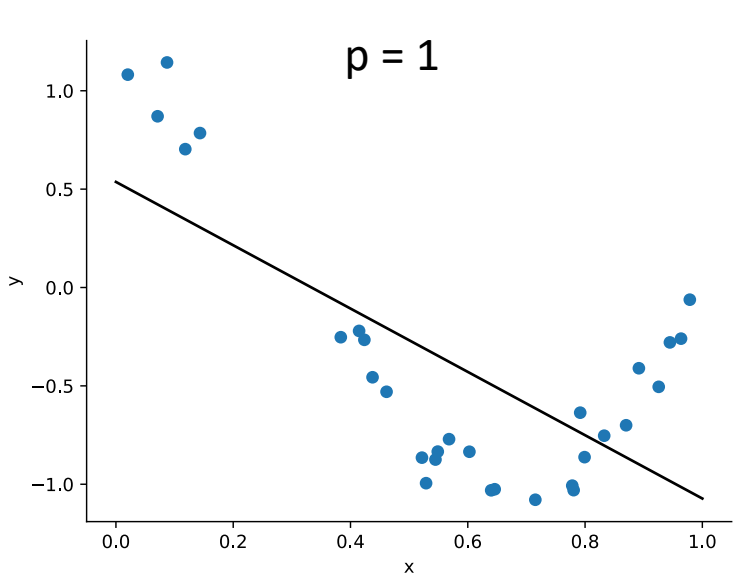
class sklearn.linear_model.**LinearRegression**(**, fit_intercept=True, copy_X=True, n_jobs=None, positive=False*)
[source]

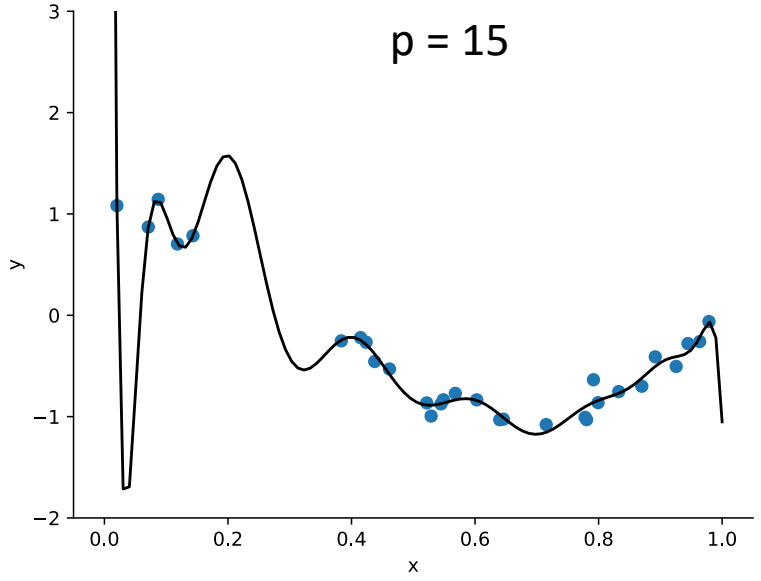
Overfitting and Underfitting

The complexity of our model should "match" the complexity of our data

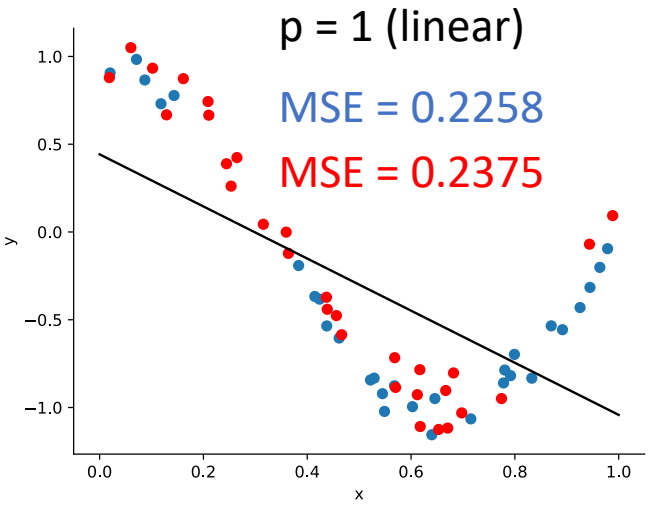
Underfitting: model is too simple to appropriately capture the data

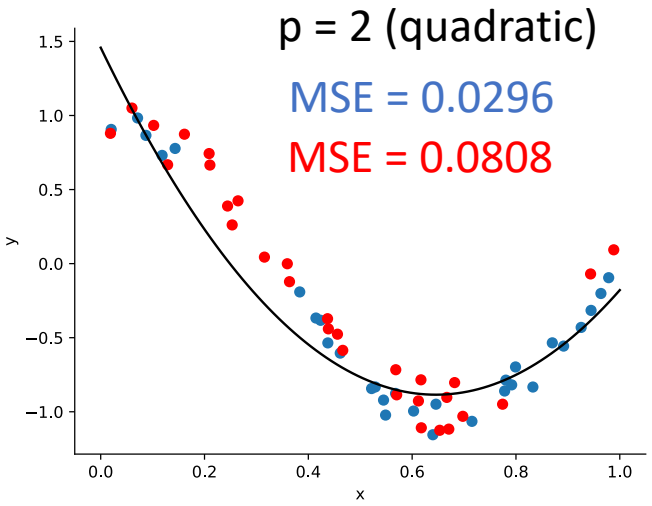
Overfitting: model is too complex; fits data too well and cannot generalize to data not seen during training

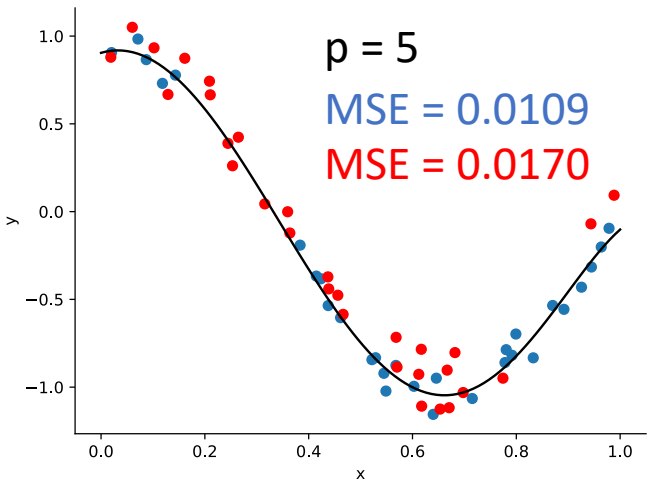


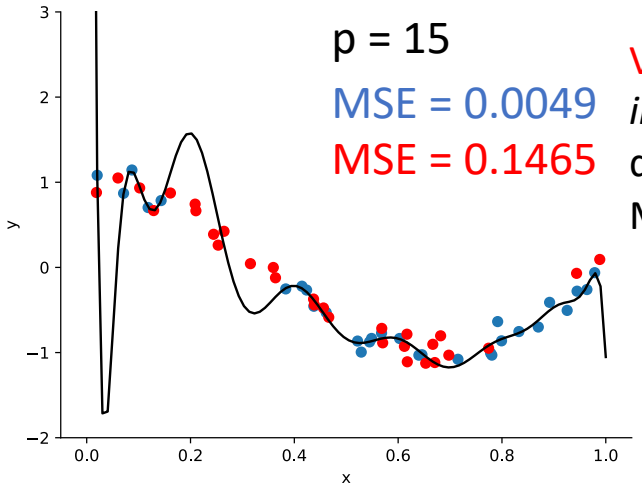


What polynomial degree should we use?



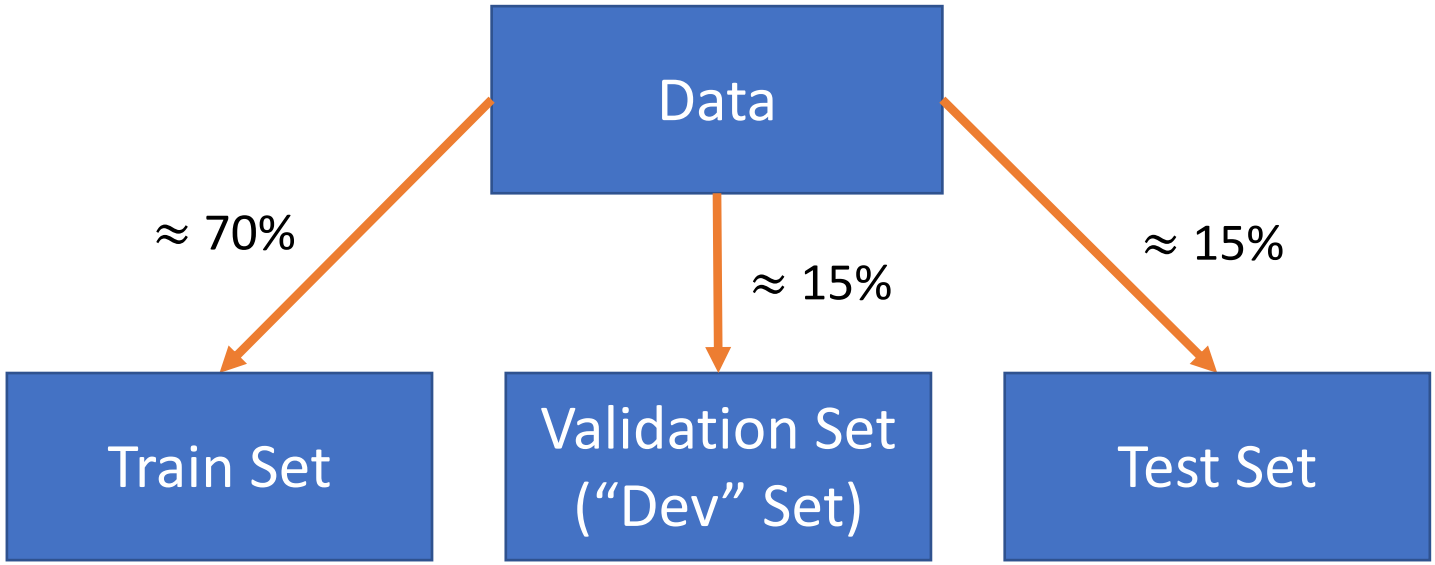






Validation MSE *increases* despite training MSE decreasing

Train/Val/Test Splits

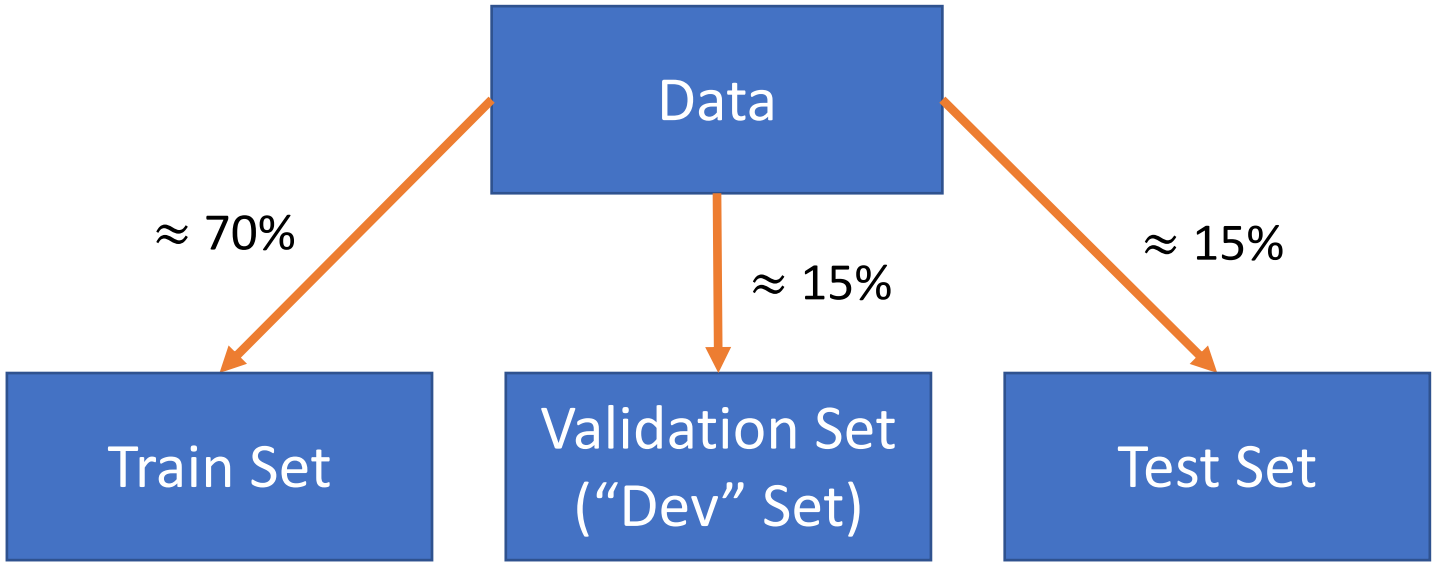


- Train many different models on this subset
- Different hyperparameters, e.g. learning rate, polynomial degree, etc.

- Evaluate each of your models on this subset
- Assess overfitting and select the best model

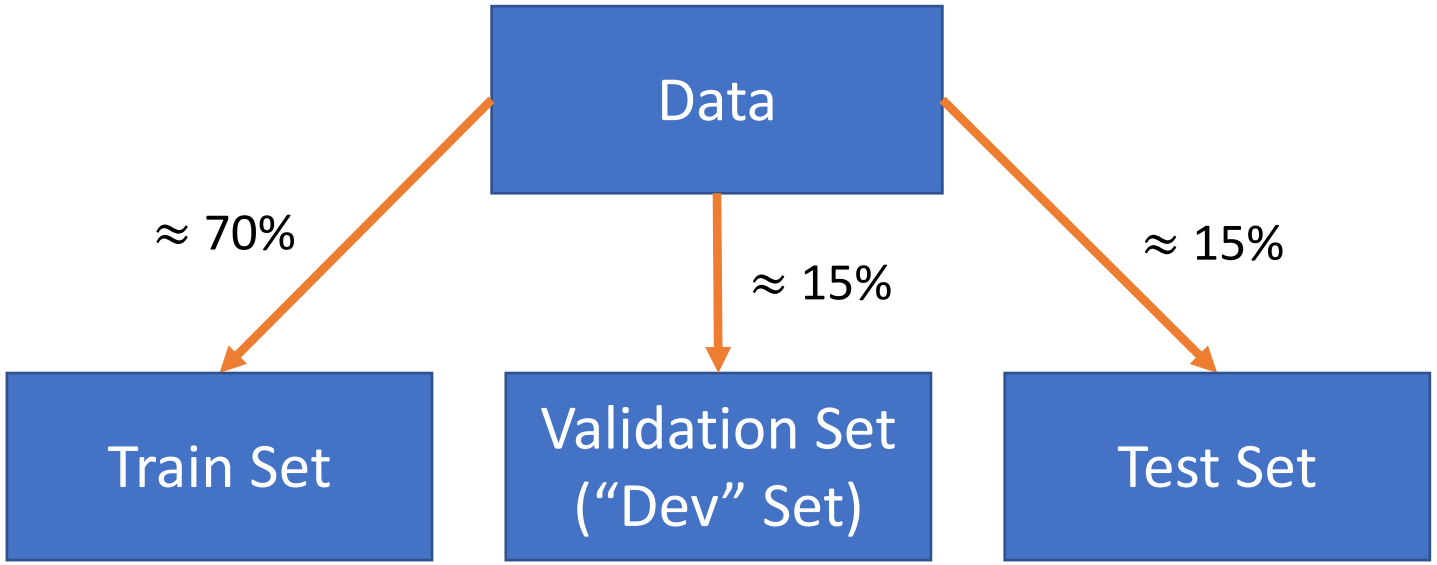
- After selecting your final model, use this set to assess how well your final model will perform on unseen data

Train/Val/Test Splits



- Percentages are just heuristics; can vary
  - Especially for very large datasets, where lower val/test percentages can work
- No overlap between splits!
  - "Data leakage" can lead to overly optimistic model evaluation
- Validation/Test set need to be sufficiently large
  - Too few datapoints → noisy, unreliable estimates of error

Train/Val/Test Splits



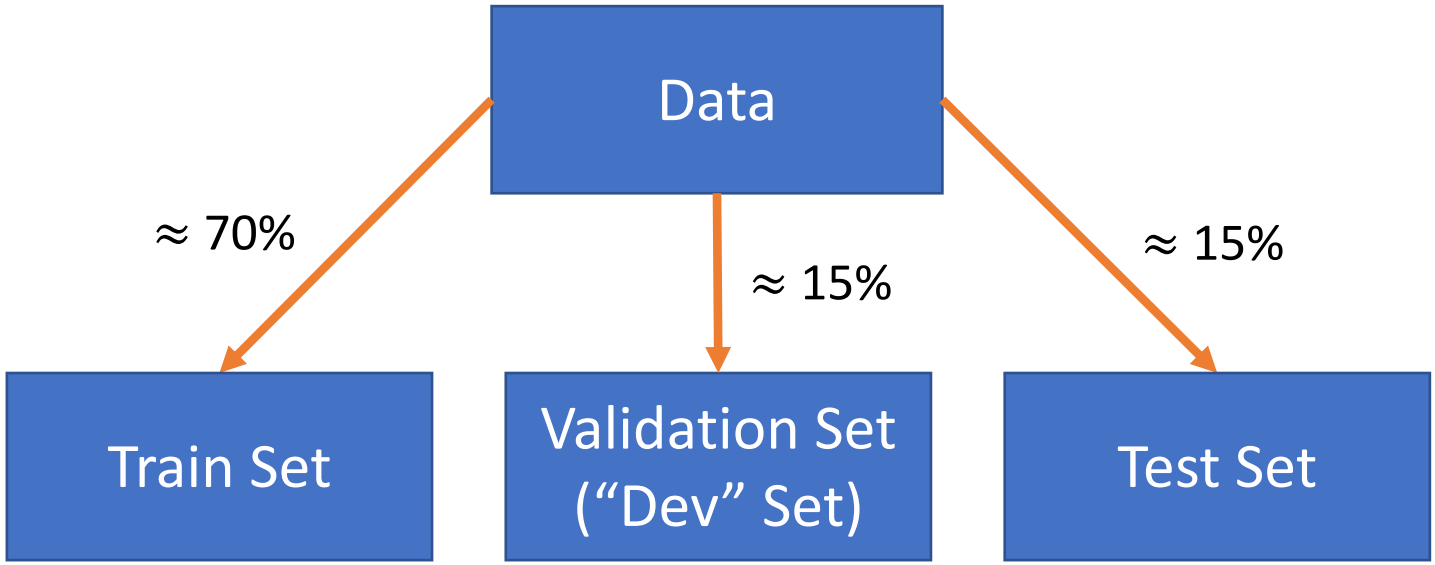
sklearn.model_selection.train_test_split

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None,
shuffle=True, stratify=None)                                                                    [source]
```

Tip: you almost always want to shuffle your data before using it

- E.g. some datasets might come ordered by y-value

Train/Val/Test Splits



- Why do we need a separate test set if we already have a validation set?
  - Performance on the validation set influences our choice of model
  - We are "cheating" by peeking at the validation set; can lead to overfitting
  - True test data is entirely unseen and does not influence the model in any way
    - Meant to simulate data the model will see after deployment

Overfitting and Underfitting

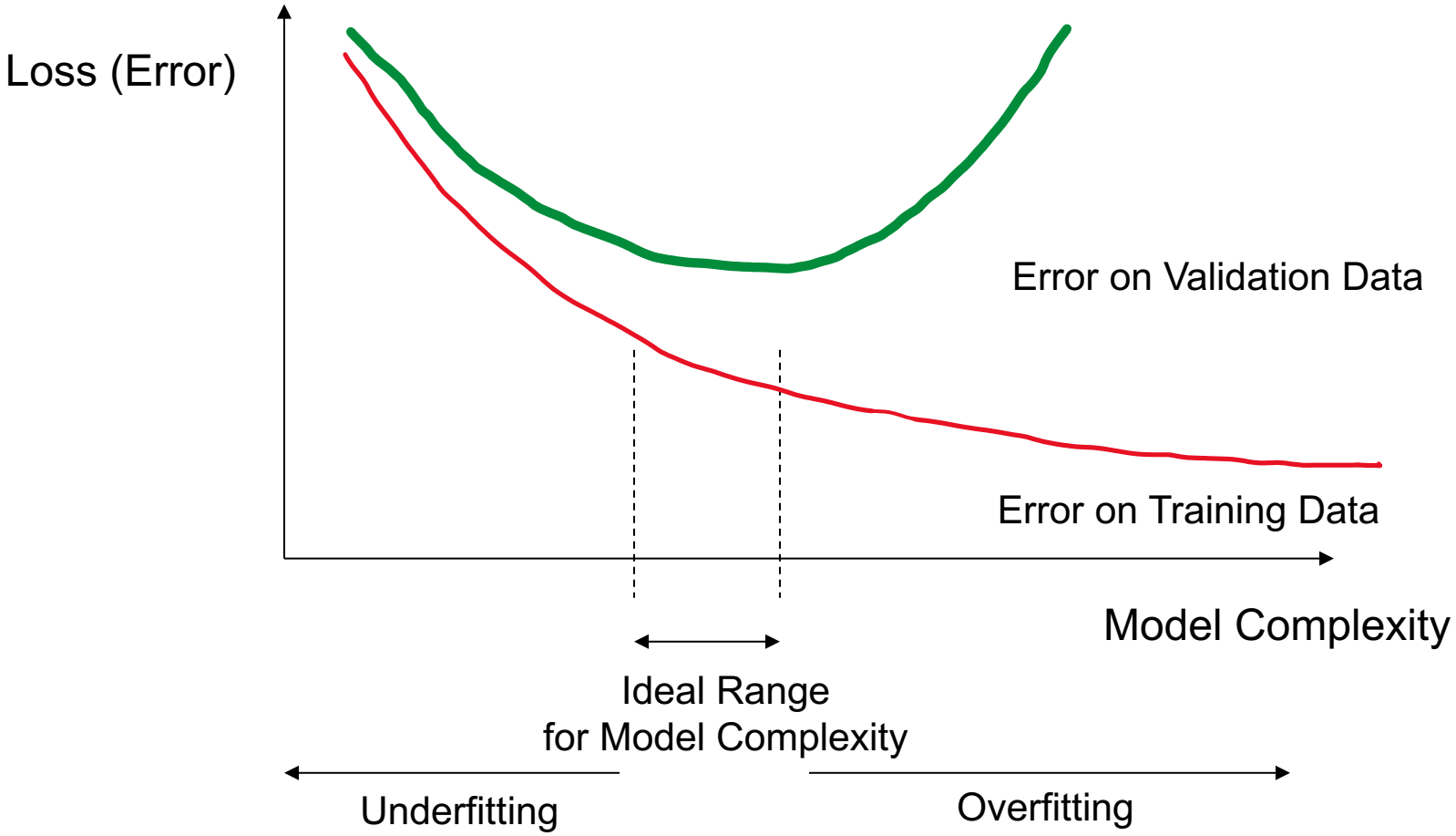
How can we determine if a model is overfitting or underfitting?

- A heuristic (but not set-in-stone) rule:

	High Training Error	Low Training Error
High Val. Error	**Underfitting**	**Overfitting**
Low Val. Error	Bug in your code ☺ (or train/val mismatch)	Appropriate Fit

- Sometimes too expensive to train all of your models at once, and then evaluate
  - E.g. a large neural network (as you will see later in the quarter)
  - Can use this heuristic to help you tune models
    - i.e. iteratively assess under/overfitting and use this to modify model

Overfitting and Underfitting



Evaluation Methods

How do we assess model performance?

- For regression, look at MSE value on different splits
  - Most straightforward way of quantifying performance

- Evaluation is not limited to MSE:
  - Other metrics, like $R^2 = 1 - MSE/Variance(y)$
    - Measures how much variation in y is captured by the model
  - How fast does the model train?
    - We may want to re-train the model if we have new data
  - How fast is prediction with the model?
    - Do we need real-time predictions?
    - e.g. autonomous vehicles
  - How much memory does the model require?
  - Is my model fair?

Questions?

Today's Lecture

Model Selection

Cross Validation

Regularization

K-Fold Cross-Validation

We can use a large validation data set to find the best model
…. but what if we have a small data set, e.g., n=20 or 100?

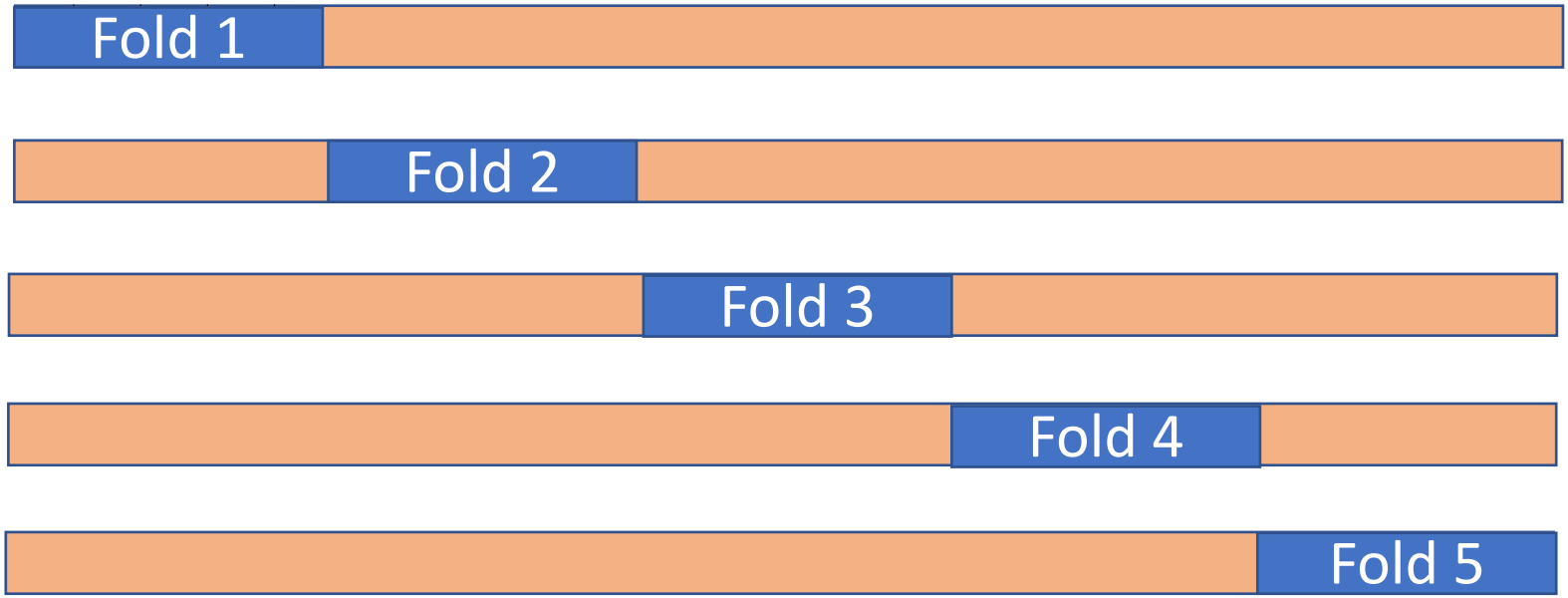
K-fold cross-validation:
-> use multiple validation sets and average over them
-> Example: 10-fold cross-validation
- randomly divide our data into 10 disjoint subsets ("folds"), each with 10% of the n datapoints
- for each of the 10 folds
  - train a model on 90% and compute validation error on the 10%
  - repeat 10 times
- cross-validation error = average of the 10 validation errors

Very general purpose technique
(can be used with classification, regression, with any model)
…. but requires training of K different models

Illustration of 5-fold Cross-Validation

Original data indices

Create 5 folds: for each fold, train on other 80%, evaluate on 20%

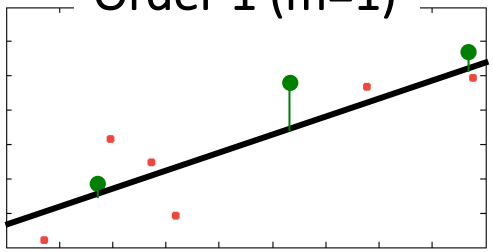


Example of Cross-Validation for Regression

Very small dataset with 9 data points
Evaluate MSE using 3-fold cross-validation to compare m=1, m=3 polynomials

$$f(\mathbf{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_m x^m$$

Order 1 (m=1)



Split 1:
MSE = 331.8

Training data

Validation data

x_i	y_i
88	79
32	-2
27	30
68	73
7	-16
20	43
53	77
17	16
87	94

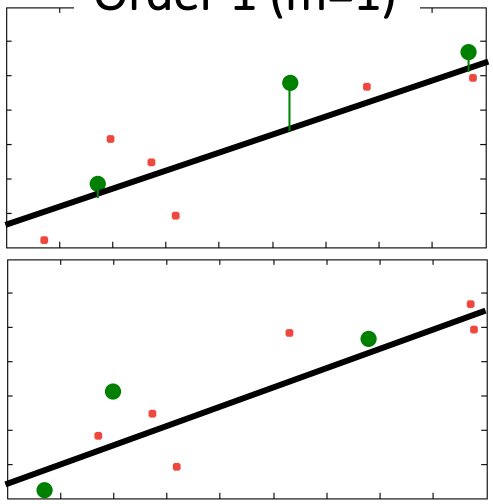
Adapted from slides by Alex Ihler

Example of Cross-Validation for Regression

Very small dataset with 9 data points
Evaluate MSE using 3-fold cross-validation to compare m=1, m=3 polynomials

$$f(\mathbf{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_m x^m$$

Order 1 (m=1)



Split 1:
MSE = 331.8

Split 2:
MSE = 361.2

Training data

Validation data

x_i	y_i
88	79
32	-2
27	30
68	73
7	-16
20	43
53	77
17	16
87	94

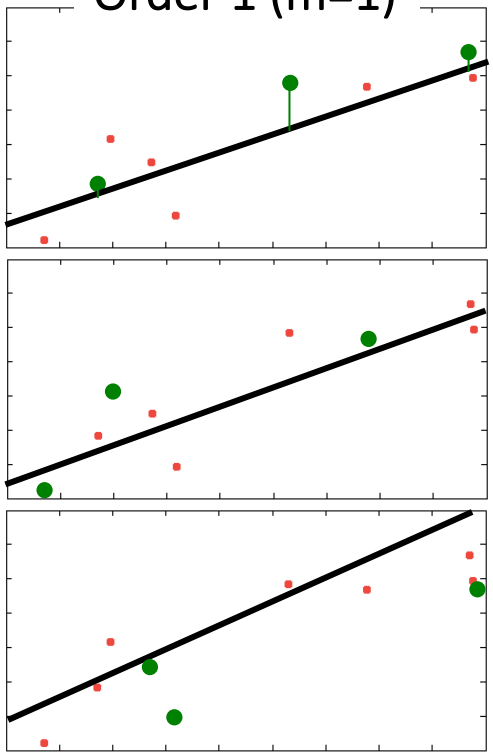
Adapted from slides by Alex Ihler

Example of Cross-Validation for Regression

Very small dataset with 9 data points
Evaluate MSE using 3-fold cross-validation to compare m=1, m=3 polynomials

$$f(\mathbf{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_m x^m$$

Order 1 (m=1)



Split 1:
MSE = 331.8

Split 2:
MSE = 361.2

Split 3:
MSE = 669.8

Training data

Validation data

3-Fold X-Val MSE = 464.1

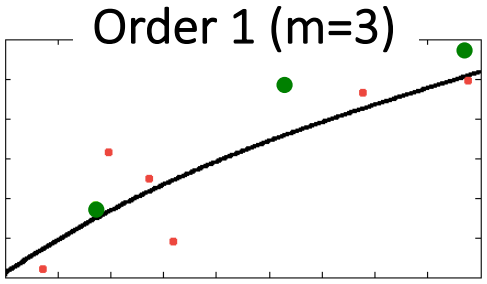
x_i	y_i
88	79
32	-2
27	30
68	73
7	-16
20	43
53	77
17	16
87	94

Adapted from slides by Alex Ihler

Example of Cross-Validation for Regression

Very small dataset with 9 data points
Evaluate MSE using 3-fold cross-validation to compare m=1, m=3 polynomials

$$f(\mathbf{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_m x^m$$



Split 1:
MSE = 280.5

Training data

Validation data

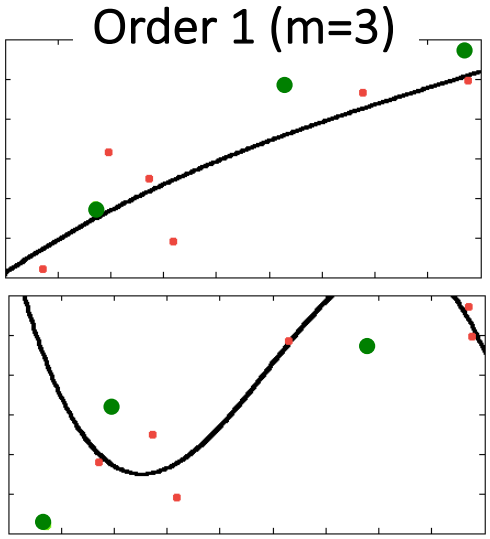
x_i	y_i
88	79
32	-2
27	30
68	73
7	-16
20	43
53	77
17	16
87	94

Adapted from slides by Alex Ihler

Example of Cross-Validation for Regression

Very small dataset with 9 data points
Evaluate MSE using 3-fold cross-validation to compare m=1, m=3 polynomials

$$f(\mathbf{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_m x^m$$



Split 1:
MSE = 280.5

Split 2:
MSE = 3081.3

Training data

Validation data

x_i	y_i
88	79
32	-2
27	30
68	73
7	-16
20	43
53	77
17	16
87	94

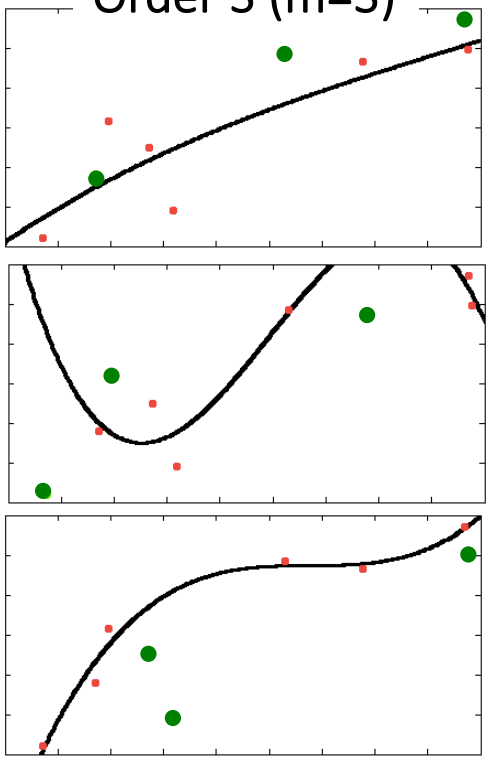
Adapted from slides by Alex Ihler

Example of Cross-Validation for Regression

Very small dataset with 9 data points
Evaluate MSE using 3-fold cross-validation to compare m=1, m=3 polynomials

$$f(\mathbf{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_m x^m$$

Order 3 (m=3)



Split 1:
MSE = 280.5

Split 2:
MSE = 3081.3

Split 3:
MSE = 1640.1

3-Fold X-Val MSE
= 1667.3

Training data

Validation data

x_i	y_i
88	79
32	-2
27	30
68	73
7	-16
20	43
53	77
17	16
87	94

Adapted from slides by Alex Ihler

Cross Validation in Sklearn

sklearn.model_selection.KFold

class `sklearn.model_selection.`**`KFold`**(*n_splits=5, *, shuffle=False, random_state=None*) [source]

sklearn.model_selection.LeaveOneOut

class `sklearn.model_selection.`**`LeaveOneOut`** [source]

- Cross validation with the number of folds equal to the number of datapoints
- i.e. with n datapoints, on each split we fit a model on n-1 datapoints and evaluate on 1 datapoint
- Useful for very small datasets
  - Lets us train model on most possible data
  - Can be too expensive for large datasets

Bootstrapping

An alternate method to cross-validation is **bootstrapping**

Have a dataset with n datapoints
Create M *new* train/val sets as follows:

1. Create a training set by sampling n datapoints from your data with replacement

2. Create a validation set by taking all datapoints not in the training set

Fit models on train sets; evaluate on validation sets and average performance

Bootstrapping

An alternate method to cross-validation is **bootstrapping**

Bootstrapping vs Cross-Validation:

- Both widely used in practice
- Bootstrapping more popular in statistics; Cross-validation more popular in ML
- Cross-validation trains models on datasets smaller than the original data
  - Can underestimate model performance
- Cross-validation can be thought of as similar to bootstrapping *without* replacement

We will use bootstrapping later in the course to build *ensemble* methods

Questions?

Today's Lecture

Model Selection

Cross Validation

Regularization

Regularization with MSE Loss Function

Linear regression model:

$$f(x \mid \theta) = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d = \theta^T \boldsymbol{x}$$

Model is learned by minimizing the MSE:

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

Recall: overfitting happens when our model is "too complex"

- Can we somehow measure the complexity of a model using the parameters θ?

Regularization for Regression

Suppose we can somehow measure the complexity of a model

- Using a function $R(\theta)$
- Higher values of $R(\theta)$ → higher complexity

We can **regularize** our model by adding a complexity penalty to the loss:

$$\begin{aligned} L_R(\theta) = &\quad \text{MSE}(\theta) + \alpha R(\theta) \\ = &\quad \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i \mid \theta))^2 + \alpha R(\theta) \end{aligned}$$

Minimizing $L_R(\theta)$ requires our model to balance:

- Error on the training data, i.e. MSE
- Complexity of the model, i.e. $R(\theta)$
- As long as $R(\theta)$ is differentiable, can still use gradient descent

Regularization for Regression

Suppose we can somehow measure the complexity of a model

- Using a function $R(\theta)$
- Higher values of $R(\theta)$ → higher complexity

We can **regularize** our model by adding a complexity penalty to the loss:

$$L_R(\theta) = \text{MSE}(\theta) + \alpha R(\theta)$$

$$= \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i \mid \theta))^2 + \alpha R(\theta)$$

$\alpha > 0$ is the **regularization strength**

- Yet another hyperparameter we can tune
- Higher α means more regularization

Norms

A **norm** is a function that takes in a vector and measures its "size" or "magnitude"

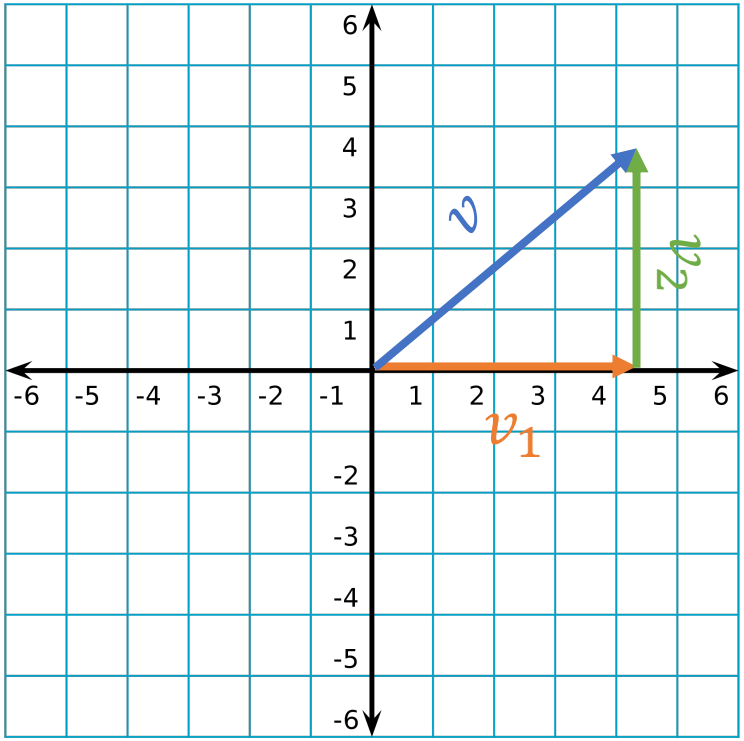
Norm of a vector v usually denoted $\|v\|$

- i.e. $\|v\|$ is a non-negative real number telling us the magnitude of the vector v
- Has a formal definition (we won't need it in this class)

Example: the **Euclidean** norm (or L2-norm):

$$v = (v_1, v_2, \ldots, v_d)$$

$$\|v\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_d^2}$$



- Corresponds to the usual "length"
  - Pythagorean Theorem

Norms

A **norm** is a function that takes in a vector and measures its "size" or "magnitude"

Norm of a vector v usually denoted $\|v\|$

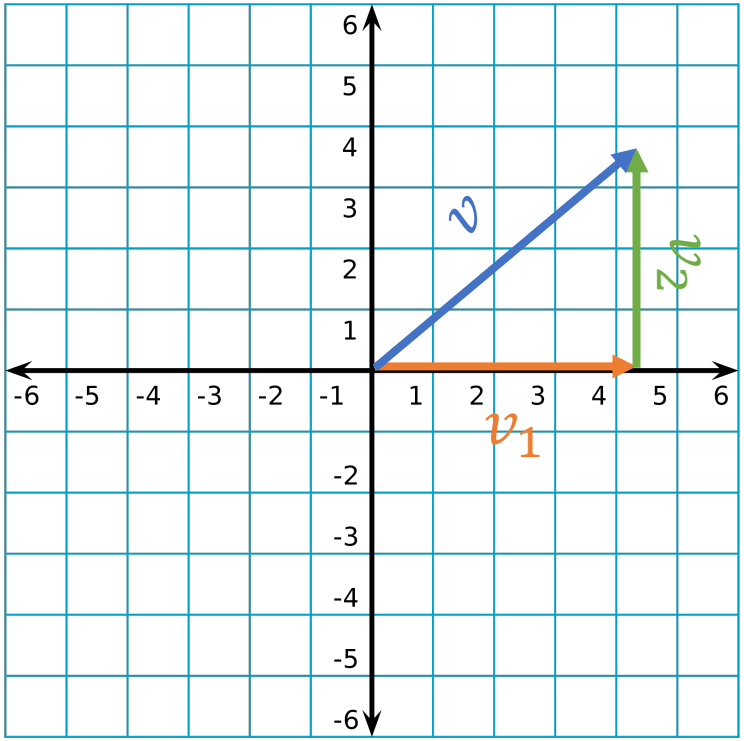
- i.e. $\|v\|$ is a non-negative real number telling us the magnitude of the vector v
- Has a formal definition (we won't need it in this class)

Example: the **L1-norm**:

$$v = (v_1, v_2, \dots, v_d)$$

$$\|v\|_1 = |v_1| + |v_2| + \cdots + |v_d|$$

- Measures length by adding lengths of individual components



Norms

A **norm** is a function that takes in a vector and measures its "size" or "magnitude"

Norm of a vector v usually denoted $\|v\|$

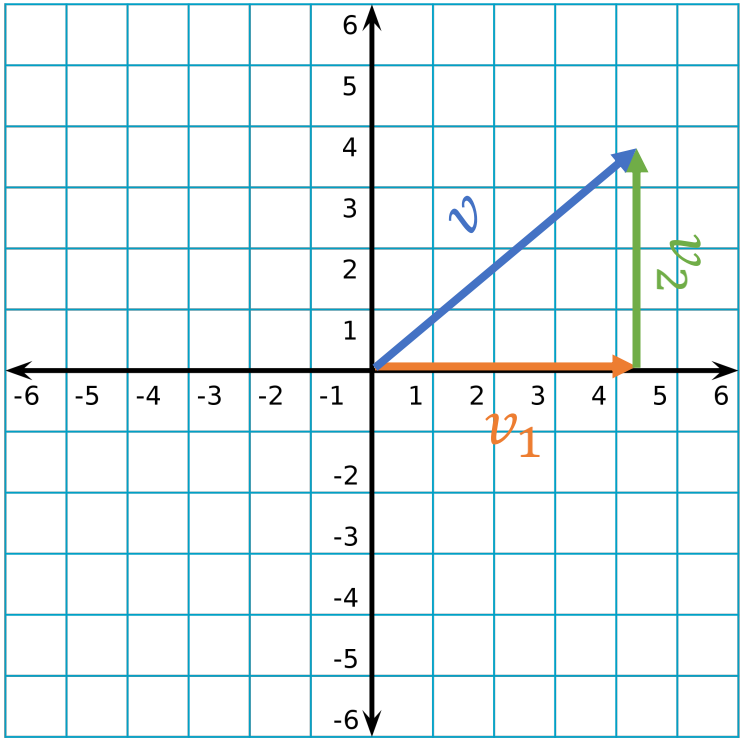
- i.e. $\|v\|$ is a non-negative real number telling us the magnitude of the vector v
- Has a formal definition (we won't need it in this class)

General formula for **Lp-norms**, p>0:

$$v = (v_1, v_2, \ldots, v_d)$$

$$\|v\|_p = \left(\sum_{i=1}^{d} |v_i|^p\right)^{1/p}$$

- Different values of p change how we measure length
  - (more details later)



Norms as Measuring Model Complexity

We can use the norm of the parameter vector θ to measure how complex the model having those parameters is

$$\|\theta\|_2 = \sqrt{\theta_0^2 + \theta_1^2 + \cdots + \theta_d^2}$$

Example: quadratic regression

$$f(\boldsymbol{x}\,|\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$

If $\theta_0, \theta_1, \theta_2$ are all small:

- The parabola is approximately the zero function: $f(\boldsymbol{x}\,|\theta) \approx 0$
  - Clearly not a "complex" model
- $\|\theta\|_2$ is also small

Regularized Regression

We can use the norm of the parameter vector θ as the "complexity function"

$$R(\theta) = \|\theta\|_p$$

$$\begin{aligned} L_R(\theta) = &\quad \text{MSE}(\theta) + \alpha R(\theta) \\ = &\quad \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i \mid \theta))^2 + \alpha R(\theta) \\ = &\quad \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i \mid \theta))^2 + \alpha \|\theta\|_p \end{aligned}$$

Most common special cases:

- $p = 1$ ("LASSO")
- $p = 2$ ("Ridge")

Regularized Regression in Sklearn

sklearn.linear_model.Lasso

class `sklearn.linear_model.`**`Lasso`**(*alpha=1.0, *, fit_intercept=True, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic'*) [source]

sklearn.linear_model.Ridge

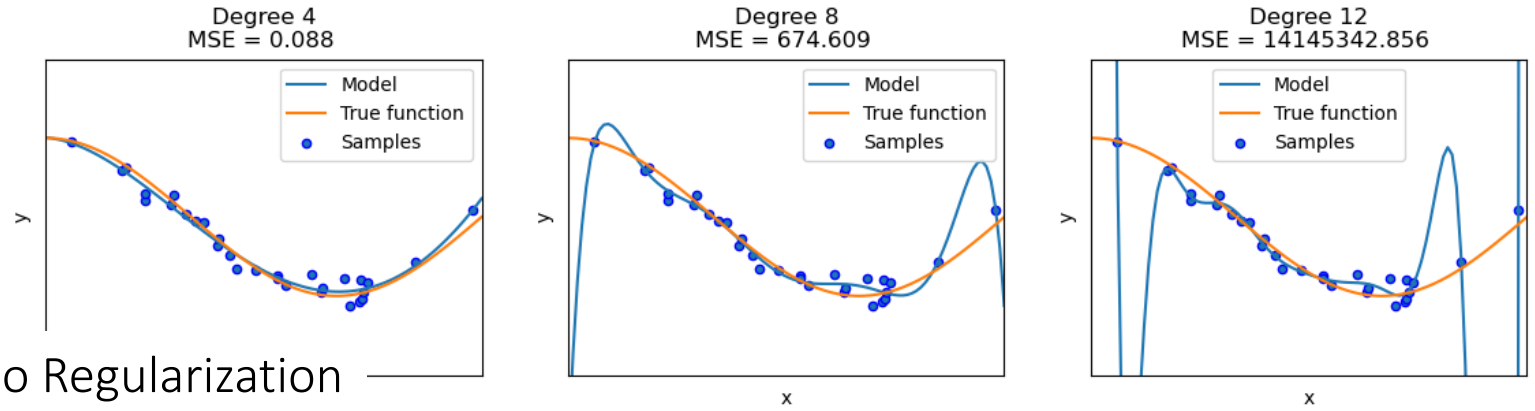
class `sklearn.linear_model.`**`Ridge`**(*alpha=1.0, *, fit_intercept=True, copy_X=True, max_iter=None, tol=0.0001, solver='auto', positive=False, random_state=None*) [source]

Regularization strength

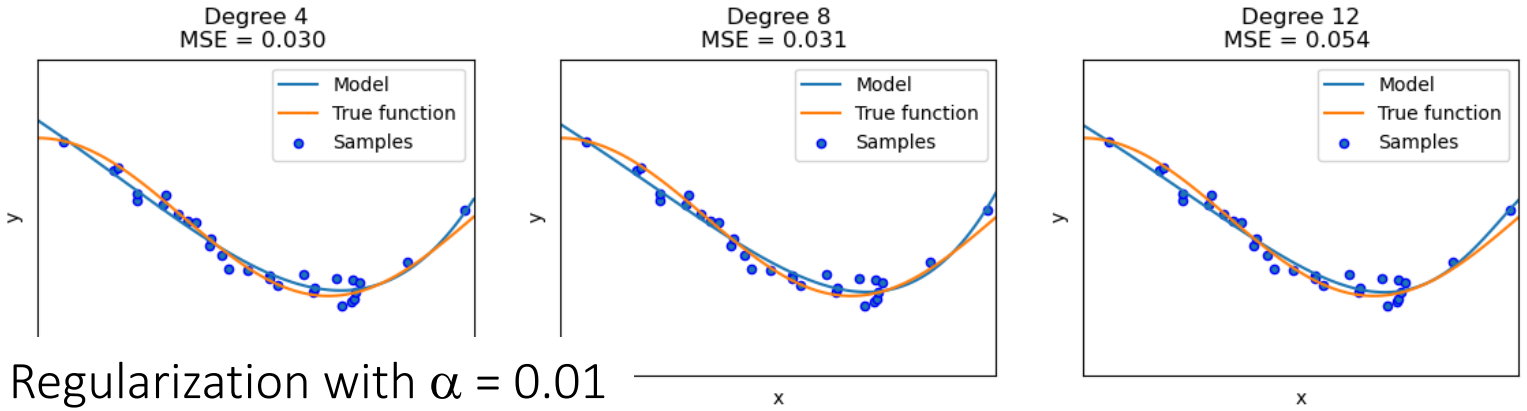
- Needs to be tuned
- Can have significant impact on model performance

Polynomial Regression with L2 Regularization

True curve = Sine Wave + noise: n = 30 datapoints, MSE estimated with 10-fold cross-validation



No Regularization



L2 Regularization with $\alpha = 0.01$

Adapted from https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html#sphx-glr-auto-examples-model-selection-plot-underfitting-overfitting-py

Comparing Coefficient Estimates with and w/o Regularization

8^{th} order polynomial fit to Sine Wave + noise, n = 30 datapoints

Coefficients	Fitted Values with No Regularization	Fitted Values with L2 Regularization, $\alpha = 0.01$	Fitted Values with L1 Regularization, $\alpha = 0.01$
θ_1	133.9	-3.54	-3.04
..	-1628.0	0.98	-0.00
..	9589.3	0.76	0.00
..	-31725.5	1.30	0.00
..	61479.8	1.14	0.00
..	-69079.0	0.68	2.14
..	41604.9	0.14	0.00
θ_8	-10373.6	-0.38	0.00

Why should L1 regularization result in so many zeros?

Geometric Intuition for L1 Sparsity

Why should L1 regularization result in so many zeros for the weights?

- Sometimes called "sparsity inducing"

Instead of the usual regularized MSE:

$$L_R(\theta) = \text{MSE}(\theta) + \alpha \|\theta\|_1$$

$$= \text{MSE}(\theta) + \alpha \sum_{j=0}^{d} |\theta_j|$$

Consider the "constrained" MSE optimization problem:

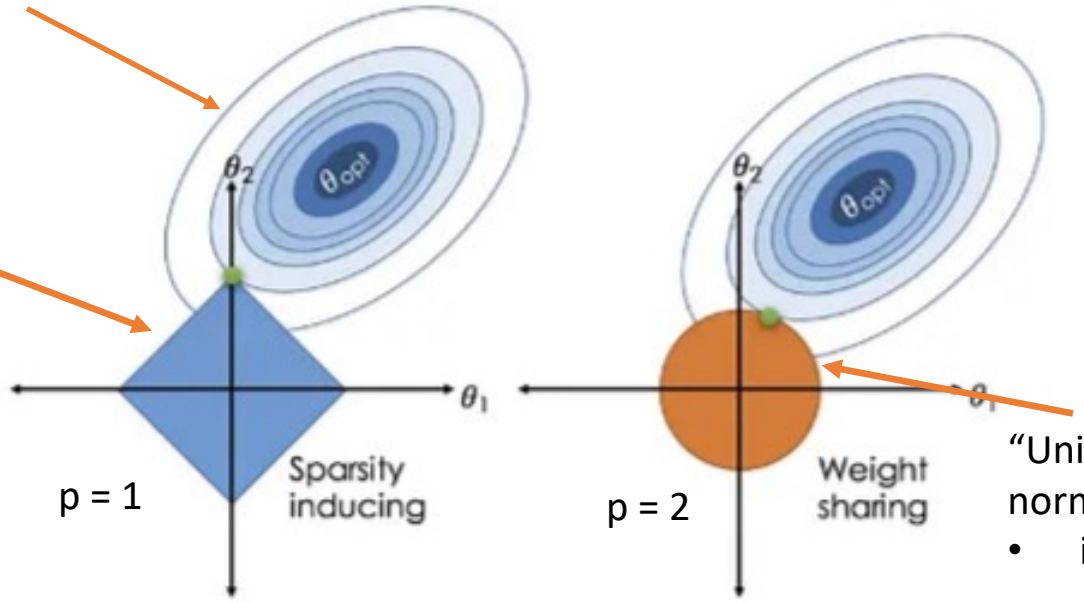
$$\min_{\theta} \text{MSE}(\theta) \quad \text{such that} \quad \|\theta\|_1 \leq 1$$

LASSO vs Ridge Regression

Loss contours

"Unit Ball" for the L1 norm

- i.e. all vectors with $\|\theta\|_1 \leq 1$



"Unit Ball" for the L2 norm

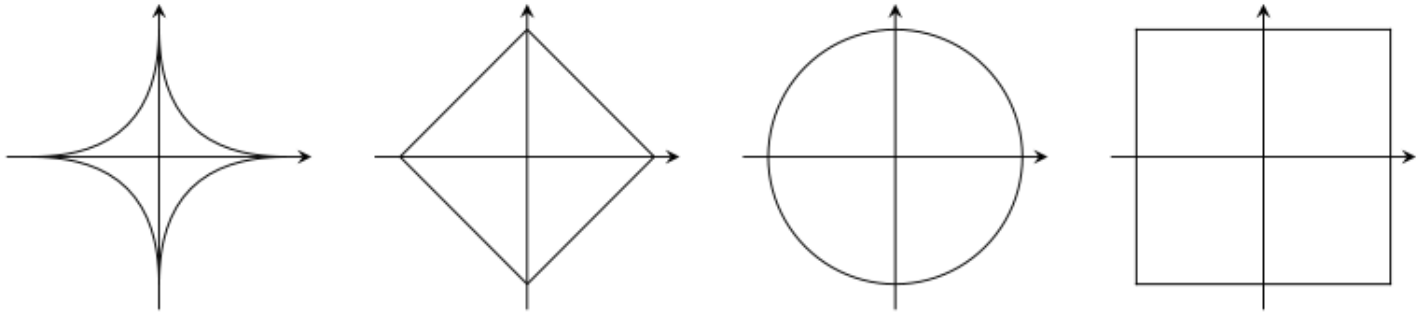
- i.e. all vectors with $\|\theta\|_2 \leq 1$

Solution to constrained MSE problem must lie in corresponding unit ball

- Due to geometry of L1 unit ball, much more likely to hit "corners"
- i.e. points where at least some of the parameters are zero!

Zou, H., & Hastie, T. (2005)

Unit Balls for Other Lp Norms



- p < 1 results in even more sparsity
  - But leads to a nonconvex optimization problem
  - $1 < p < \infty$ always leads to a convex loss $L_R(\theta)$
- Special case p = ∞
  - $\|\theta\|_\infty = \max_{i=0,1,\ldots,d} |\theta_i|$
  - Not differentiable

Elastic Net Regression

We can be creative in how we define the complexity function

Example: “Elastic-Net” regularization uses a sum of L1 and L2 norms:

$$R(\theta) = \alpha_1 \|\theta\|_1 + \alpha_2 \|\theta\|_2$$

Why is this useful?

- L1 regularization can lead to non-unique solutions
- Can be problematic if you want to interpret the weights of your model
- Elastic-Net is a convex optimization problem; unique minimizer
- Achieves both sparsity and uniqueness

Summary and Wrapup

- *Model Selection* is the process of deciding which model to use
  - Tuning hyperparameters
  - Choosing which model to use (later in the course)

- Best practices:
  - Train/val/test splits
  - Cross validation or bootstrapping if you have limited data
  - Use train/val performance to assess over/under fitting

- Regularization
  - Add a term to your loss which penalizes model complexity
  - E.g. norm of parameter vector
  - L1 (Lasso) and L2 (Ridge) regression

Next Lecture

- Regression portion of the course complete

- We'll focus on classification for the next several weeks
  - Nearest centroids, kNN, logistic classifiers, neural networks

Week 2				
Monday 4/10	Lec04	Gradient Descent; Nonlinear Regression		Partial Derivatives [1] [2]
Wednesday 4/12	Lec05	Model Selection and Regularization		
Thursday 4/13	Dis02	Intro to scikit-learn and matplotlib		
Friday 4/14	Lec06	Classification; Nearest Centroids	**HW1 Due**	
Week 3				
Monday 4/17	Lec07	Probability review; Classification metrics; Bayes Error		
Wednesday 4/19	Lec08	k-Nearest Neighbors		
Thursday 4/20	Dis03			
Friday 4/21	Lec09	k-Nearest Neighbors		
Week 4				
Monday 4/24	Lec10	k-Nearest Neighbors; Logistic Classifiers		
Wednesday 4/26	Lec11	Logistic Classifiers		
Thursday 4/27	Dis04			
Friday 4/28	Lec12	Logistic Classifiers	**HW2 Due**	
Week 5				
Monday 5/1	Lec13	Logistic Classifiers		
Wednesday 5/3	Lec14	Logistic Classifiers		
Thursday 5/4	Dis05			
Friday 5/5	Lec15	**Midterm exam (in class)**		

Questions?