

Course Announcements

Due today (Friday; 11:59 PM):

- D7
- Q7
- Project Checkpoint #2: EDA
- Weekly Project Survey

Grades to be released:

- Data Checkpoint, D6, Q6 (Mon afternoon)
- A3 (later this week)

Project Checkpoint #2: EDA

Details: https://github.com/COGS108/Projects/blob/master/FinalProject_Guidelines.md#checkpoint-2-eda

Sections:

- Question
- Setup
- Data
- Data Cleaning
- Data Analysis & Results: EDA

At the end of this checkpoint, it should be clear that you know your data well. Note that visualizations do not have to be perfect (yet!), but they do have to be appropriate and interpreted. Include explanations of what you learn from each visualization generated.

Machine Learning Example

Shannon E. Ellis, Ph.D
UC San Diego

• • •

Department of Cognitive Science
sellis@ucsd.edu

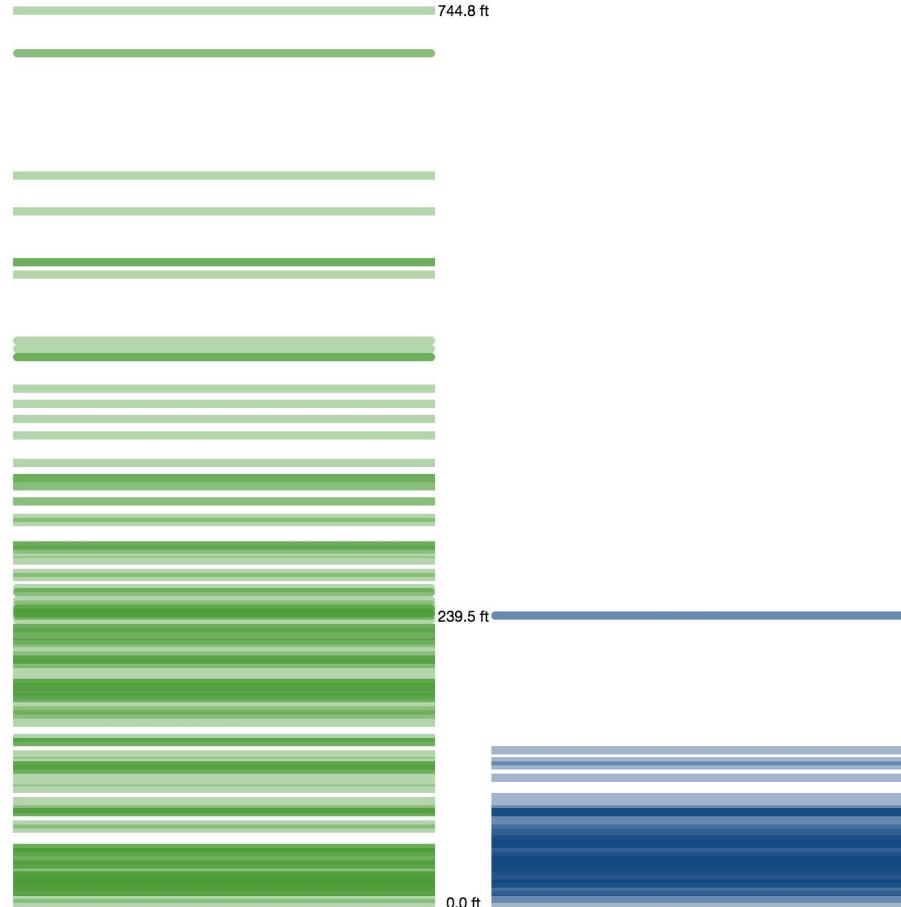
What features distinguish a
house in New York from a
house in San Francisco?

First, some intuition

Let's say you had to determine whether a home is in **San Francisco** or in **New York**. In machine learning terms, categorizing data points is a **classification** task.

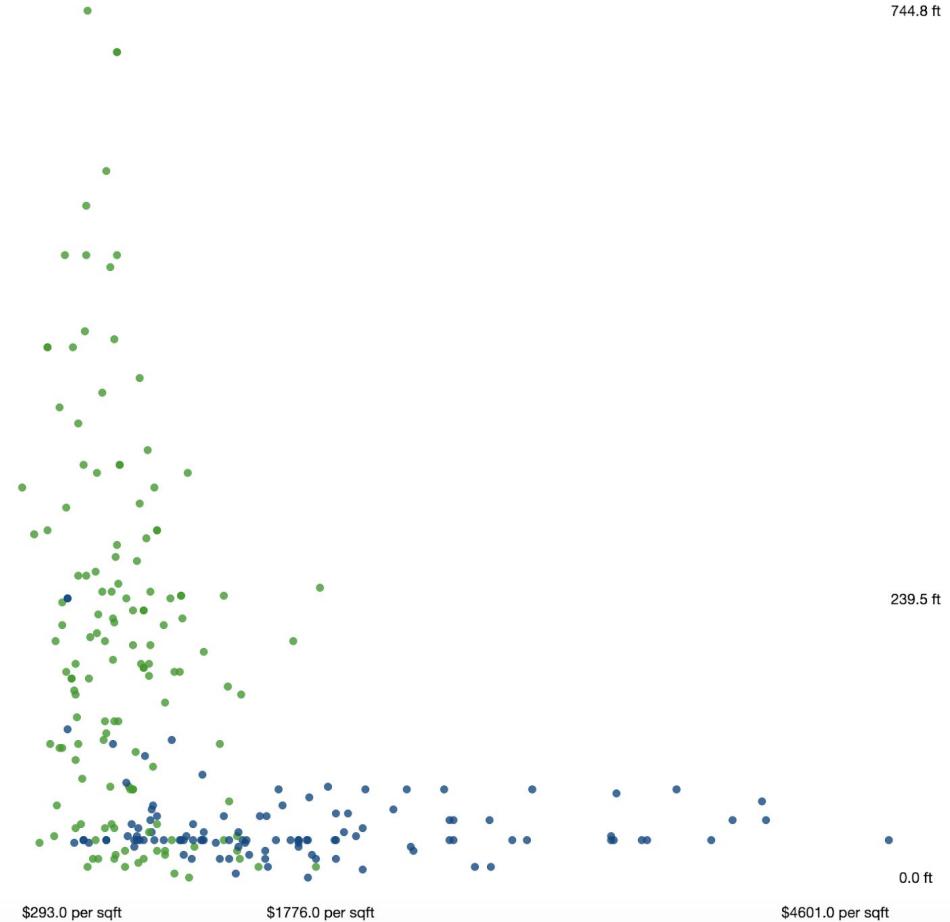
San Fran is hilly ...so elevation may be a helpful feature.

With the data here, homes > ~73m should be classified as San Fran homes



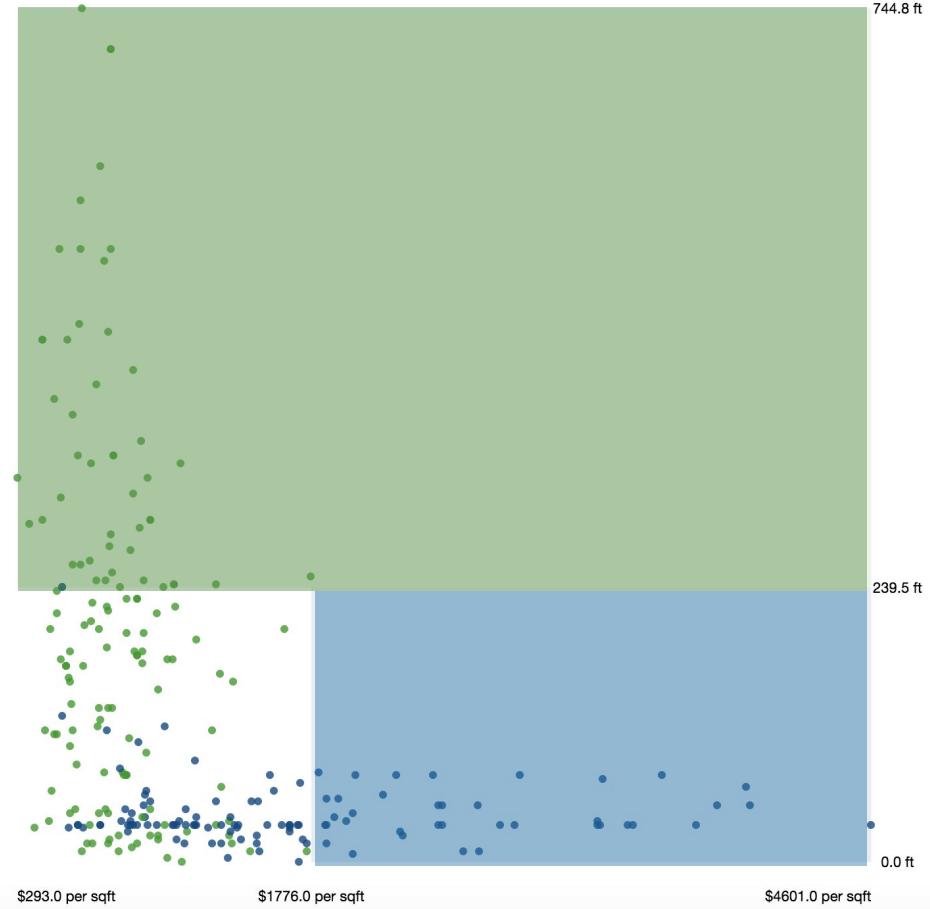
Adding nuance

Elevation isn't a perfect feature for classification, so we can look at its relationship to other features, like *price per square foot*



Drawing boundaries

Boundaries can be drawn so that if a house falls in the green box, it's classified as a San Fran home. Blue box, New York. Statistical learning figures out how to best draw these boxes.



Our training set will use 7 different **features**. At the right we see the **scatterplot matrix** of the relationship between these features.

Patterns are clear, but boundaries for delineation are not obvious.



New York
San Fran

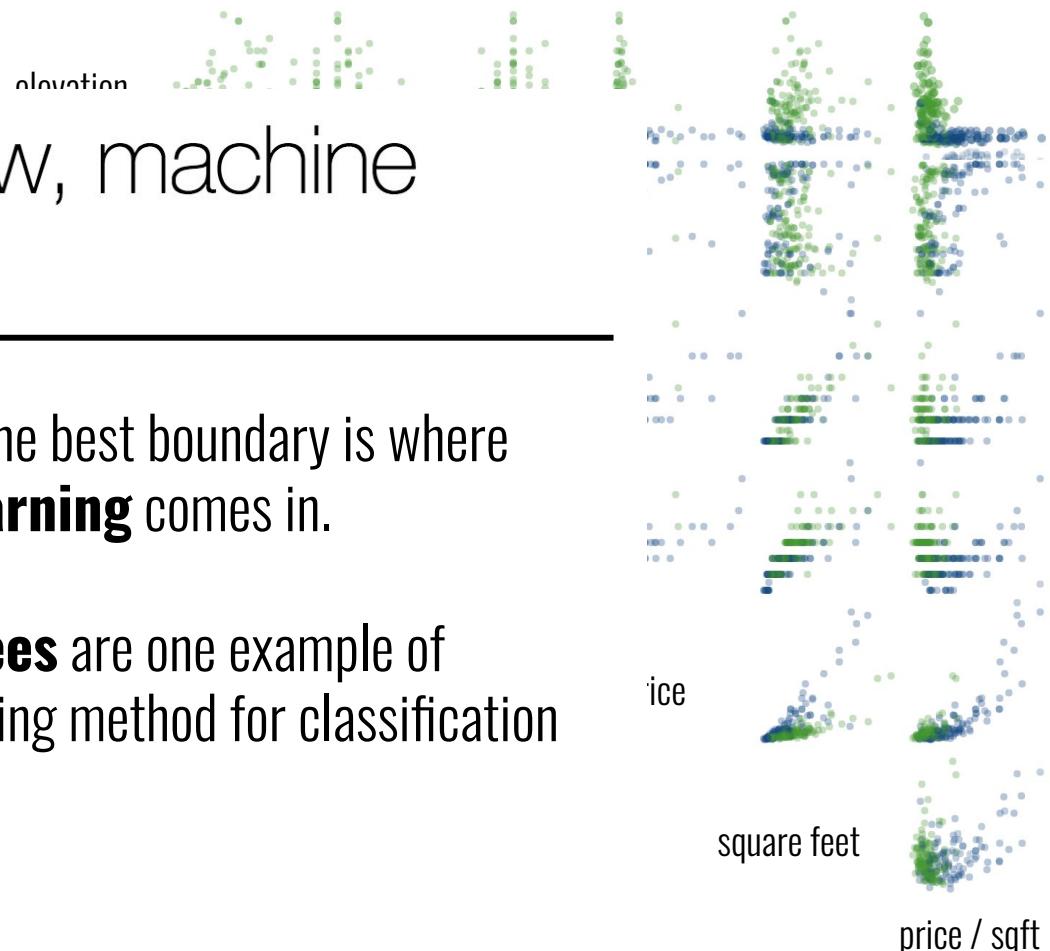
Our training set will use **features**. At the right is a **scatterplot matrix** showing the relationship between

Patterns are clear, but delineation are not obvious.

And now, machine learning

Determining the best boundary is where **machine learning** comes in.

Decision trees are one example of machine learning method for classification tasks.





Finding better boundaries

We guessed ~73m before. Let's improve on that guess...



A **histogram** helps display frequency of homes by elevation more easily.

73m is the highest home in New York, but most of them have lower elevations

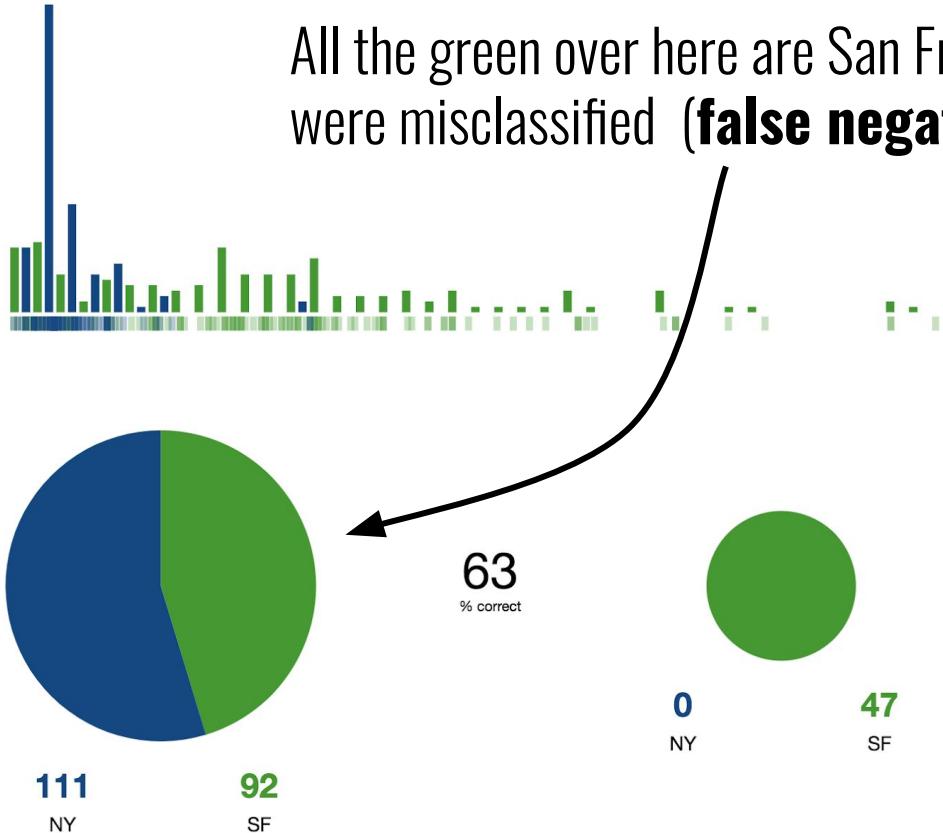
In machine learning, the splits are called **forks** and they split the data into **branches** based on some value.

The value that splits the branches is the **split point**. Homes to the left get categorized differently than those on the right.



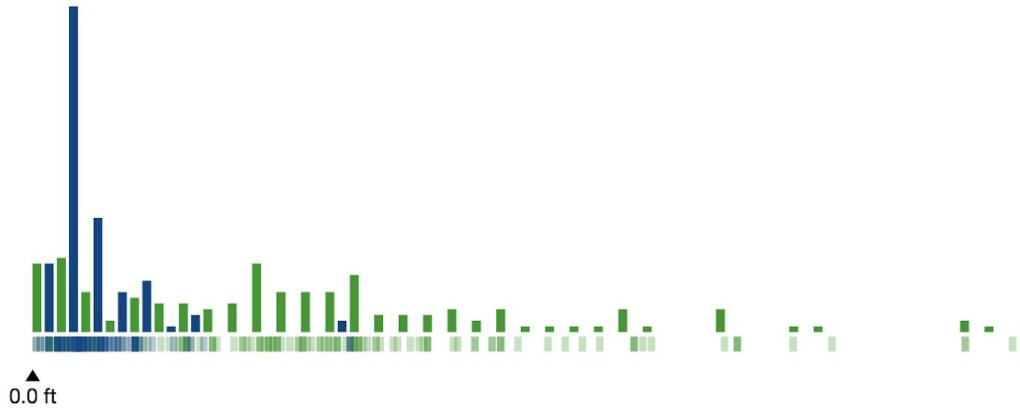
Your first fork

A decision tree uses if-then statements to define patterns in the data.



Tradeoffs

Splitting at ~73m incorrectly classifies some San Francisco homes as New York homes.

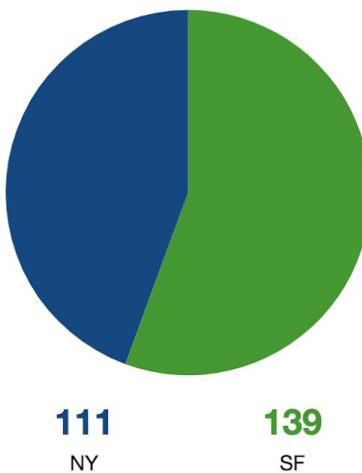


0.0 ft

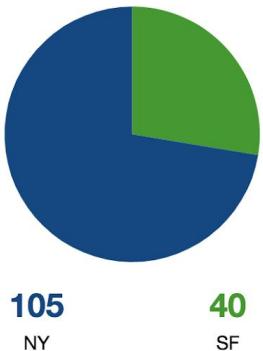
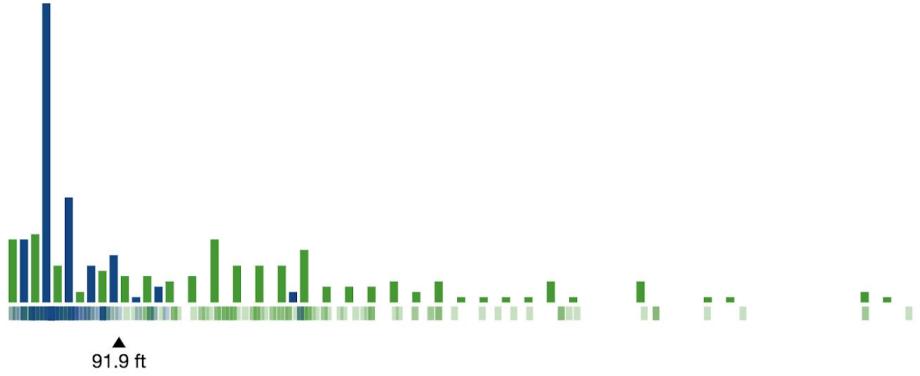
0
NY

0
SF

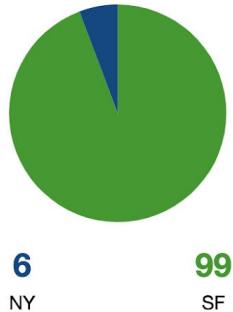
56
% correct



If you split to capture *every* home in San Fran, you'll also get a bunch of New York homes (**false positives**)

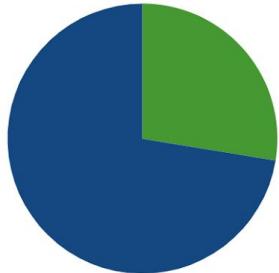


82
% correct

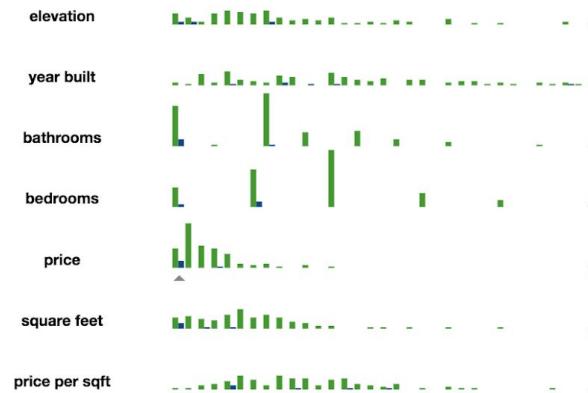
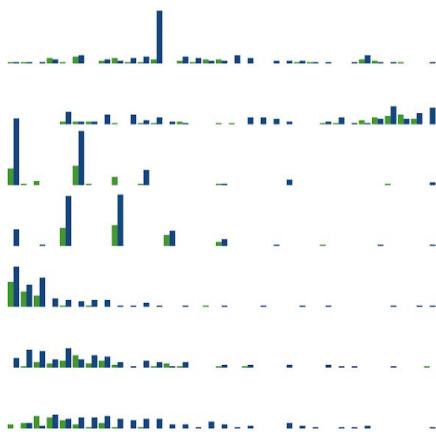


The best split

The best split point aims for branches that are as homogenous (pure) as possible

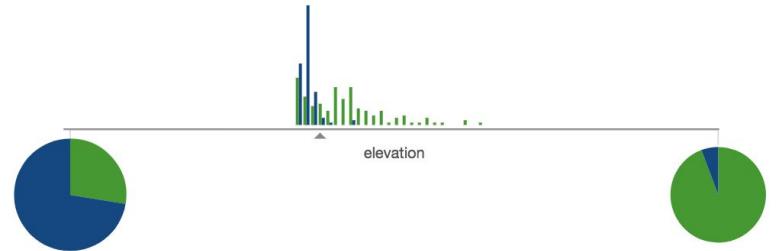


82
% correct



Recursion

Additional split points are determined through repetition (**recursion**)



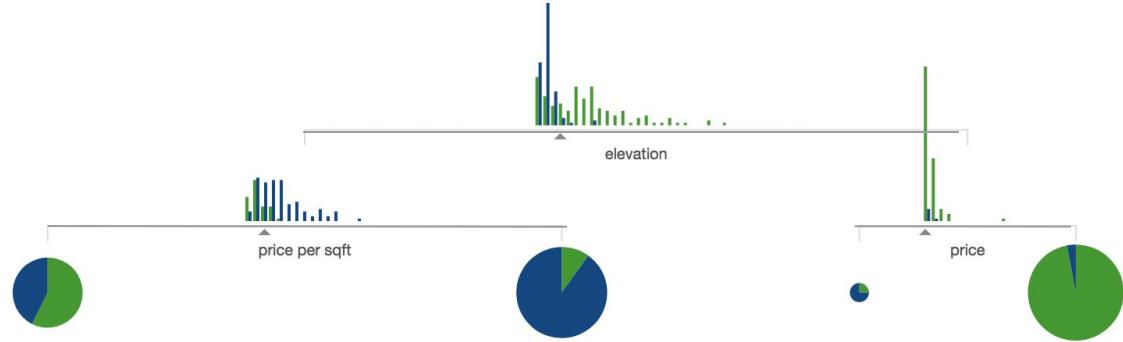
Growing a tree

Additional forks add new
information to improve
prediction accuracy.

Accuracy: 82%

Growing a tree

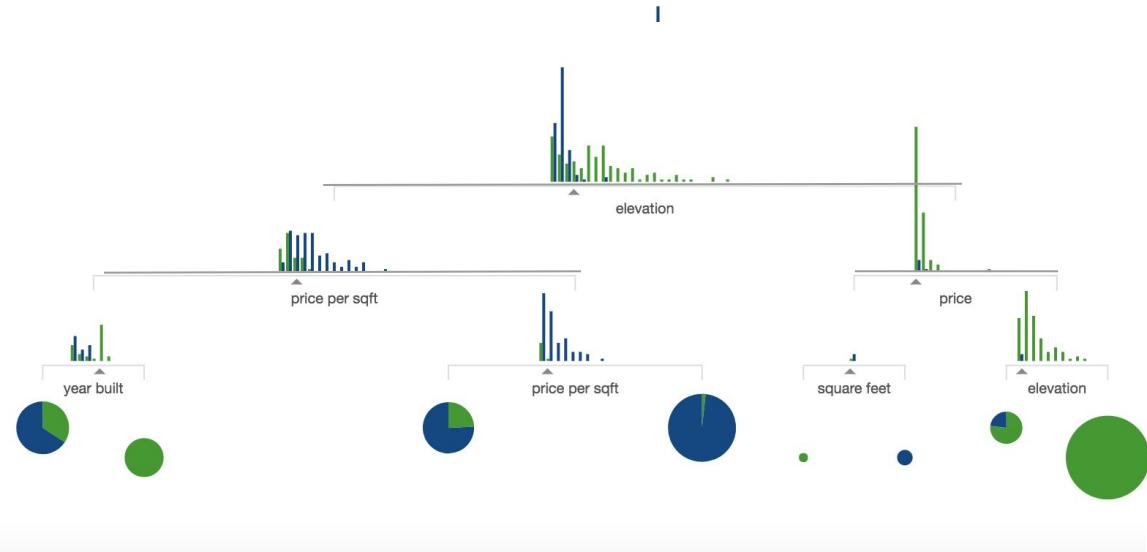
Additional forks add new
information to improve
prediction accuracy.



Accuracy: 86%

Growing a tree

Additional forks add new information to improve
prediction accuracy.



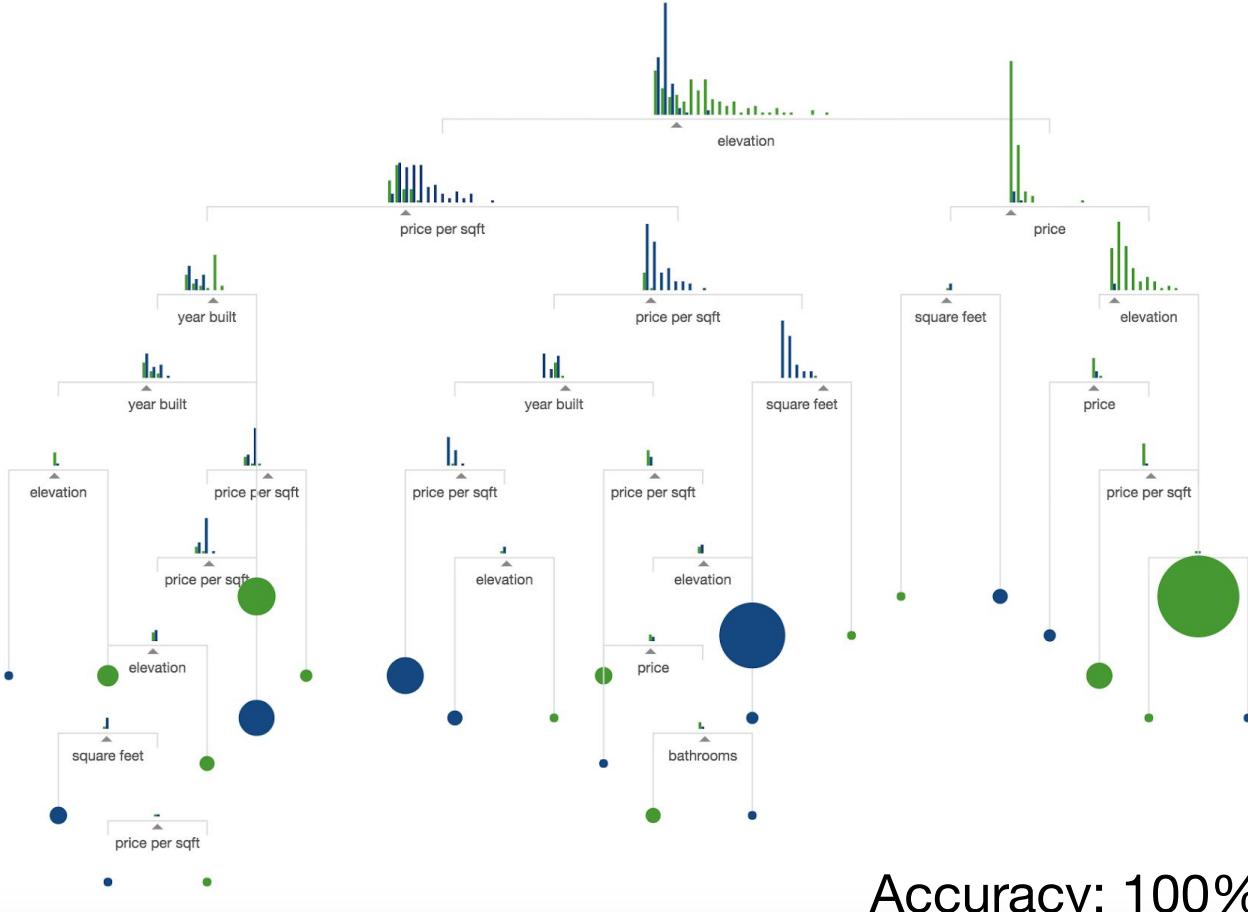
Adding several more layers, we get to

96%.



Accuracy: 96%

It's possible to add branches until your model is **100% accurate.**

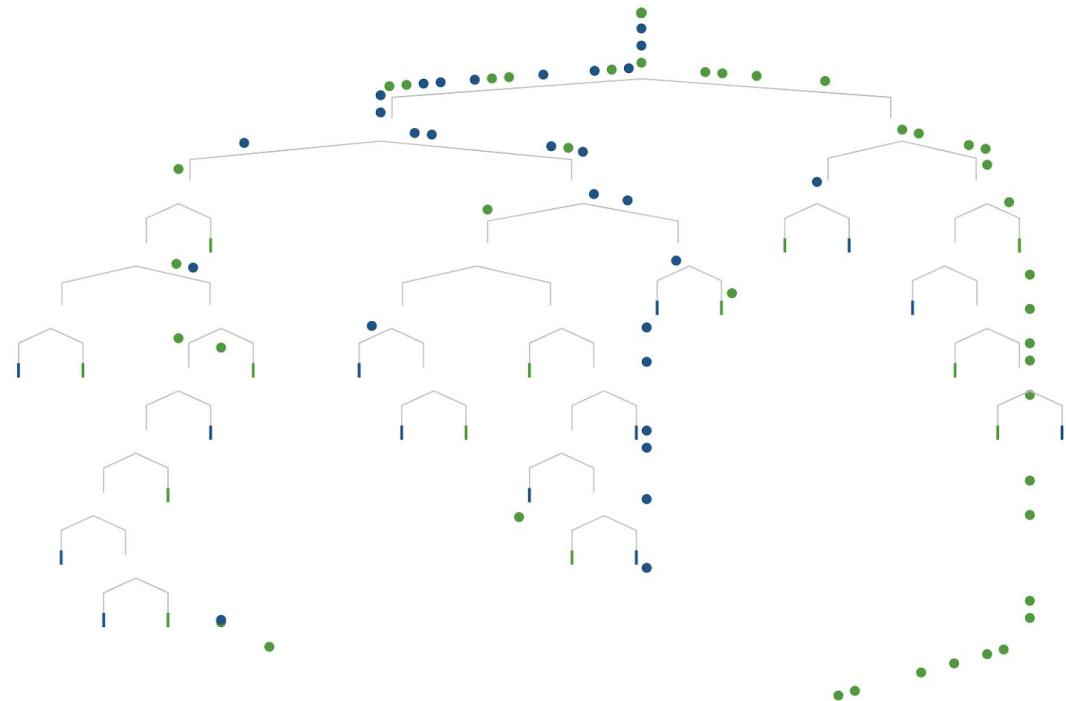


Accuracy: 100%

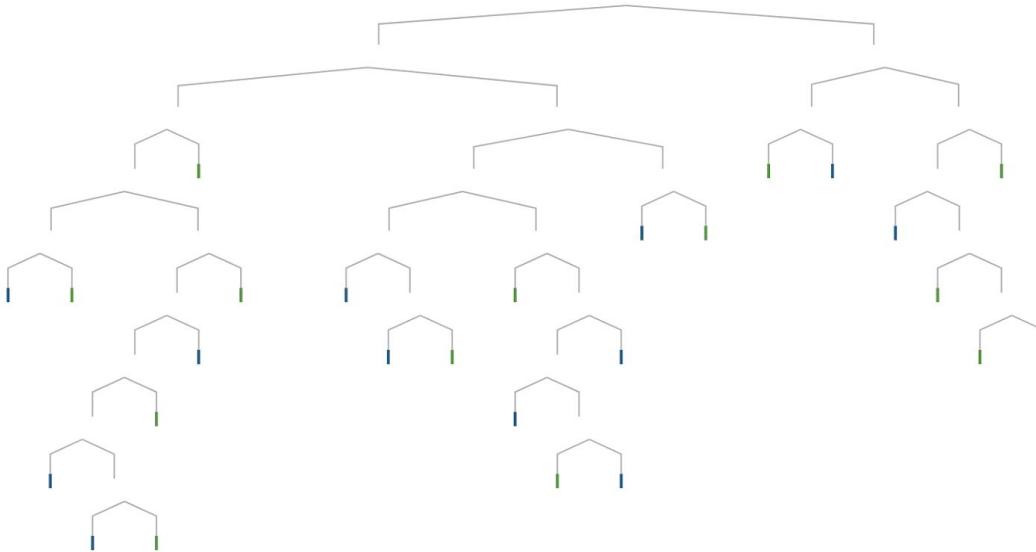
Making predictions

The decision tree **model** can then predict which homes are in which city.

Here, we're using the **training data**.



Because our tree was trained on this data and we grew the tree to 100% accuracy, each house is perfectly sorted



111/111

Training Accuracy
100%

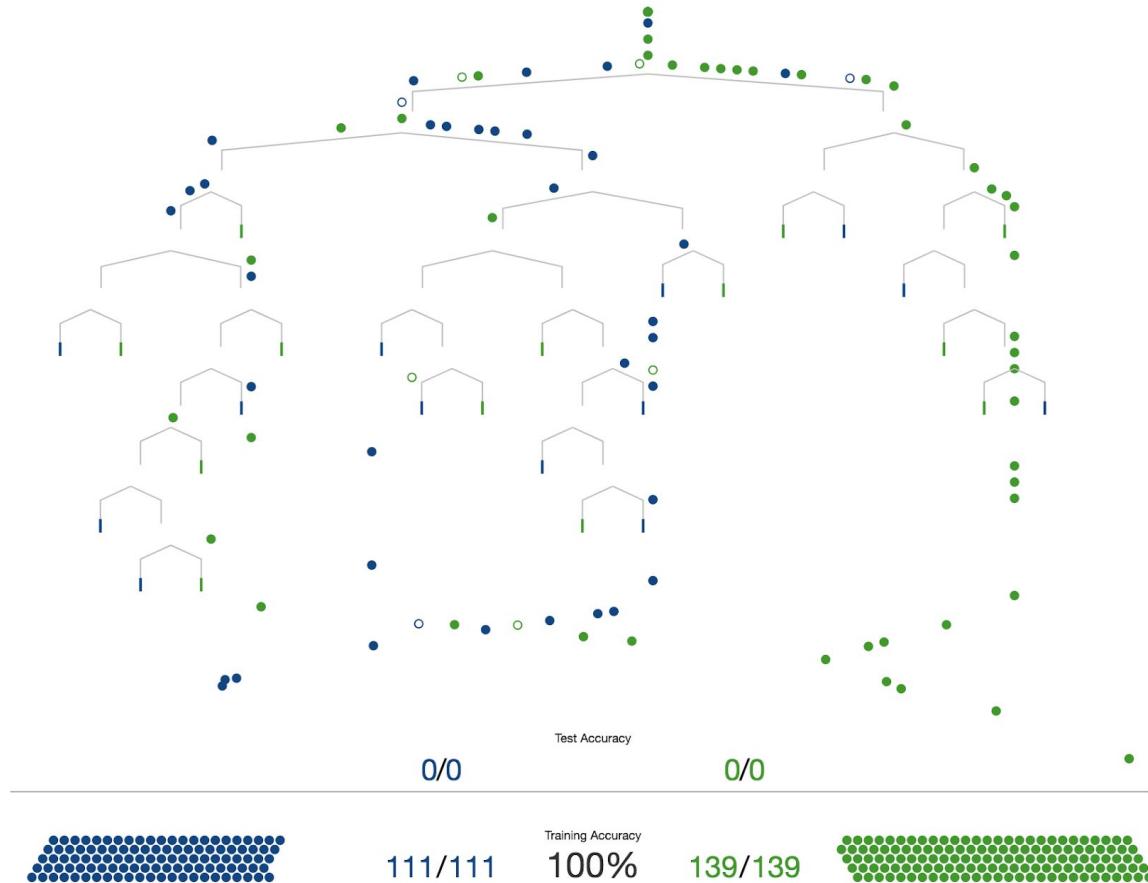
139/139



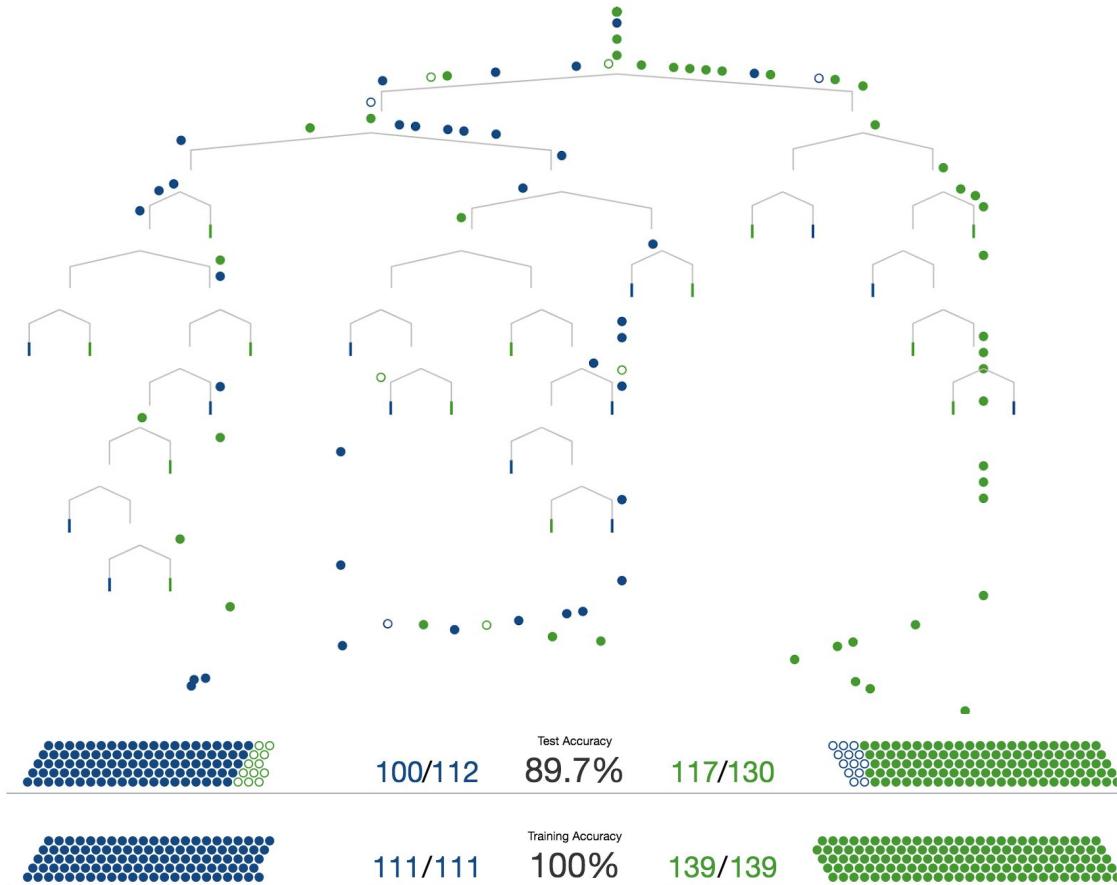
Reality check

But...how does this tree
on data that the model
hasn't seen before?

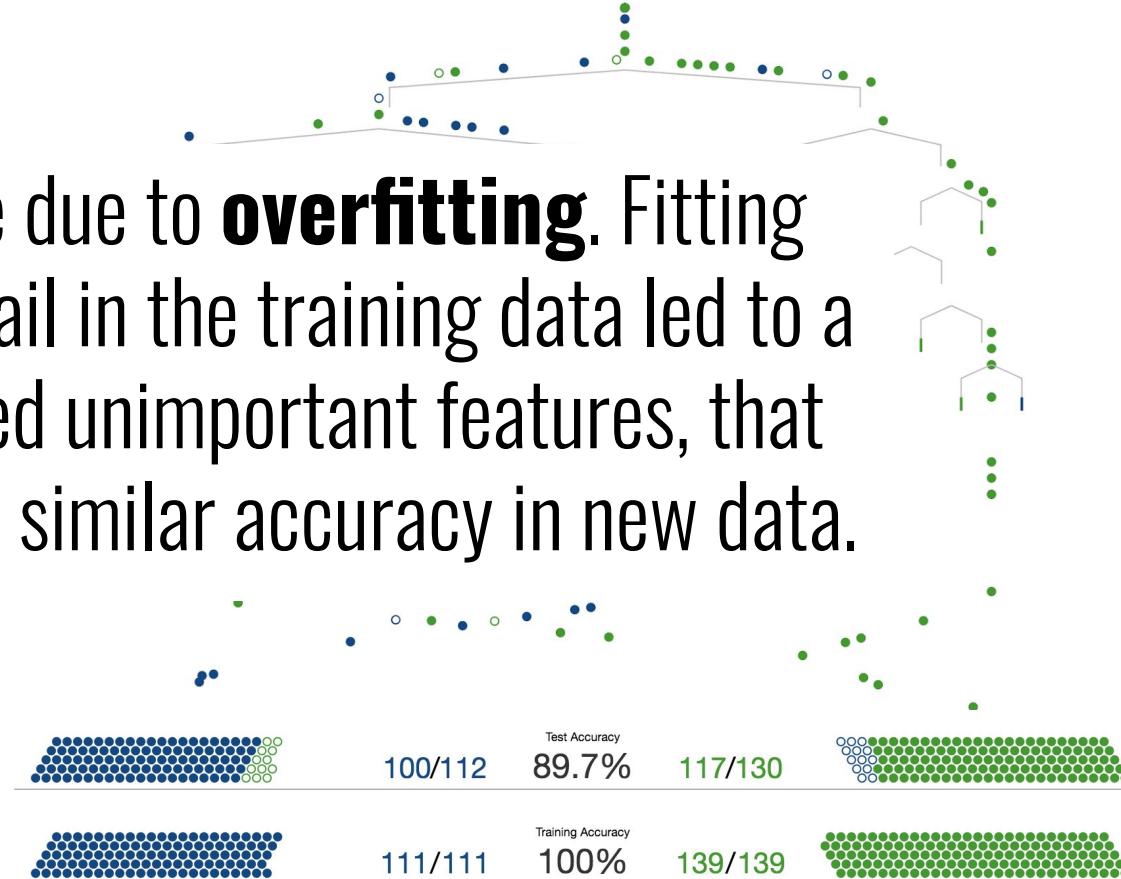
The **test set** then
makes it way through
the decision tree.



Ideally the tree should perform similarly on both known and unknown data



These errors are due to **overfitting**. Fitting every single detail in the training data led to a tree that modeled unimportant features, that did not allow for similar accuracy in new data.

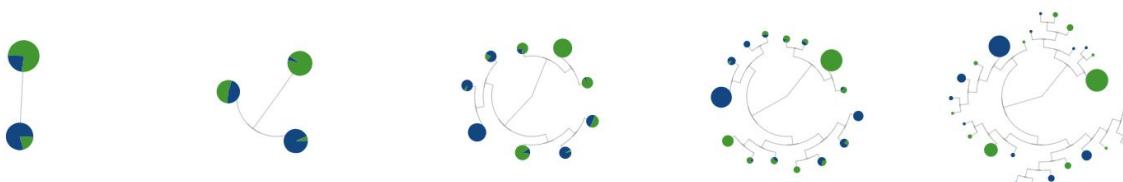
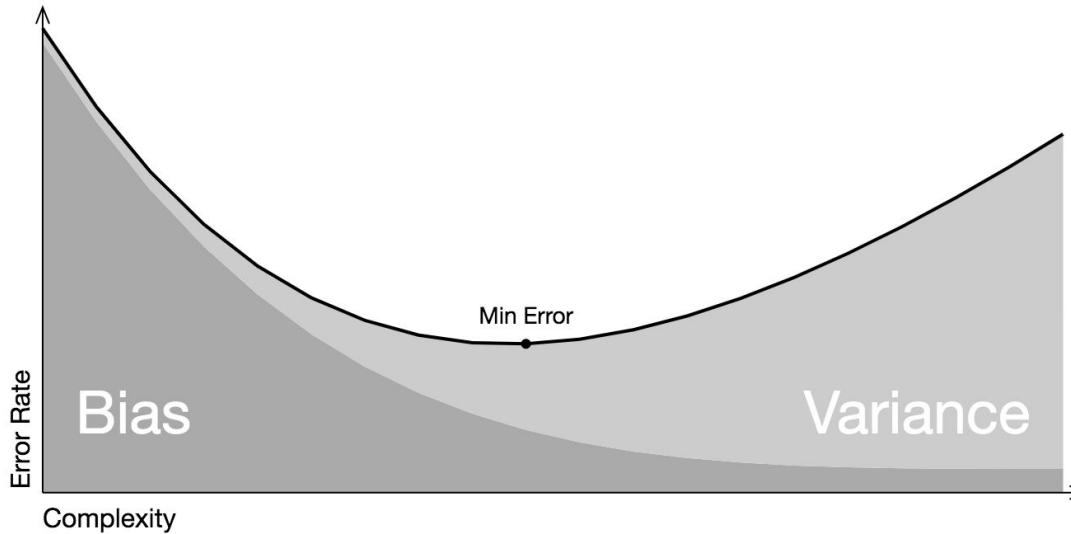


Recap

1. Machine learning identifies patterns using **statistical learning** and computers by unearthing **boundaries** in data sets. You can use it to make predictions.
2. One method for making predictions is called a decision trees, which uses a series of if-then statements to identify boundaries and define patterns in the data.
3. **Overfitting** happens when some boundaries are based on *on distinctions that don't make a difference*. You can see if a model overfits by having test data flow through the model.

So...what can we do
about overfitting?

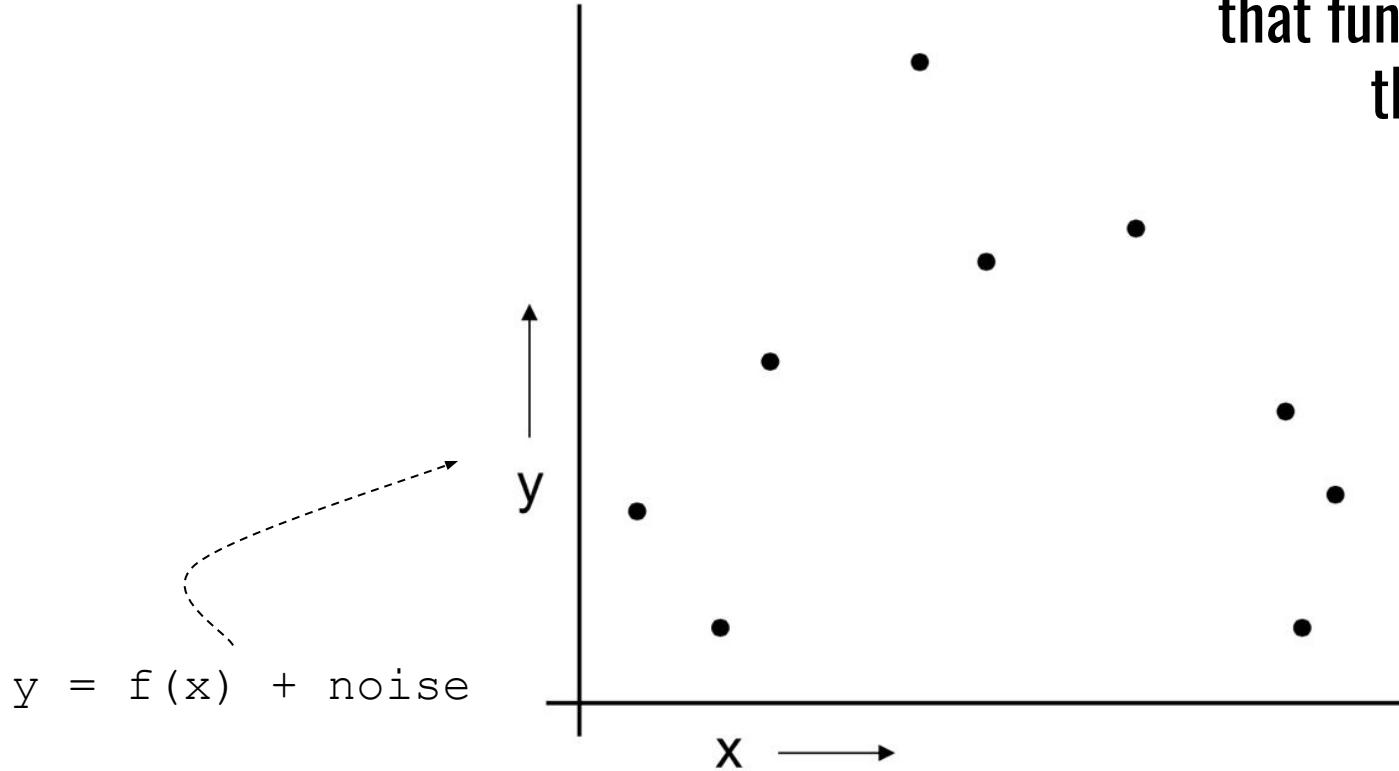
Bias-variance tradeoff



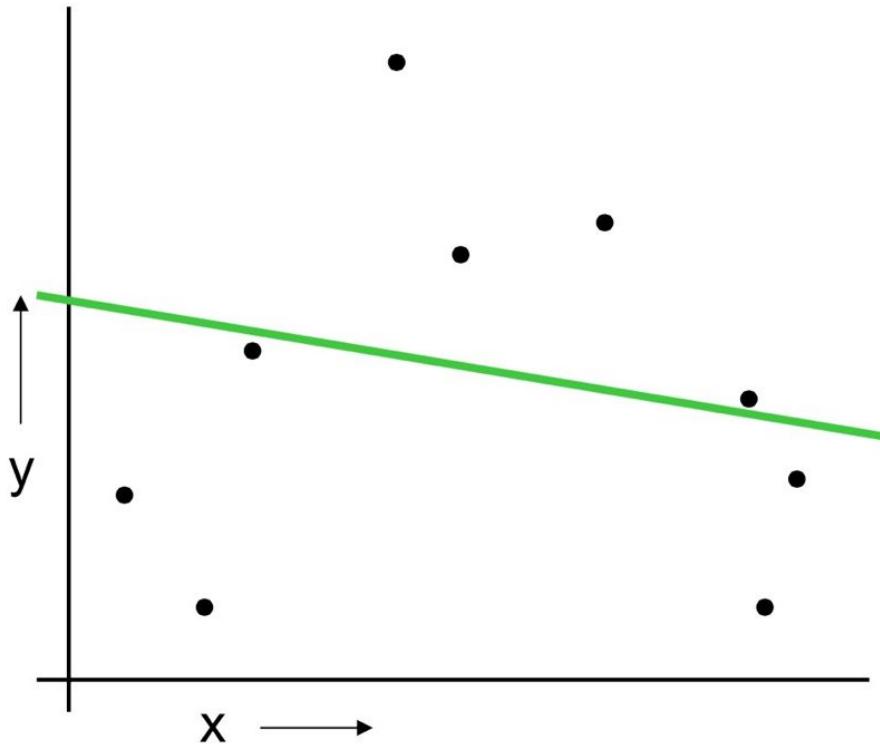
Bias-variance tradeoff

- **High variance** models make mistakes in *inconsistent* ways.
- **Biased models** tend to be overly simple and not reflect reality
- What to do:
 - Consider tuning parameters in the model
 - can avoid overfitting by setting minimum node size threshold (fewer splits; variance decreased)
 - Changing model approach
 - Bagging, boosting, & ensemble methods
 - Re-consider data splitting approach
 - Training + test?
 - LOOCV
 - K-fold CV

Can we determine what
that function (f) *is* using
these data?

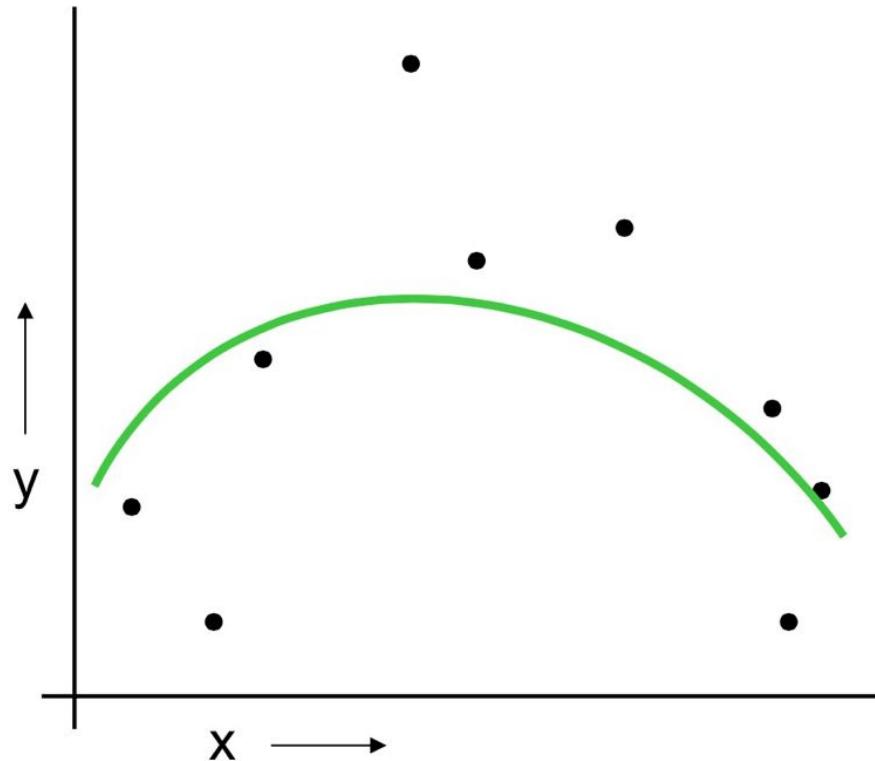


Linear regression



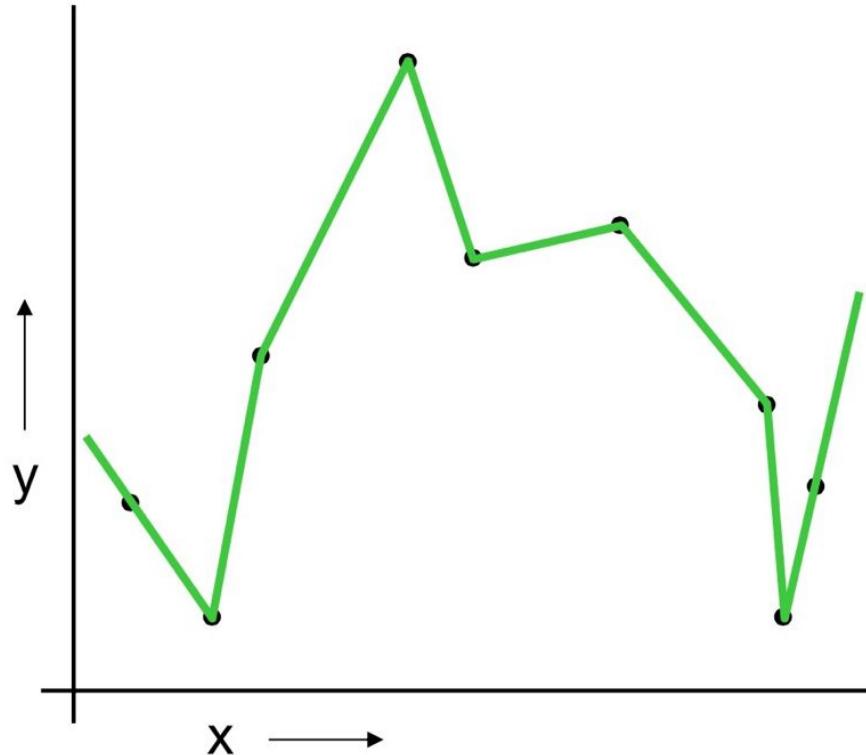
adapted from Brad Voytek

Quadratic regression



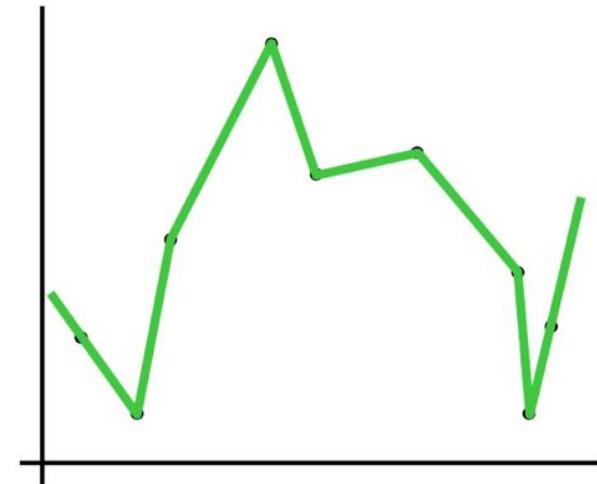
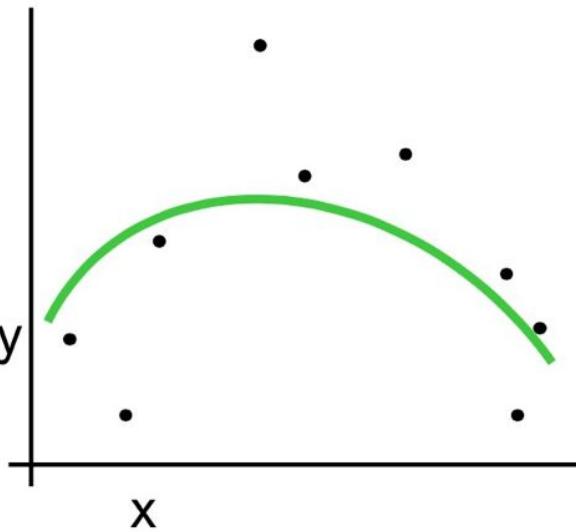
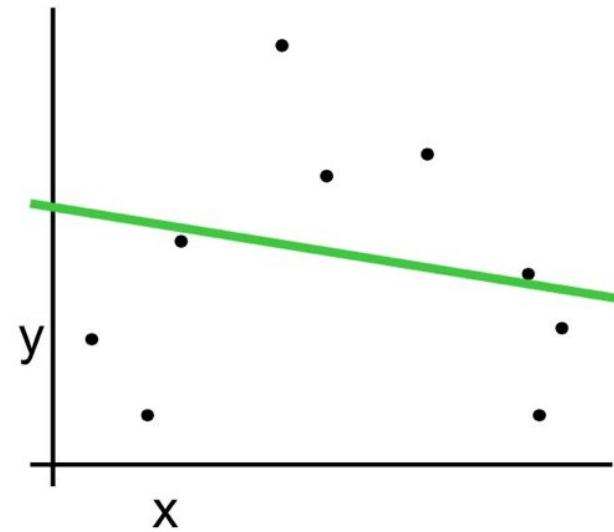
adapted from Brad Voytek

Piecewise linear nonparametric regression



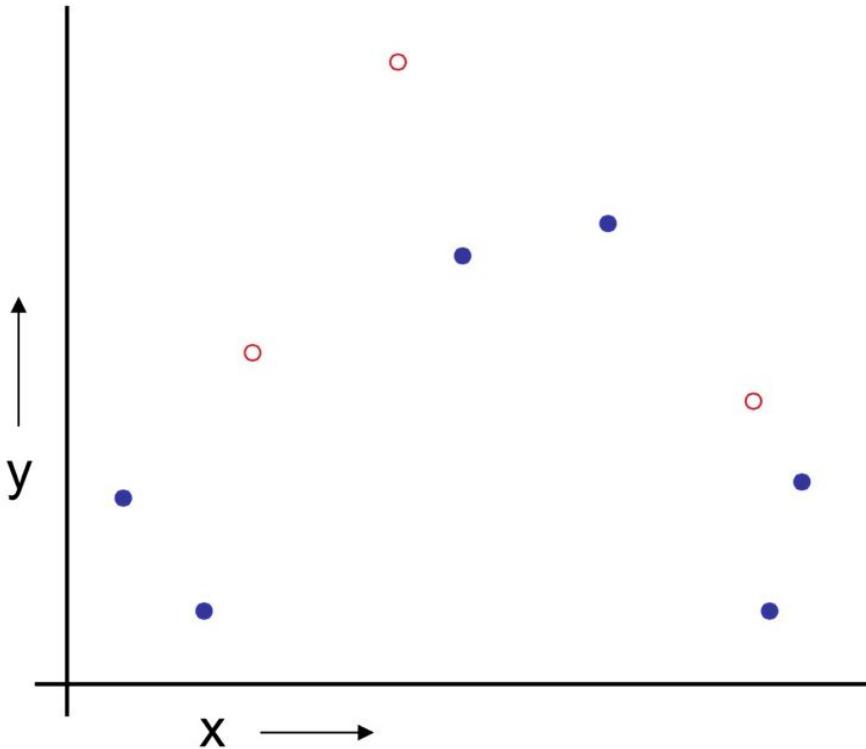
adapted from Brad Voytek

Which to choose?



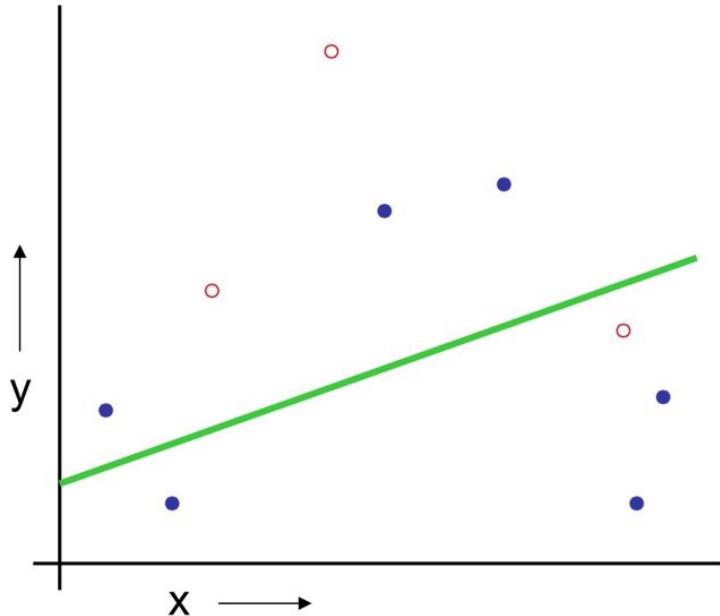
adapted from Brad Voytek

The data partition method



1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**

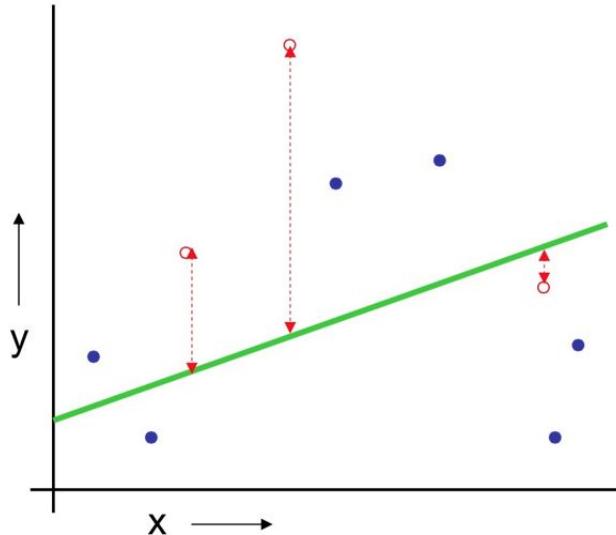
Train the model on your training set



1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**
3. Perform your regression on the training set

(Linear regression example)

Assess future performance using the **test set**



(Linear regression example)

Mean Squared Error = 2.4

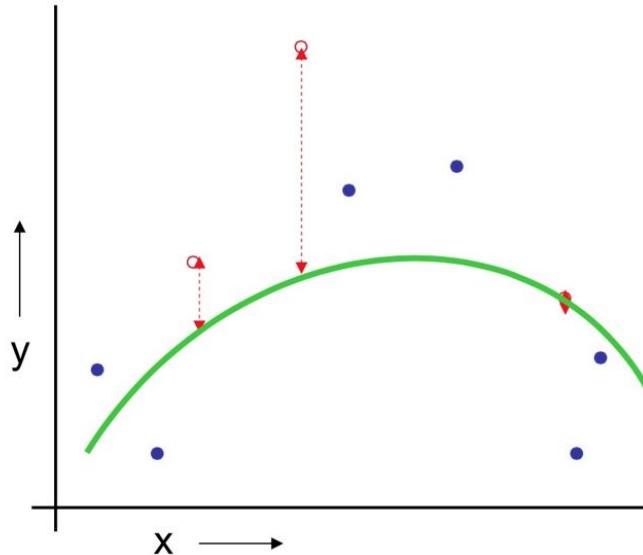
1. Randomly choose
30% of the data to be in a
test set

2. The remainder is a
training set

3. Perform your
regression on the training
set

4. Estimate your future
performance with the test
set

Go through this process for each possible model

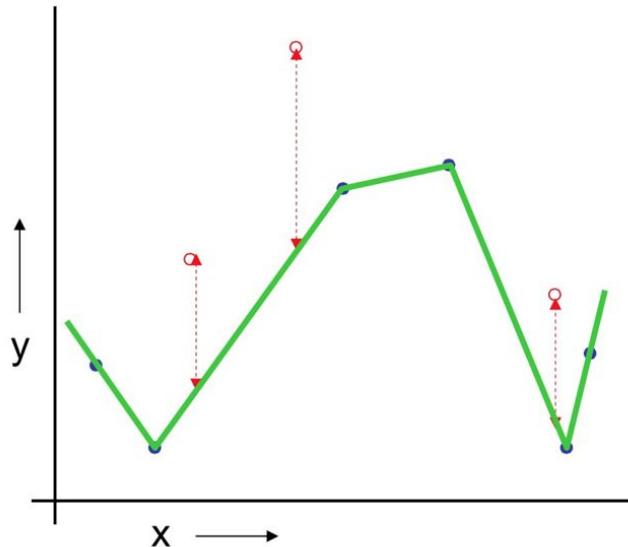


(Quadratic regression example)

Mean Squared Error = 0.9

1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**
3. Perform your regression on the training set
4. Estimate your future performance with the test set

Go through this process for each possible model



(Join the dots example)

Mean Squared Error = 2.2

1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a training set
3. Perform your regression on the training set
4. Estimate your future performance with the test set

Pros and cons of data partitioning

Pros:

- Simple approach
- Can choose model with best test-set score

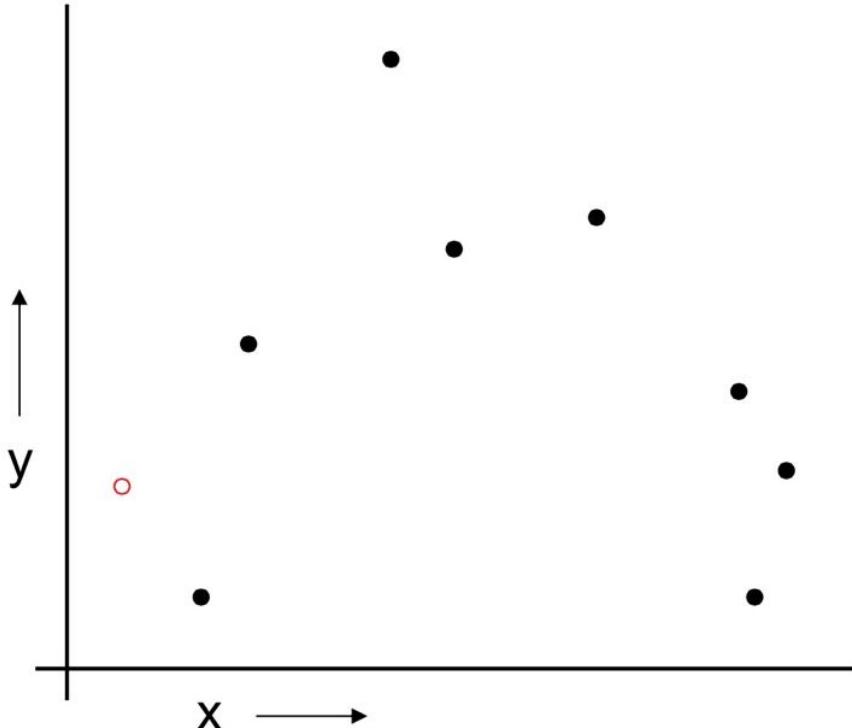
Cons:

- Model fit on 30% less data than you have
- Without a large data set, removing 30% of the data could bias prediction

Leave one out cross validation (LOOCV)

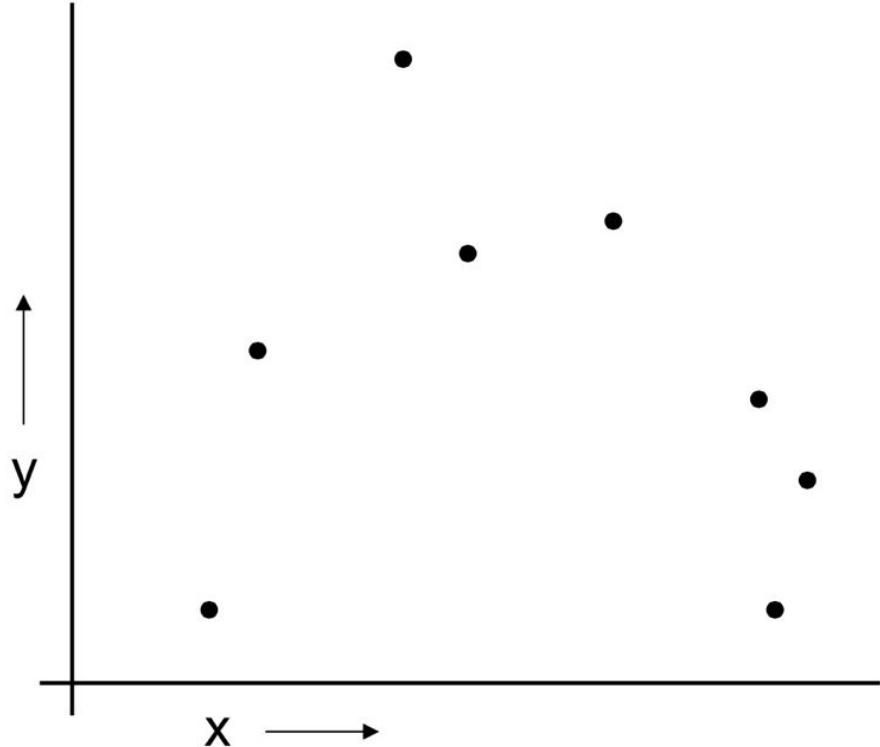
For k=1 to R

1. Let (x_k, y_k) be the k^{th} record



Leave one out cross validation (LOOCV)

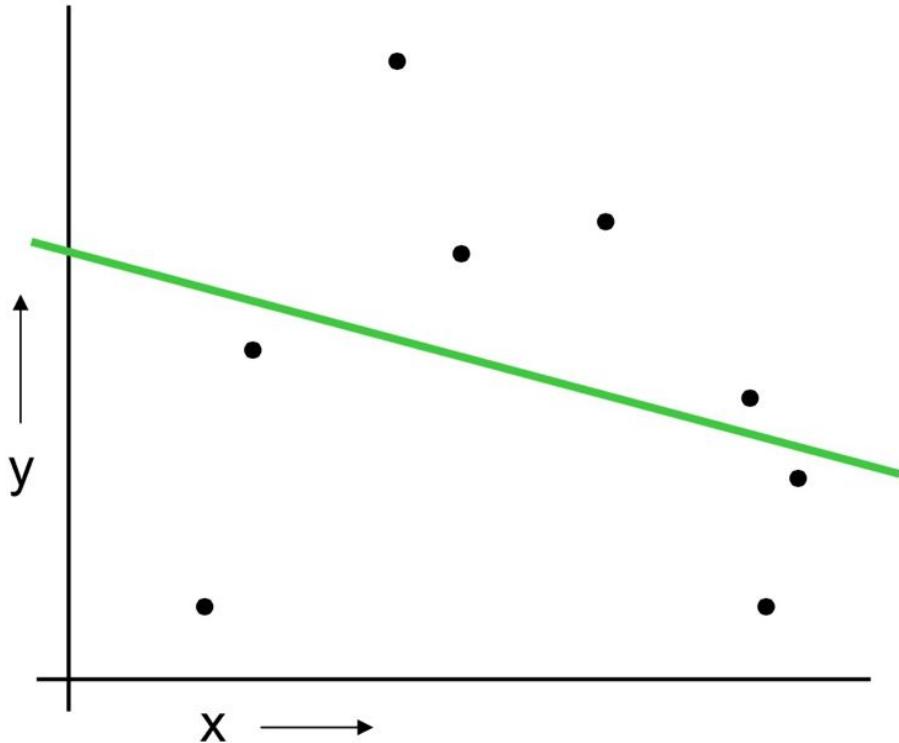
For k=1 to R



1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset

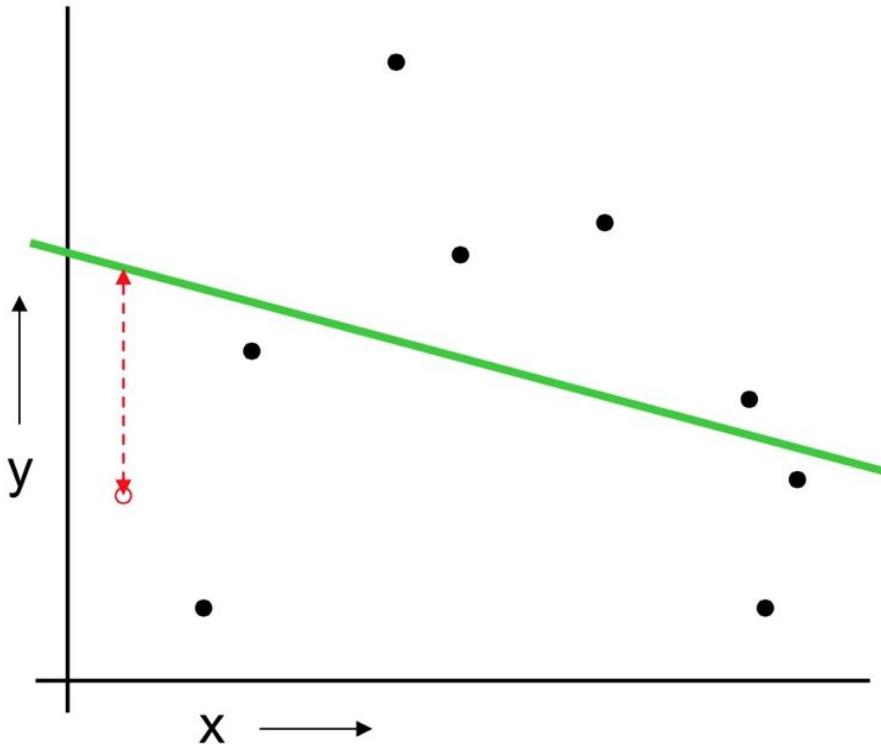
Leave one out cross validation (LOOCV)

For k=1 to R



1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints

Leave one out cross validation (LOOCV)

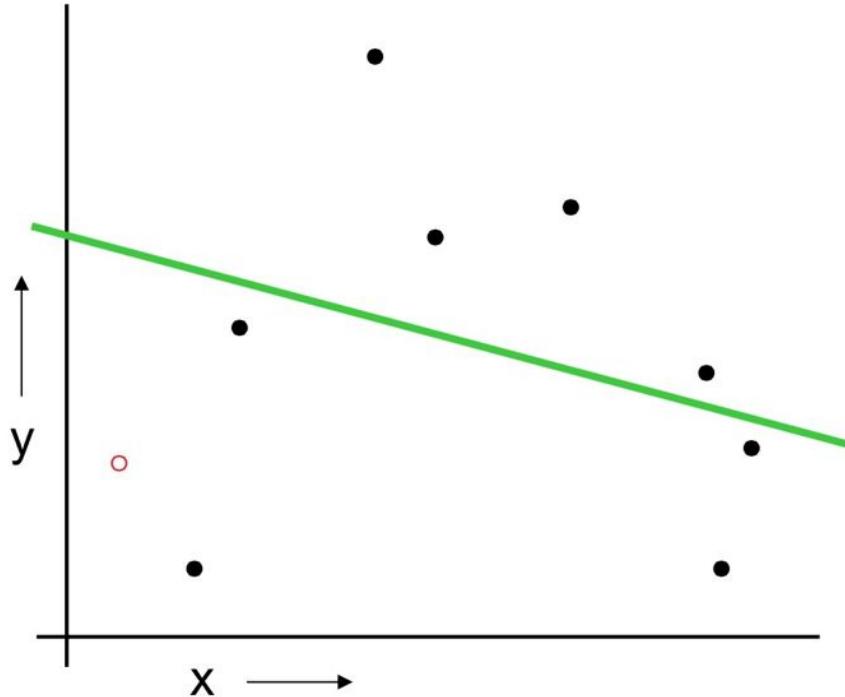


For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

Leave one out cross validation (LOOCV)

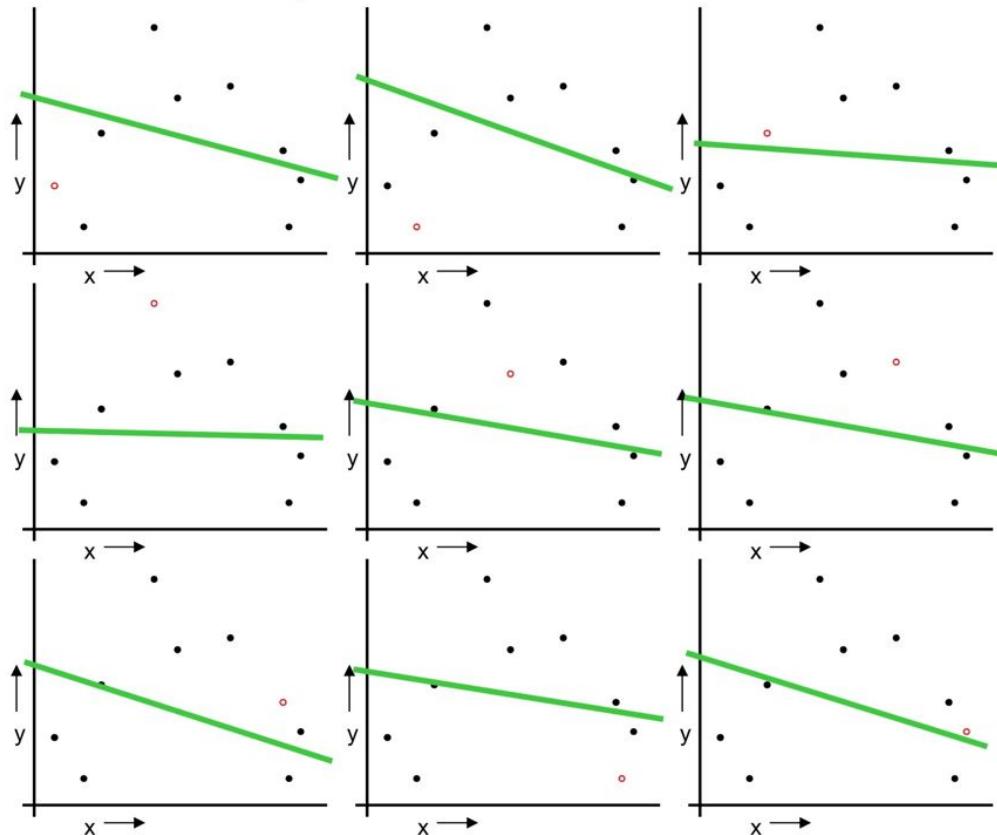
For k=1 to R



1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points,
report the mean error.

Leave one out cross validation (LOOCV)



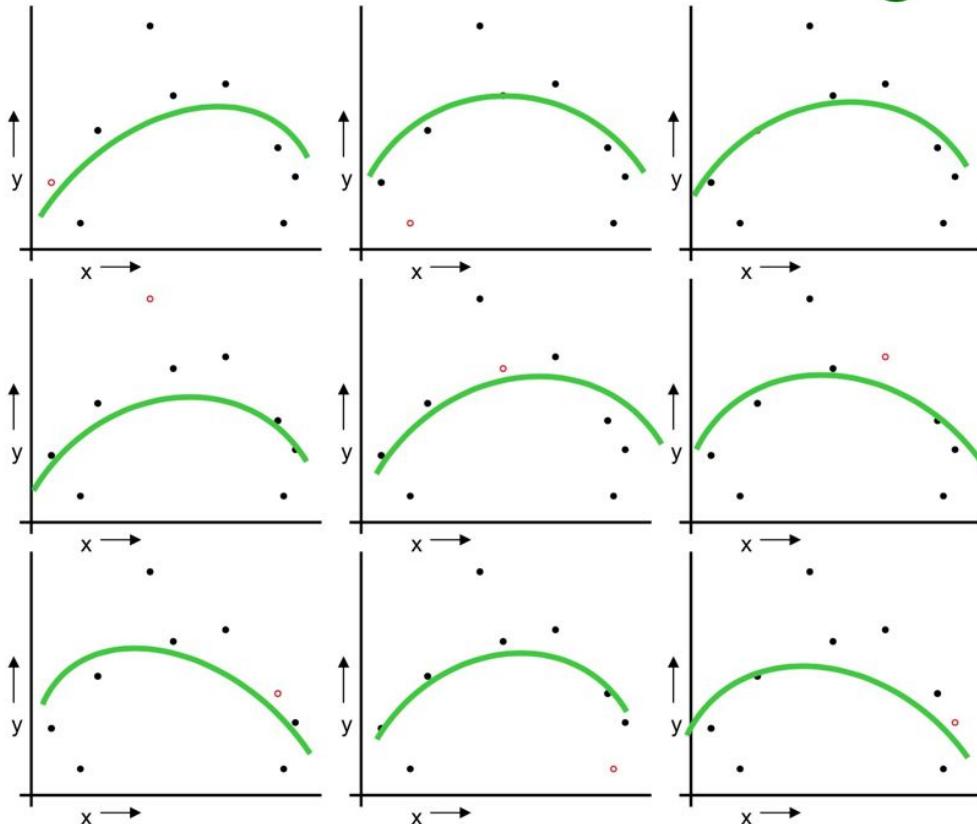
For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{\text{LOOCV}} = 2.12$$

Leave one out cross validation (LOOCV)



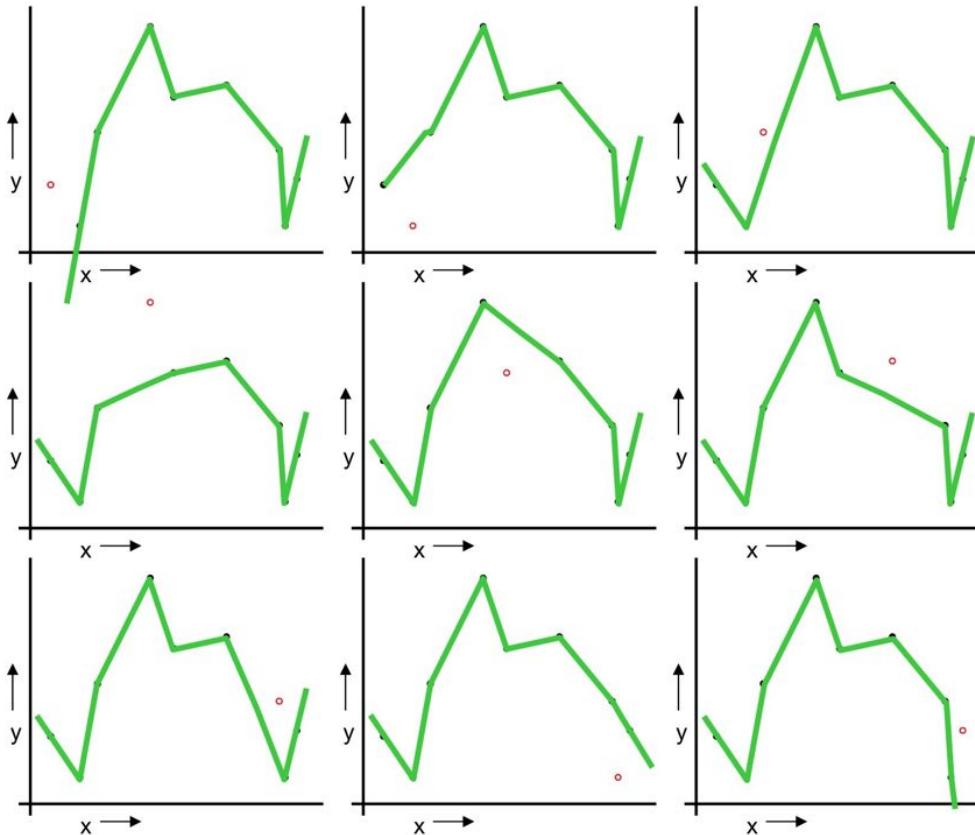
For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{\text{LOOCV}} = 0.962$$

Leave one out cross validation (LOOCV)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

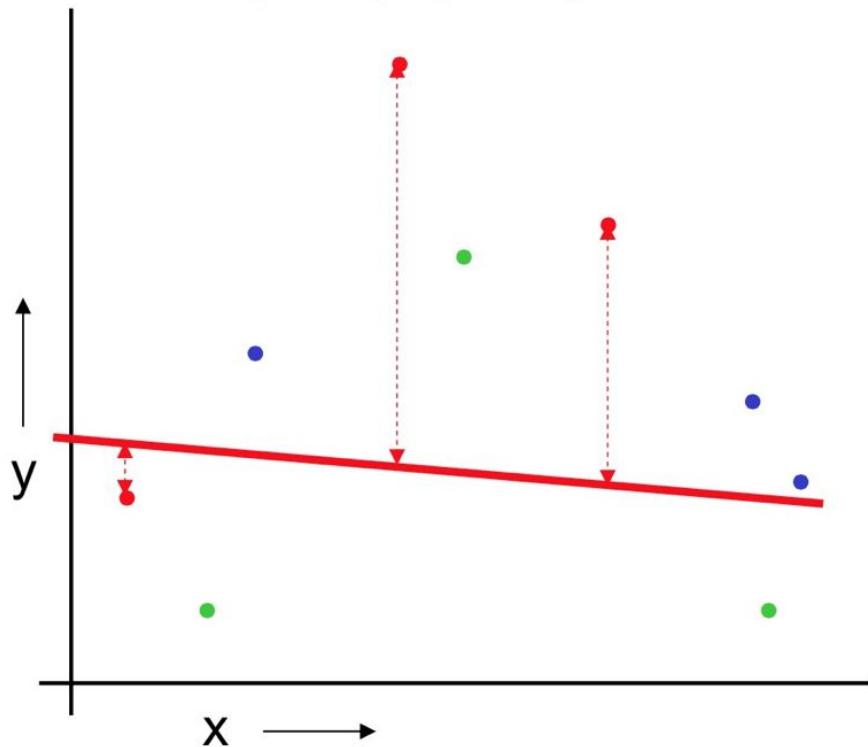
When you've done all points, report the mean error.

$$MSE_{\text{LOOCV}} = 3.33$$

Method Comparison

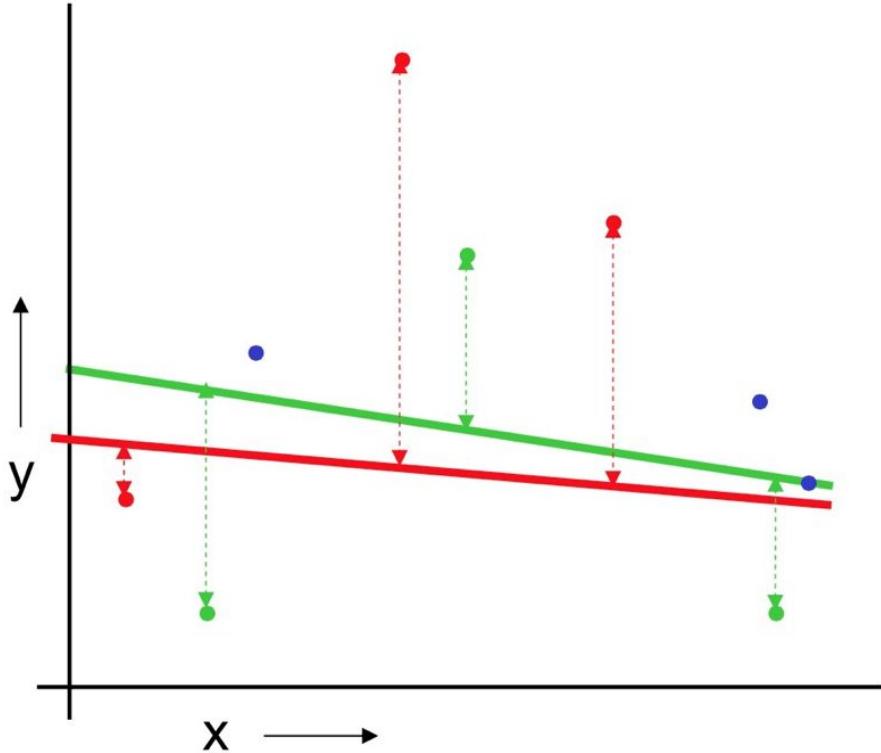
	Cons	Pros
Data partitioning	Variance: unreliable estimate of future performance	Cheap
LOOCV	Computationally expensive; has weird behavior	Uses all your data

k -fold cross validation



For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

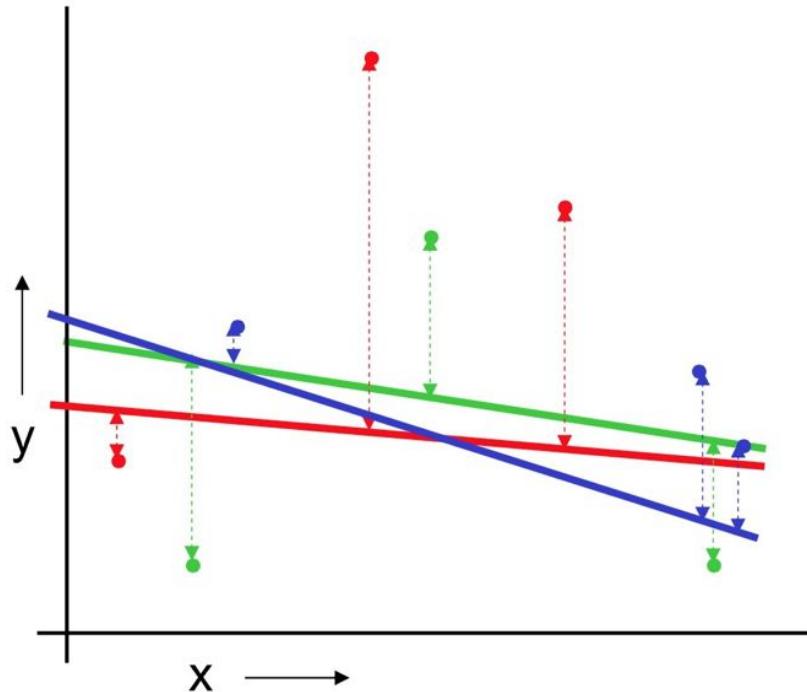
k -fold cross validation



For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

k -fold cross validation

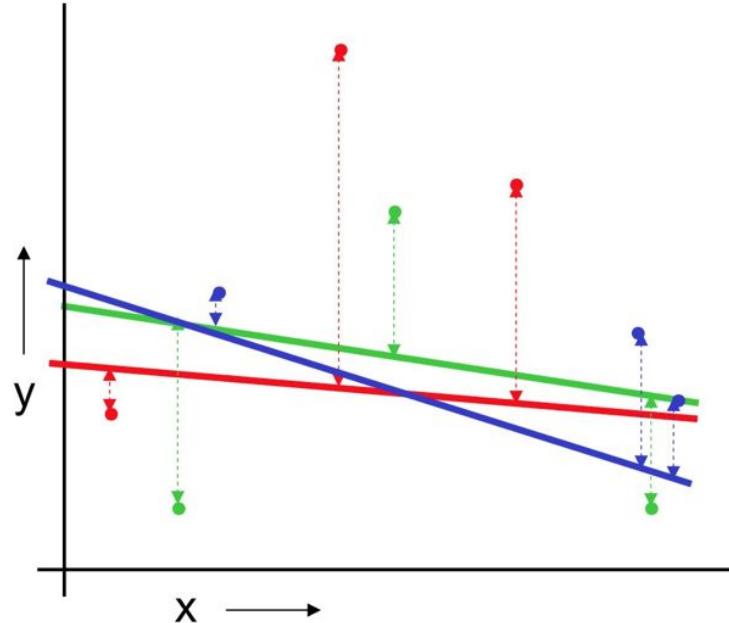


For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

k -fold cross validation



Linear Regression

$$MSE_{3FOLD} = 2.05$$

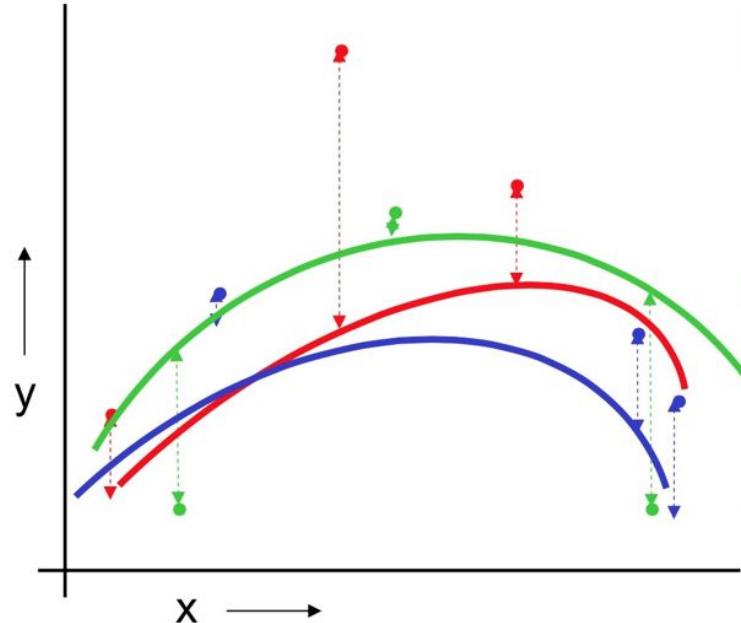
For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

k -fold cross validation



Quadratic Regression

$$MSE_{3FOLD} = 1.11$$

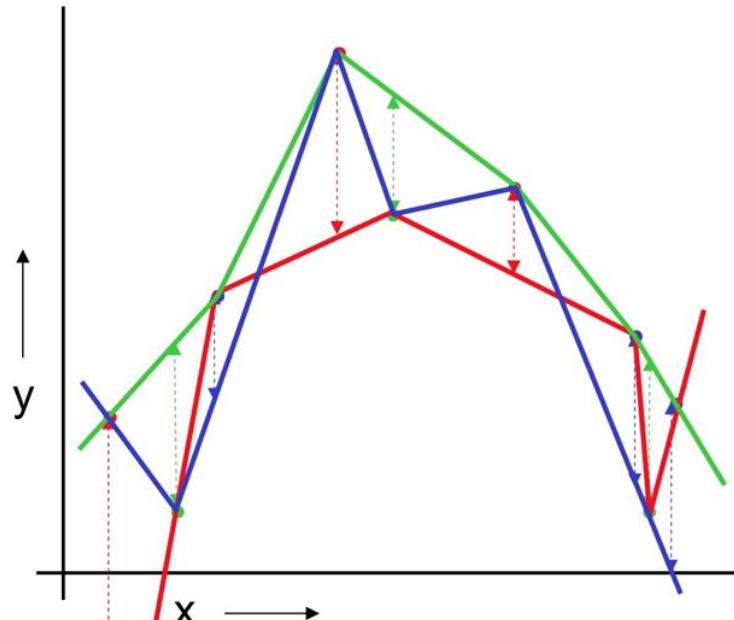
For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

k -fold cross validation



Joint-the-dots

$$MSE_{3FOLD} = 2.93$$

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error



Validator

Given the example we just worked, how would
you model these data?

A
linear
regression

Linear Regression
 $MSE_{3FOLD}=2.05$

B
quadratic
regression

Quadratic Regression
 $MSE_{3FOLD}=1.11$

C
pairwise linear
nonparametric
regression

Joint-the-dots
 $MSE_{3FOLD}=2.93$



Validator

Which approach would *you* use to limit overfitting?

