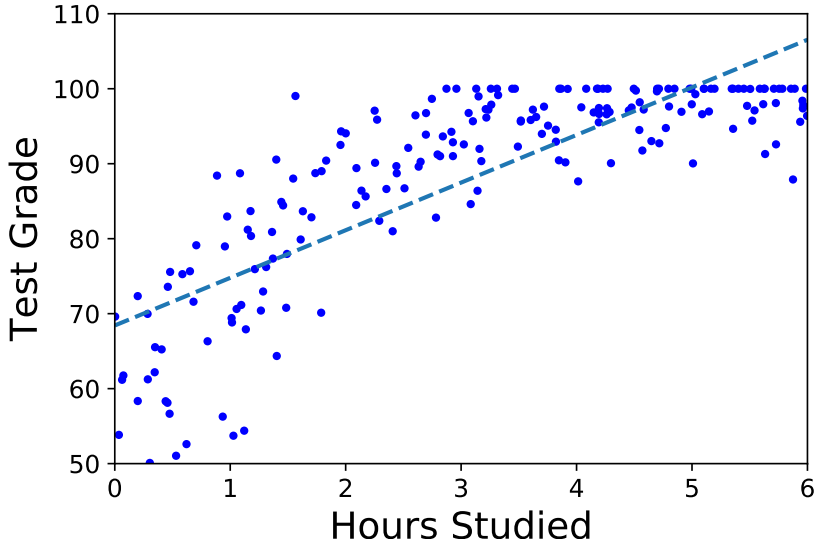
Lecture 3:
Nearest Neighbors and Model Selection

David Carlson

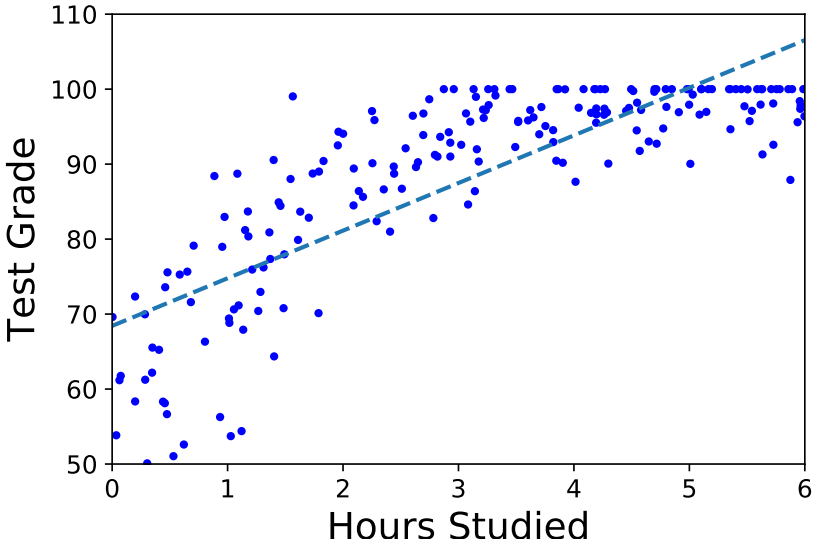
Quick Recap

- Last time we discussed:
  - Linear regression
  - Logistic regression
  - Mathematical goals
- Linear methods can be limited…



Can we predict the test score better?

- Want to capture the effect of “diminishing returns” in the data
- Can we construct automatic ways of “learning” this as we get more data?



An initial machine learning algorithm

K-NEAREST NEIGHBORS

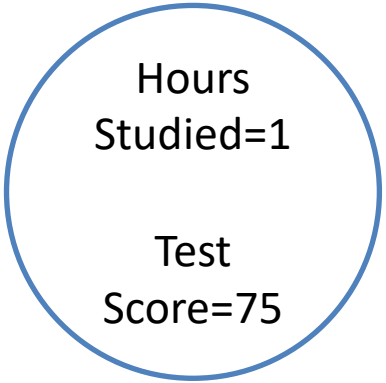
Case-Based Reasoning

- One way to think about making predictions is to think about *similarities* to other cases
- I.E. patient 1 has all the symptoms of patient 2; if patient 2 has the flu so does patient 1…

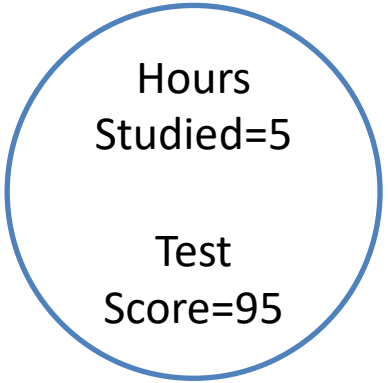
Nearest Neighbor Classifier

- One rule is conceptually:
  - 1. Receive a new data point
  - 2. Find the “nearest neighbor” in the training dataset (i.e. smallest distance between points)
  - 3. Predict the new data point is the same class as the “nearest neighbor”

Quick Example

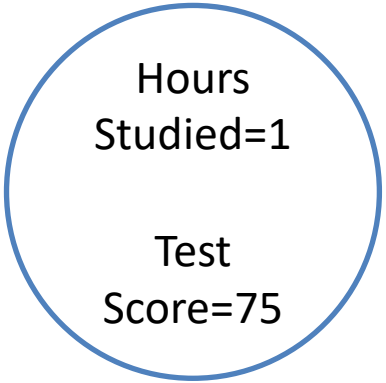


Student A

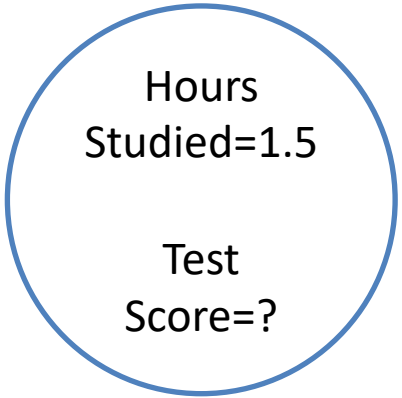


Student B

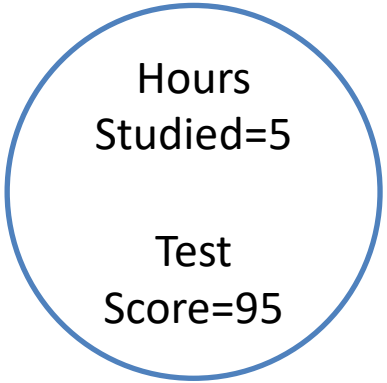
Quick Example



Student A

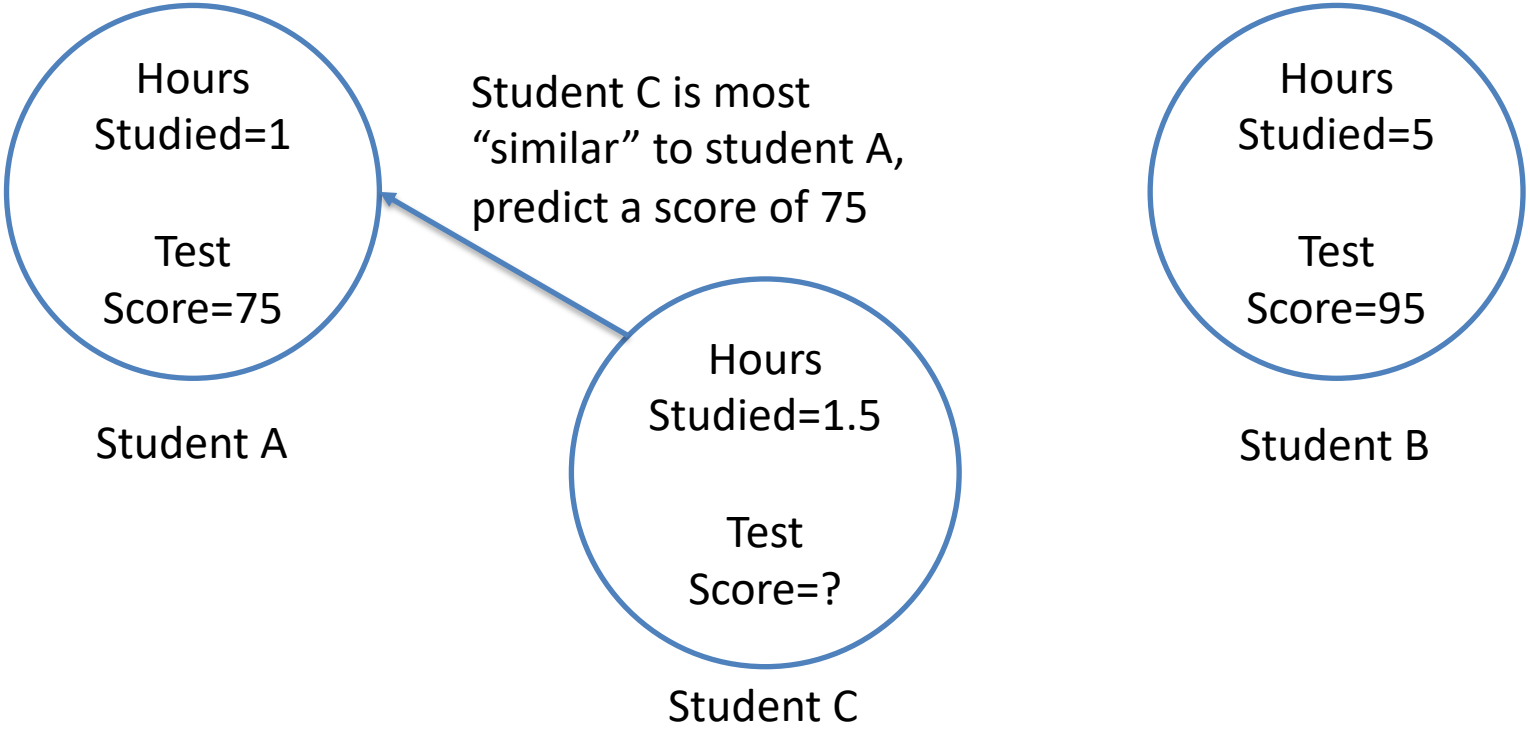


Student C

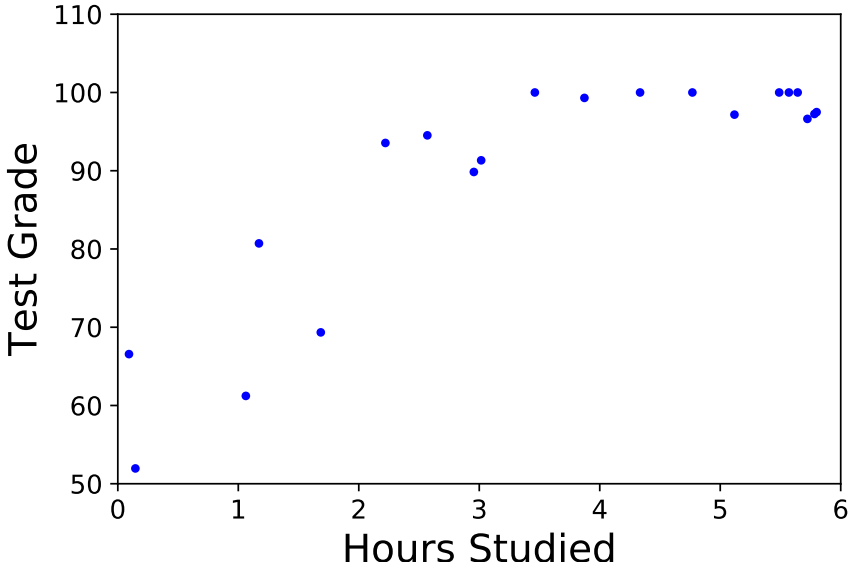


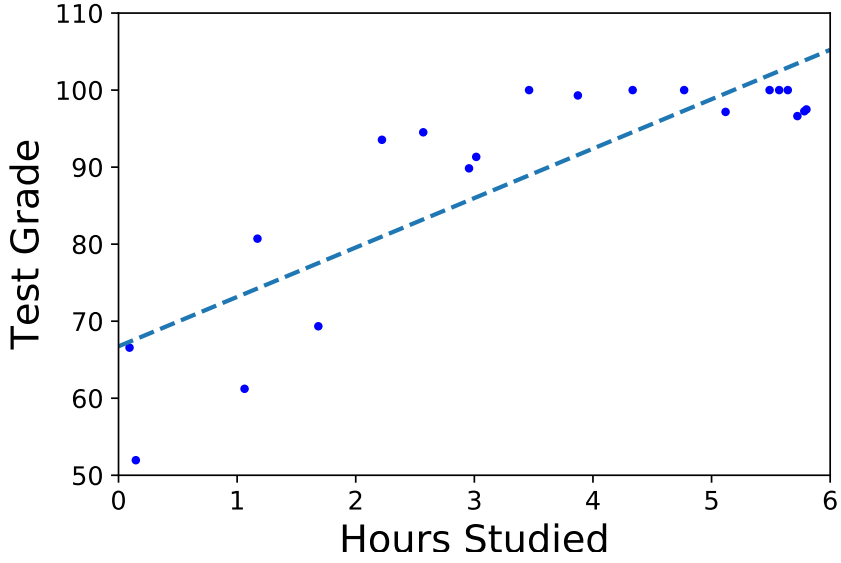
Student B

Quick Example

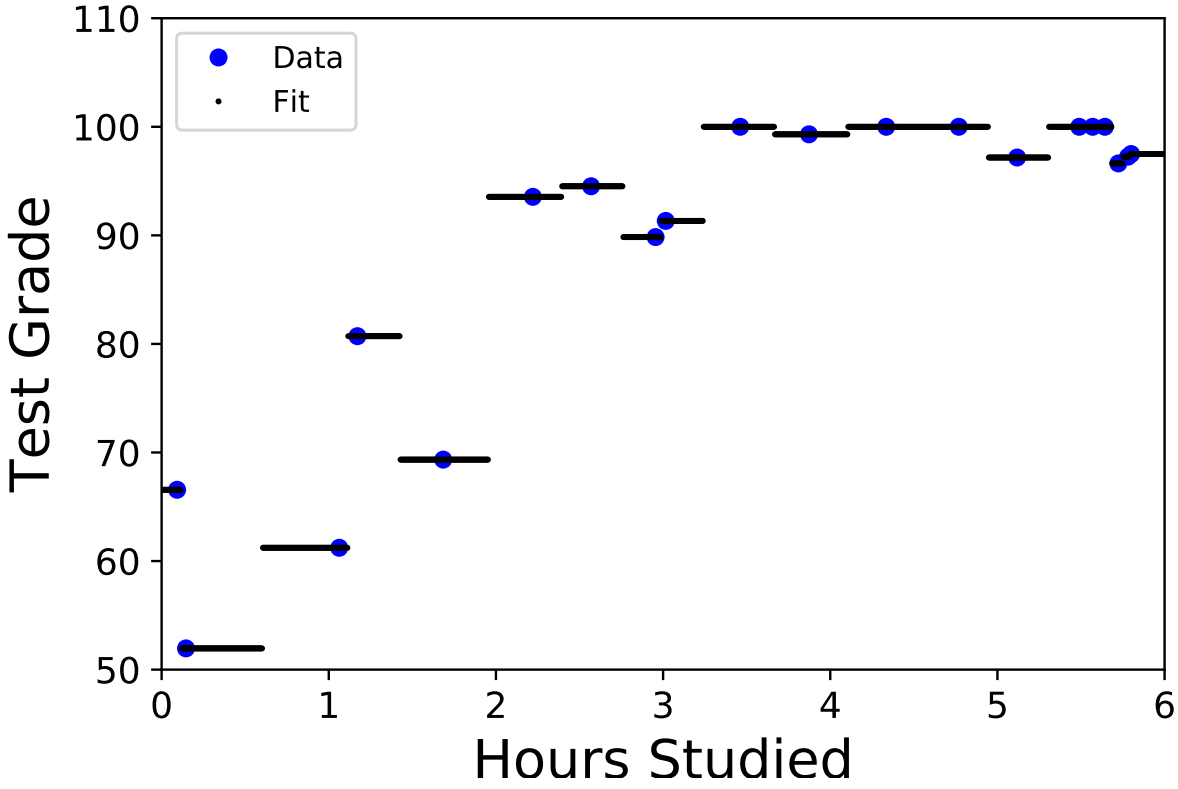


Does this strategy predict our data better?

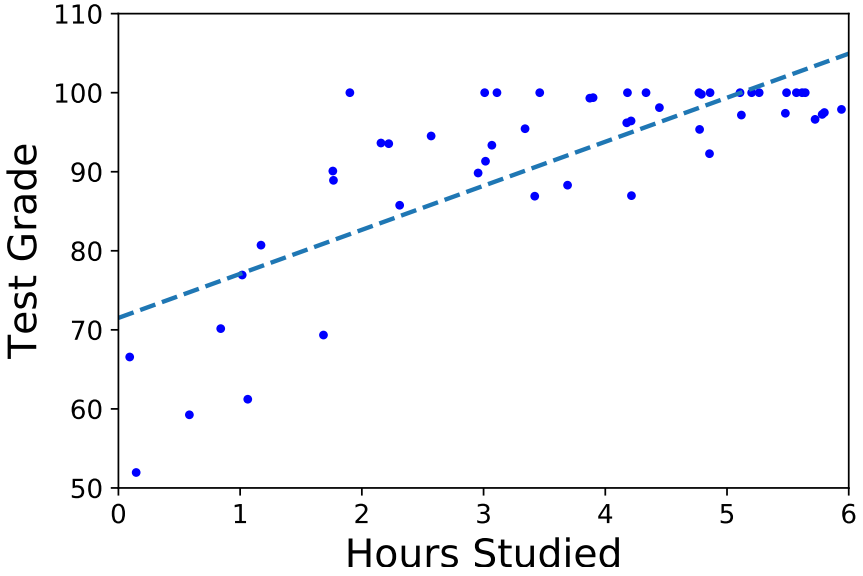


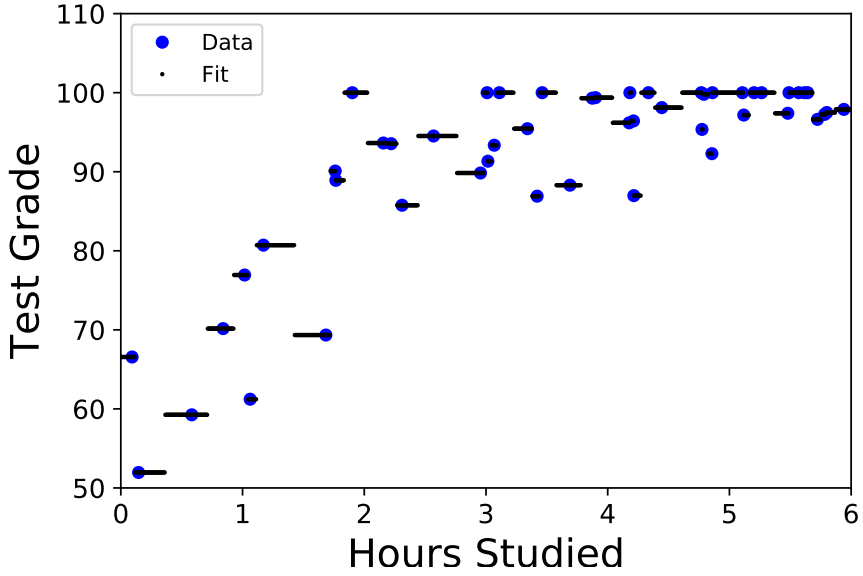


Nearest Neighbor Visualization

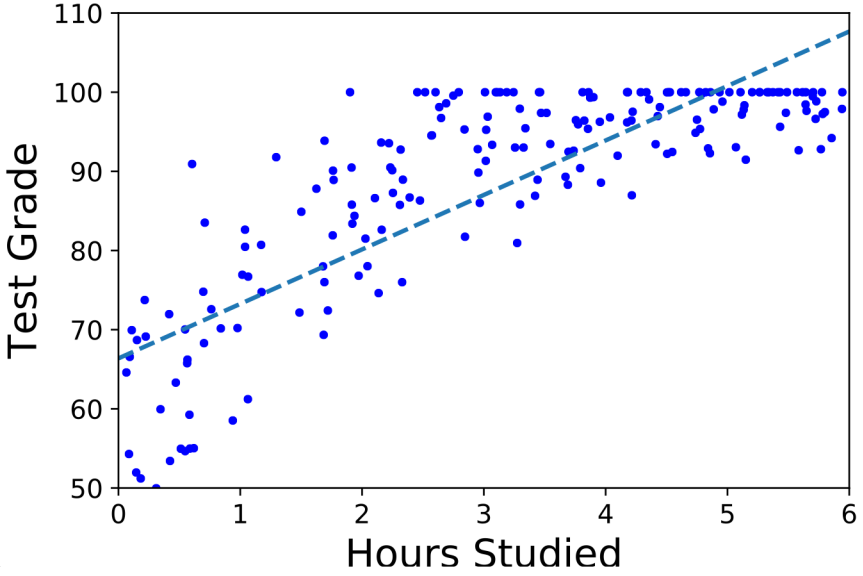


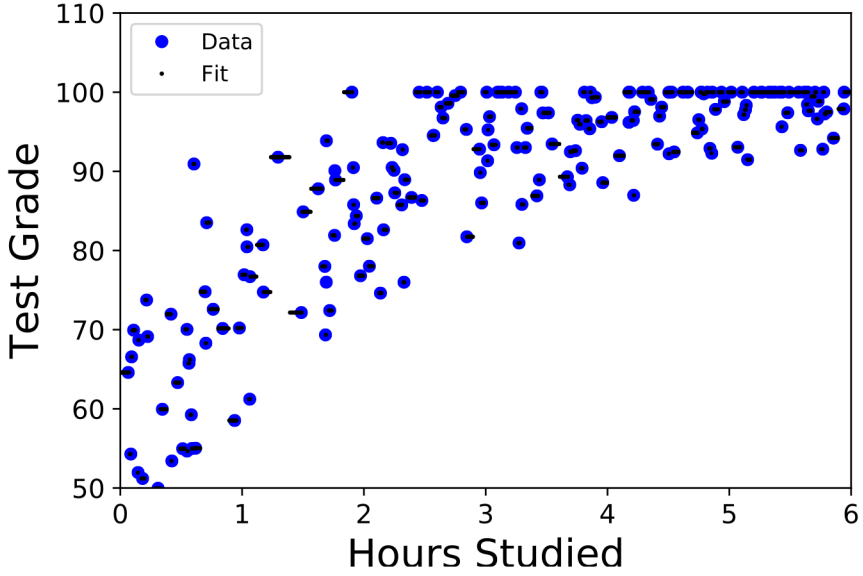
Increasing Data





Even More Data





Some comments:

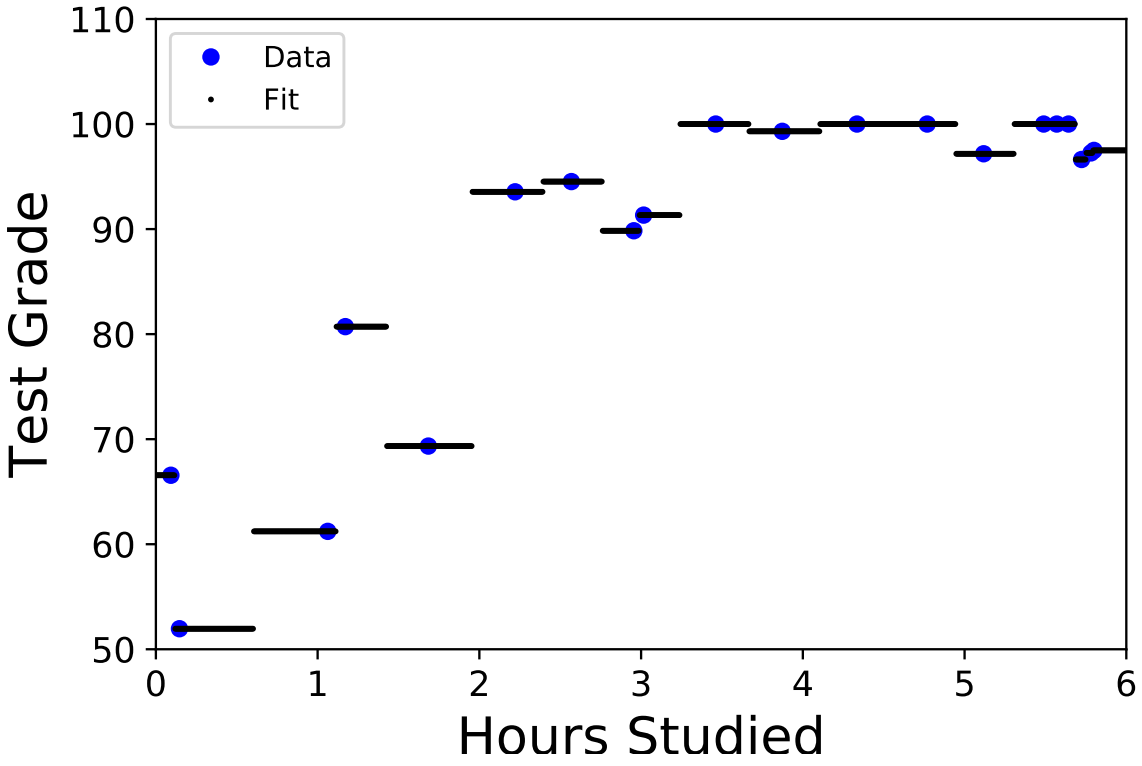
- Linear regression doesn't really change when we get more data
- The nearest neighbors approaches gets more complicated
- Nearest neighbors is very disjointed; may look funky, can we fix this?

K-Nearest Neighbors

- Using one neighbor might be too simplistic and erratic
- One approach:
  - Find the *K* closest points (i.e. the *K* nearest neighbors)
  - Predict using the average value of those *K* points

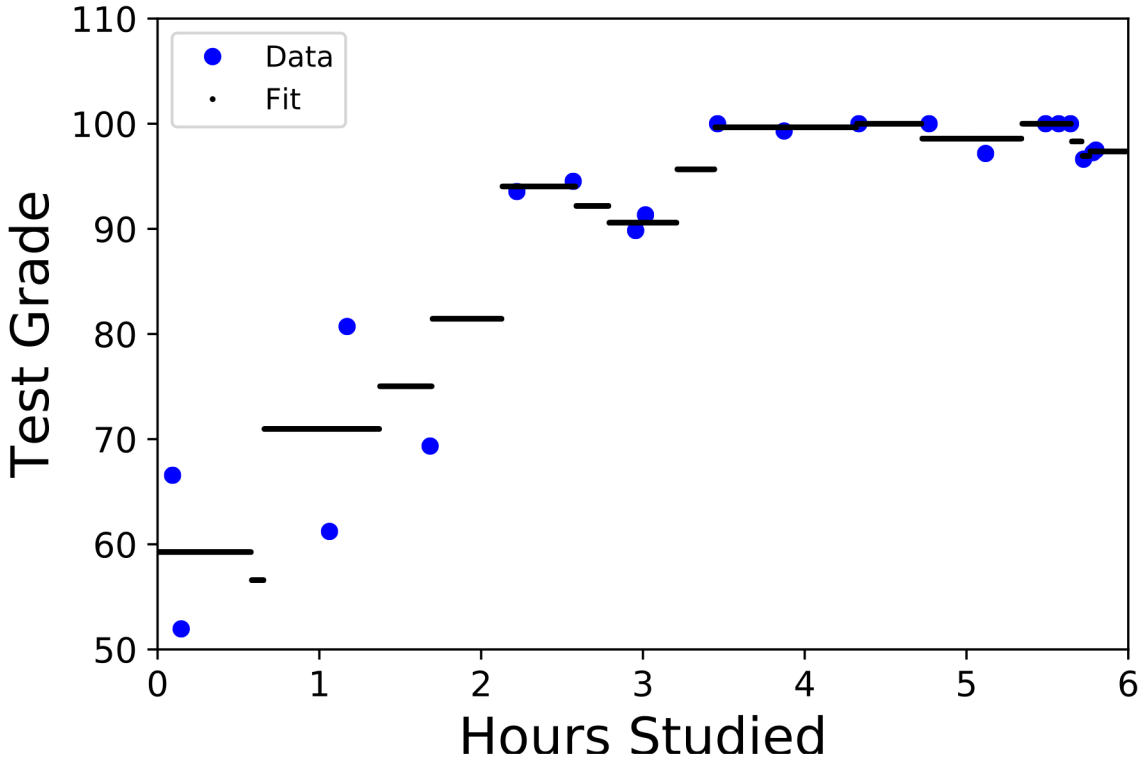
K-Nearest Neighbors

K=1 with 20 data points



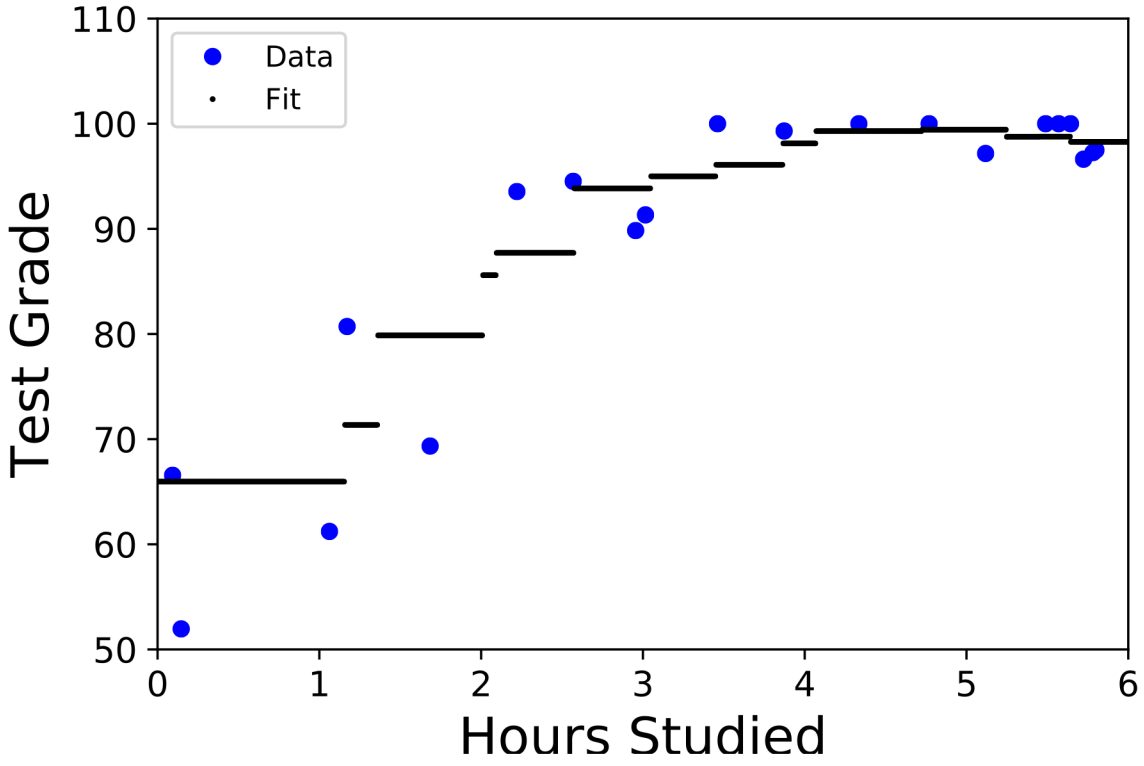
K-Nearest Neighbors

K=2 with 20 data points



K-Nearest Neighbors

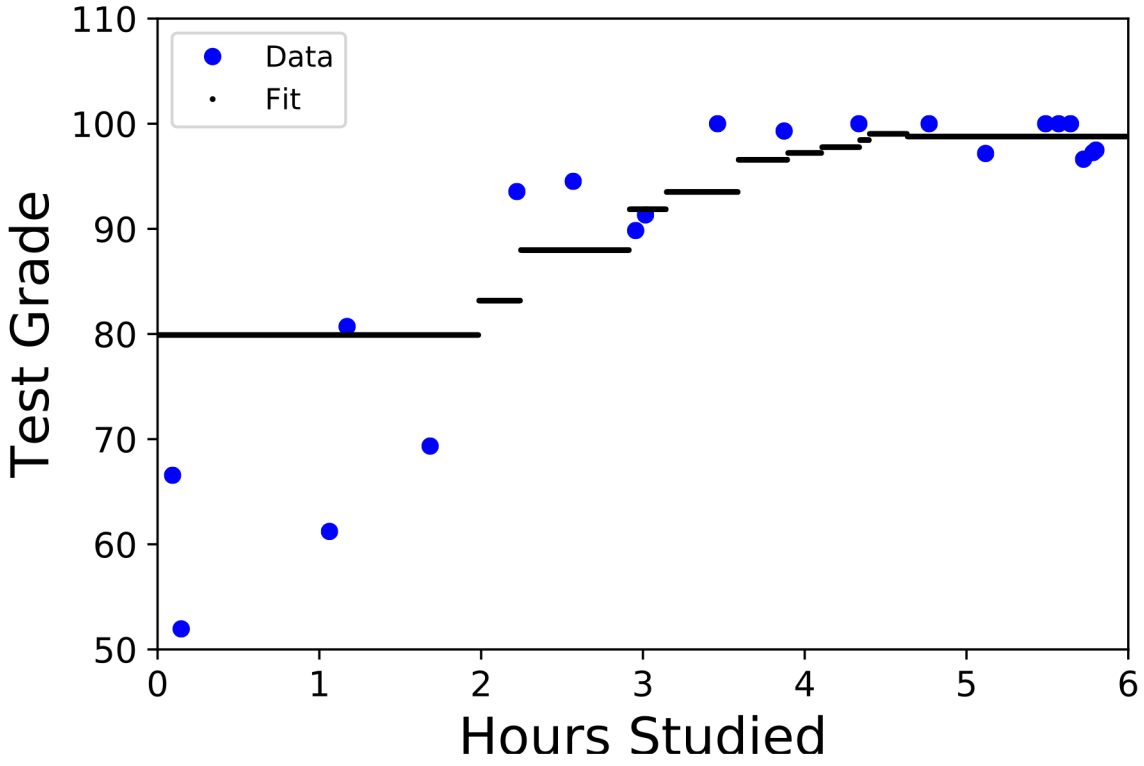
K=5 with 20 data points



K-Nearest Neighbors

K=10 with 20 data points.

This is starting to look a little strange

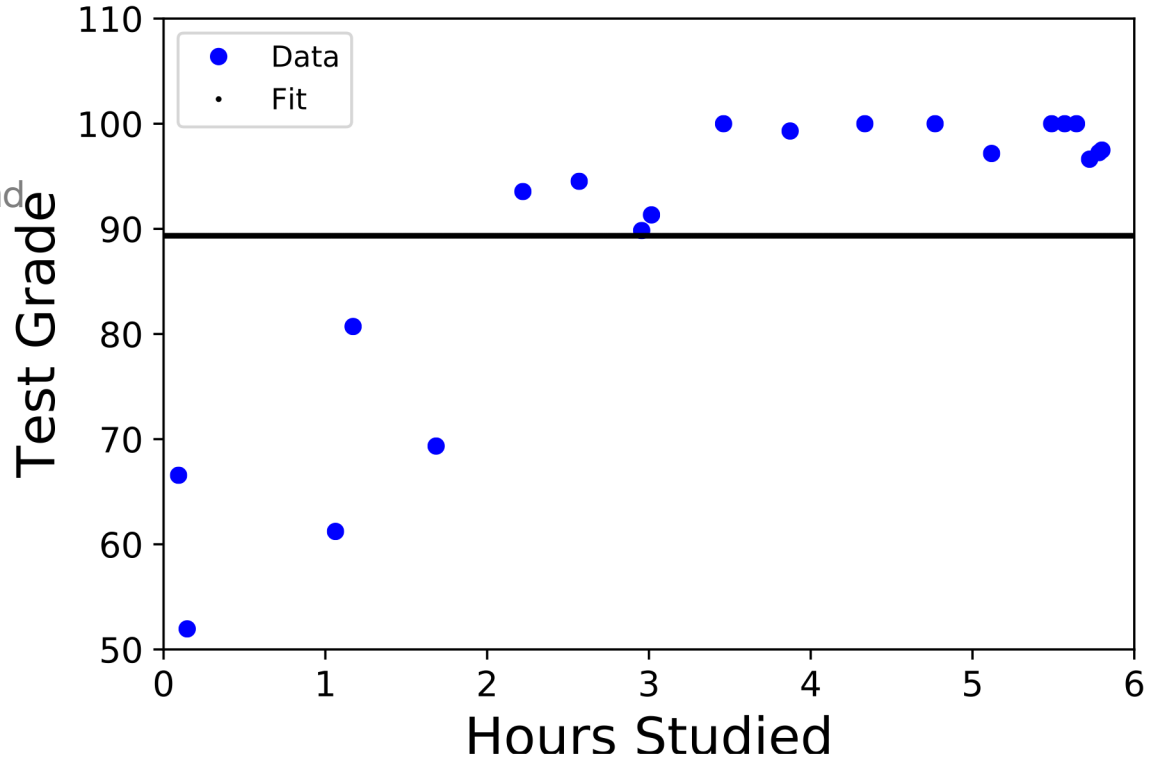


K-Nearest Neighbors

K=20 with 20 data points.

Here we've clearly overdone it and used too many neighbors!

How can we rigorously figure out which approach is best?



What is my fitting error?

- One way to assess model fit is to observe the residual error (root mean squared error) on the observed data points:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

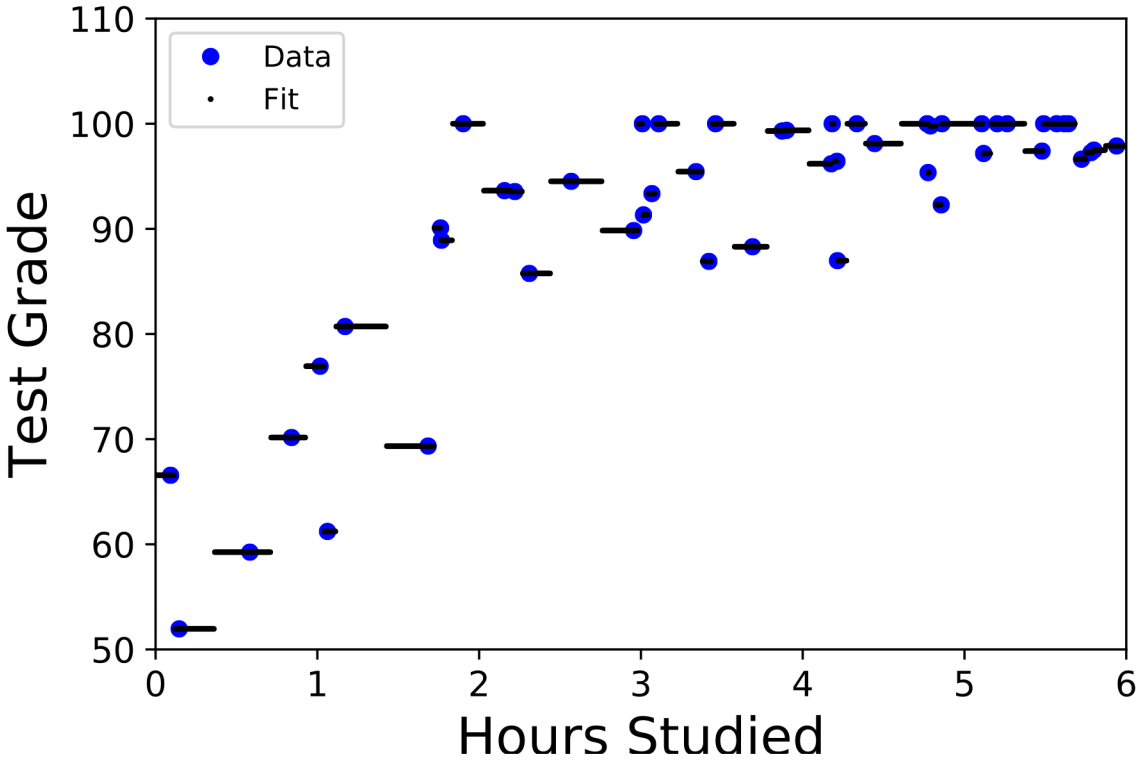
- Looking at this value on observed data is:
  - **Very useful** in traditional statistics (where theory/ asymptotics kick in)
  - **Very bad** in machine learning

K-Nearest Neighbors

K=1 with 50 data points.

We can learn more complex patterns as the data size increases.

The RMSE on the observed data is 0, mean the model *fits perfectly!*

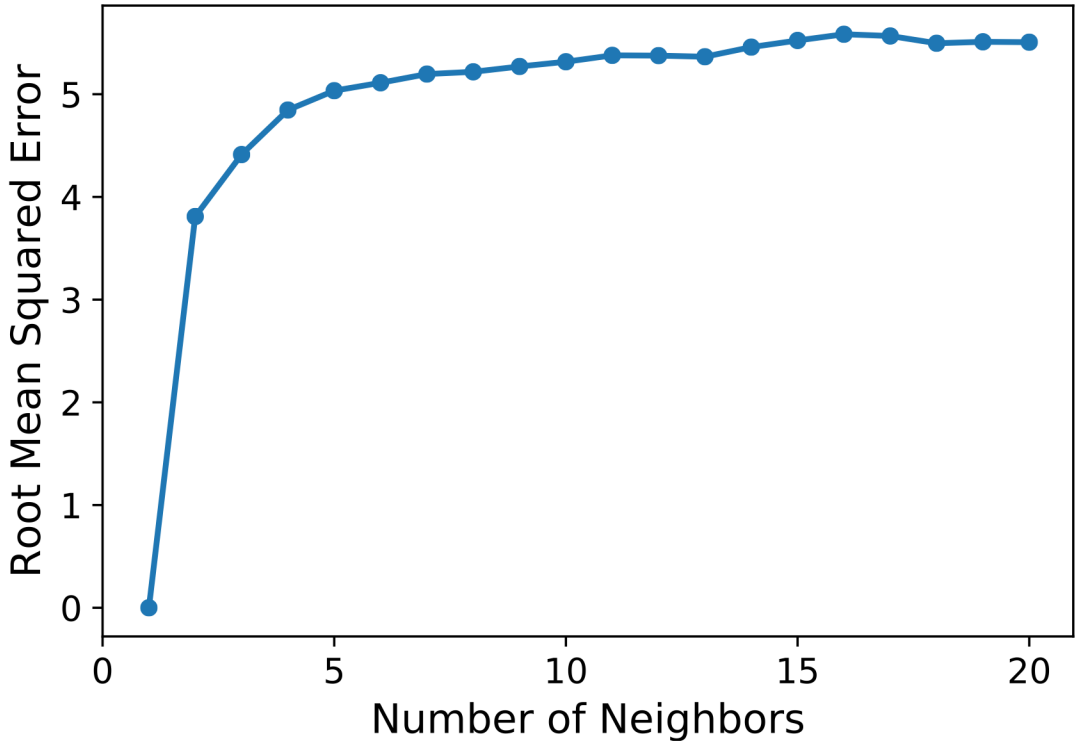


RMSE on the observed data vs # of neighbors

This figure shows the RMSE as a function of the number of neighbors.

1 neighbor is perfect!

Even 2 neighbors starts to add a lot of RMSE…

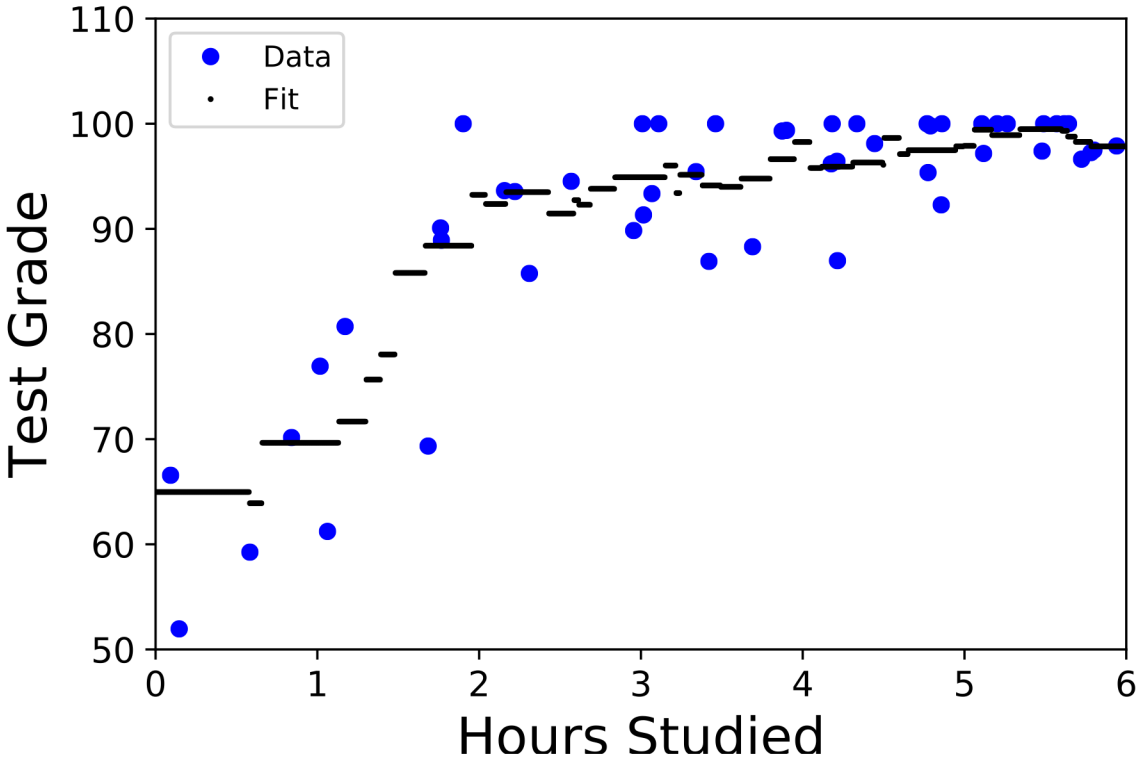


K-Nearest Neighbors

K=5 with 50 data points.

This visually looks *better* than K=1, but the RMSE on the observed data is much worse.

But we really care how this will work on future data.

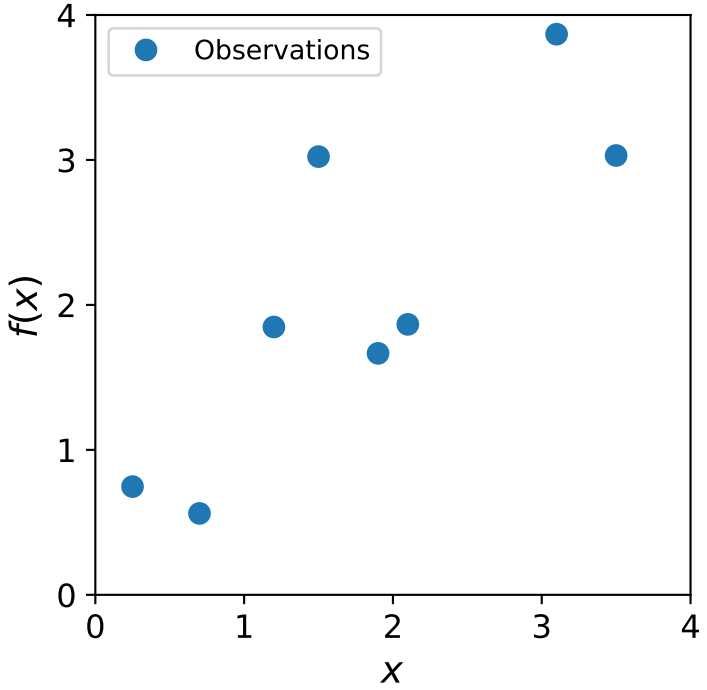


MODEL VALIDATION

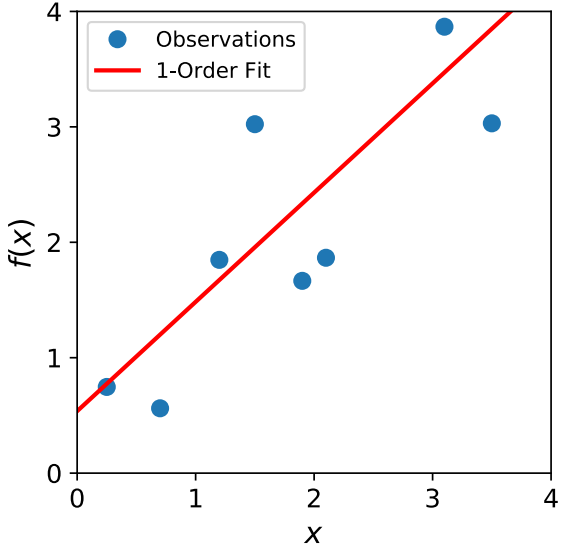
Overfitting

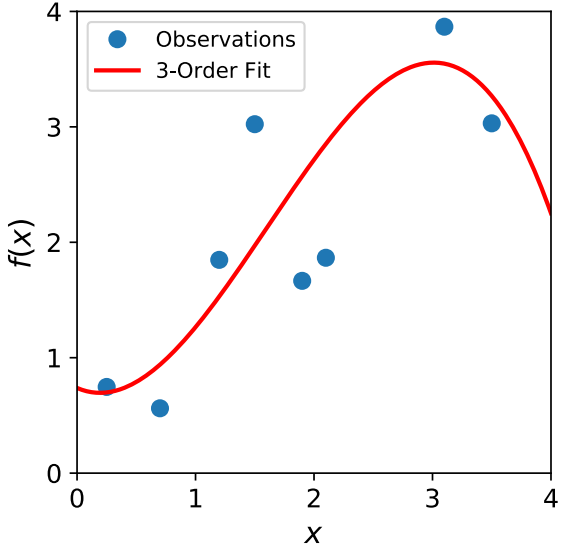
"Overfitting" is when the learned model increases complexity to fit the observed training data *too well* – will not work to predict future data!

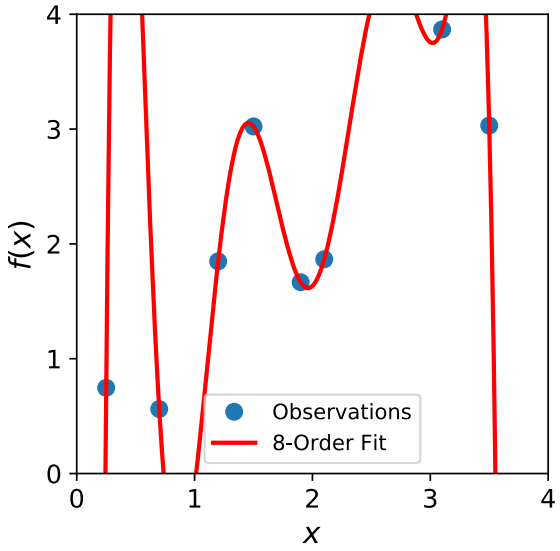
What would we want to use to fit these example data points?



Classic Example: Increasing Polynomial Order





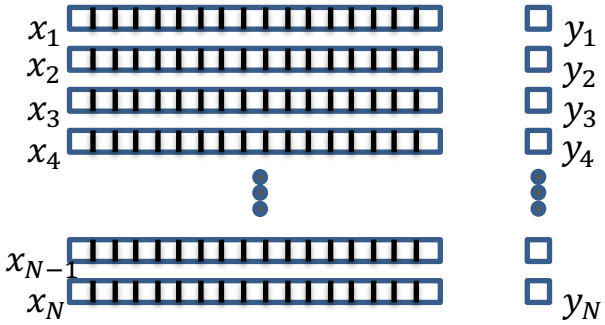


Increasing complexity visually looks ridiculous…

What happens in overfitting?

- We are increasing the number of parameters in the model, which means:
  - More parameters to estimate (all of their errors add up)
  - Can learn *complex* relationships, maybe too complex for reality

- When we *overfit*, this means that we will not *generalize*
  - We want our models and analysis to generalize, or provide accurate predictions on newly collected data



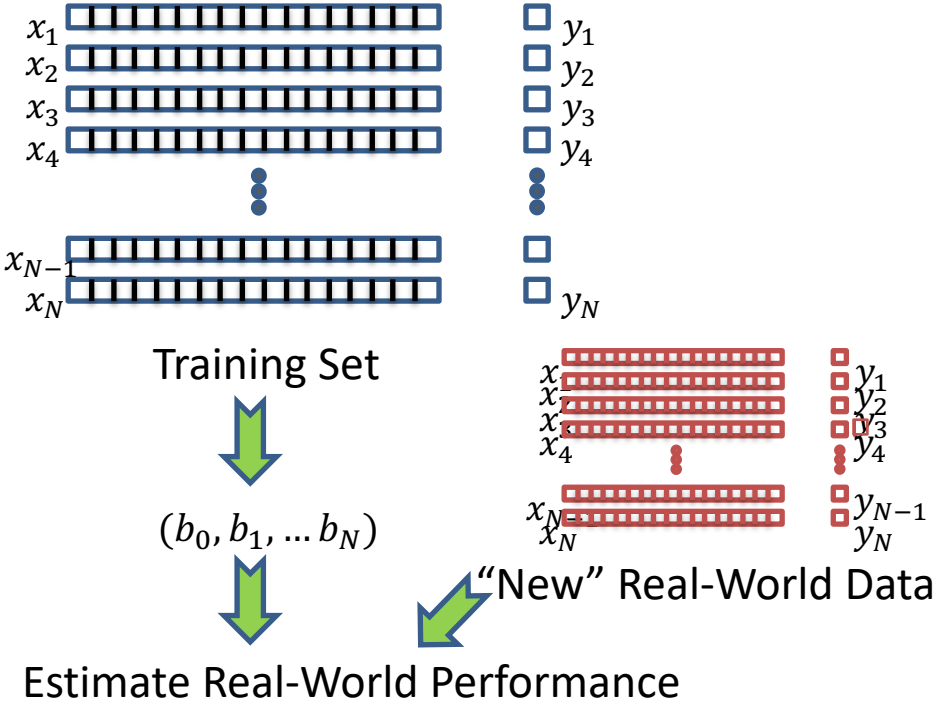
Training Set

$(b_0, b_1, \ldots b_N)$

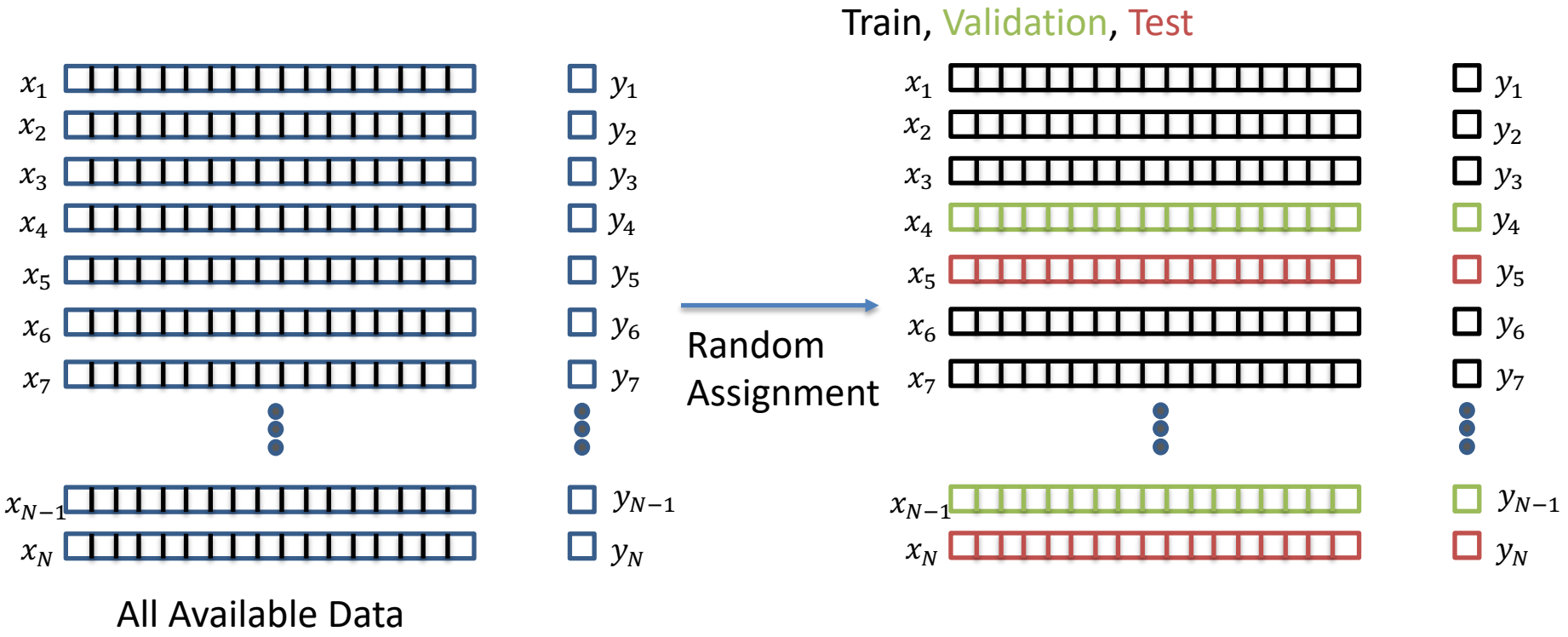
Performance in the Real World??

Standard Validation Strategy

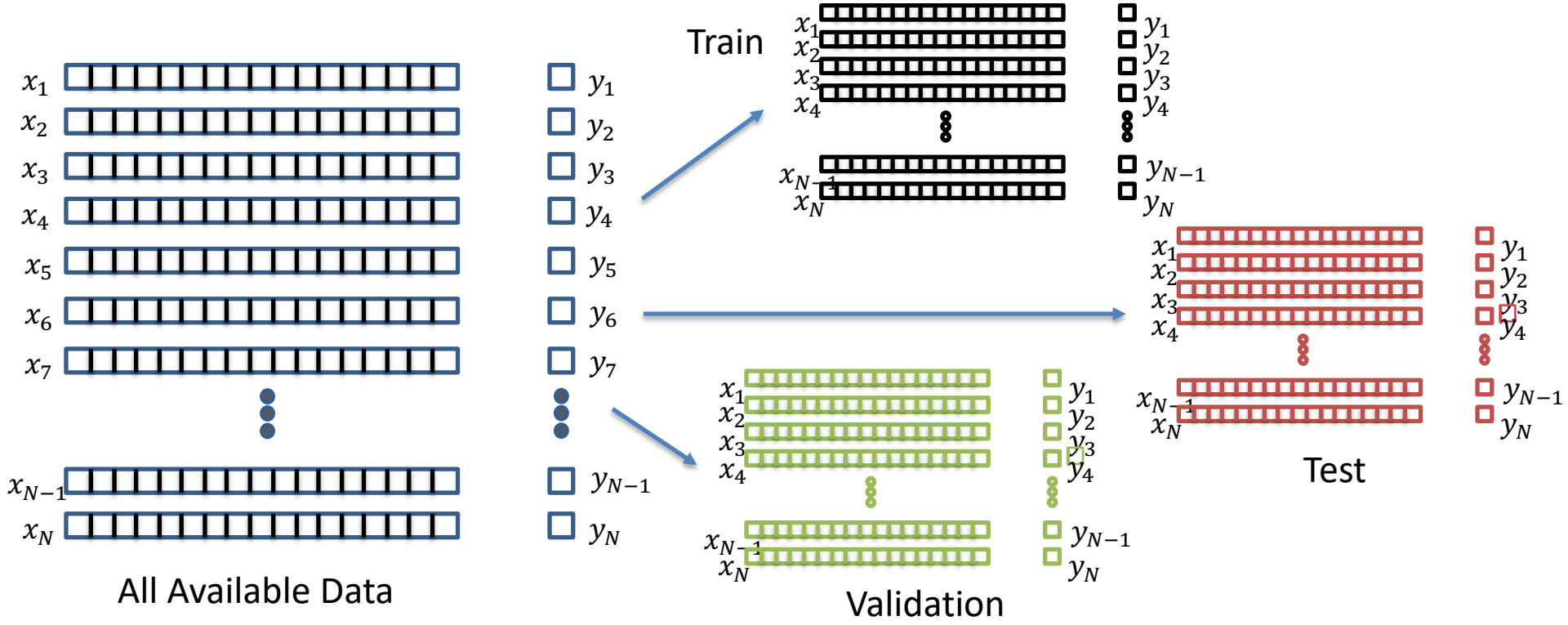
- In the end, we want to know how the network will perform *in the real world*
- Standard approach: actually try it in the real world
- This is costly; instead, can we use existing data to estimate performance?



Split Data into Separate Groups



Split Data into Separate Groups



Training Set

- Used to fit the model/algorithm
- Already discussed at length…

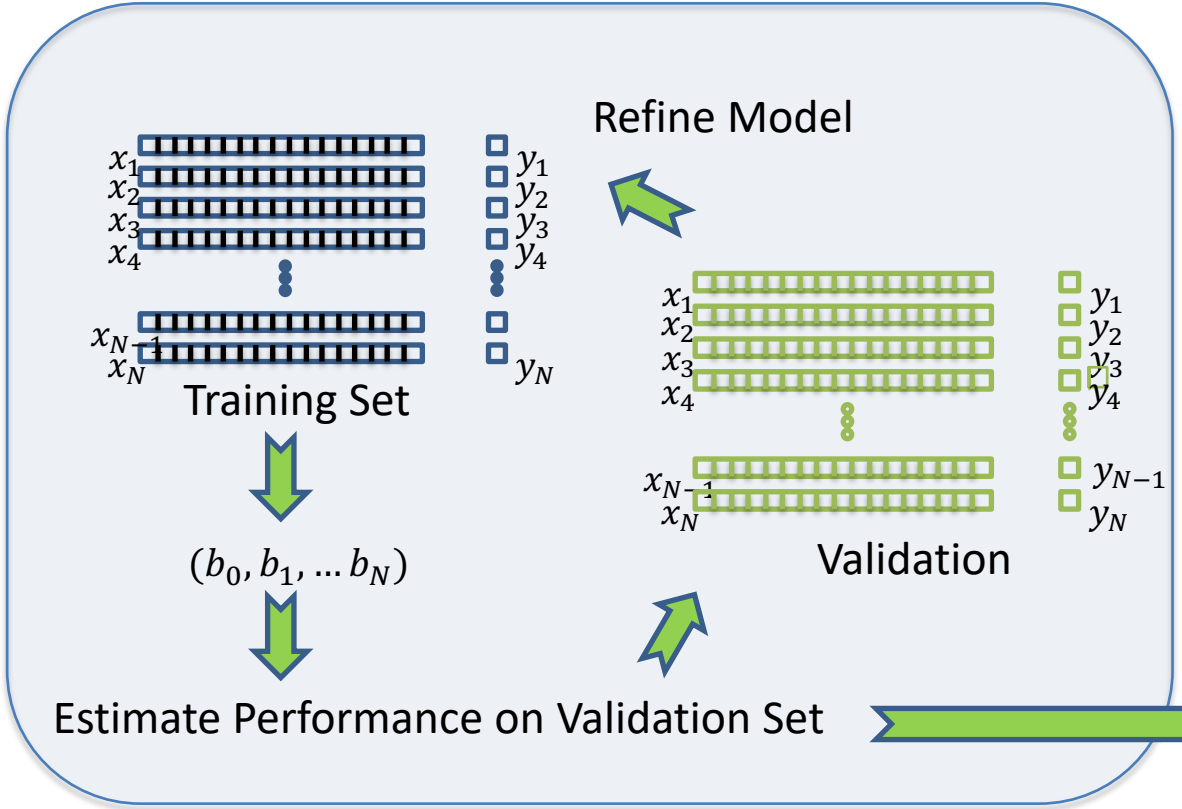
Test set

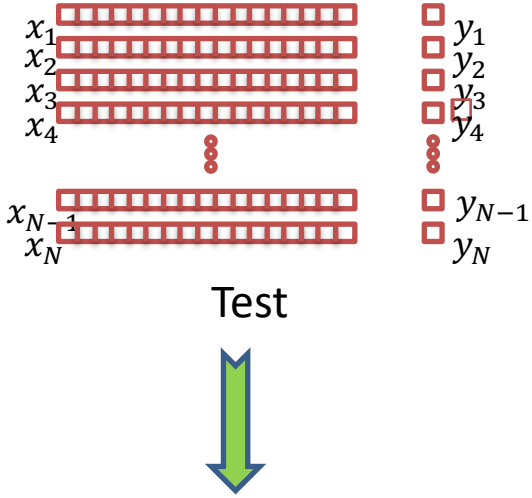
- Standard practice to create prior to any analysis—**will not be used to learn or fit any parameters**
- After learning the network, can evaluate its performance on the test set
  - This data was not included in the training/fitting, so it is analogous to running a new synthetic experiment

- Ideally, the test set will be used *once*.
  - Reusing the test set leads to bias; performance estimates will be optimistic

Validation Set

- Want to be able to compare which approach is best
  - Problematic if we only want to use a test set once
  - Can create a second held-out dataset
- The validation data is not used for learning parameters, but can be used repeatedly to estimate performance of a model
- We can pick the model with the best performance on the validation set, and run a final evaluation on the test data

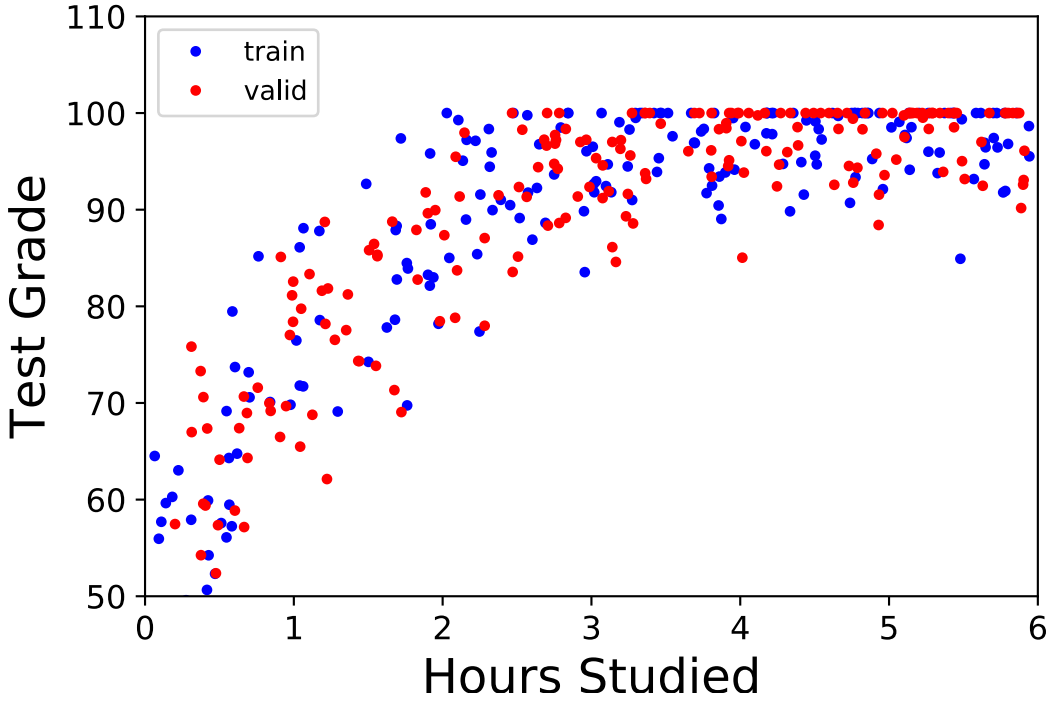




Final Performance Evaluation

VALIDATION STRATEGIES ON NEAREST NEIGHBORS

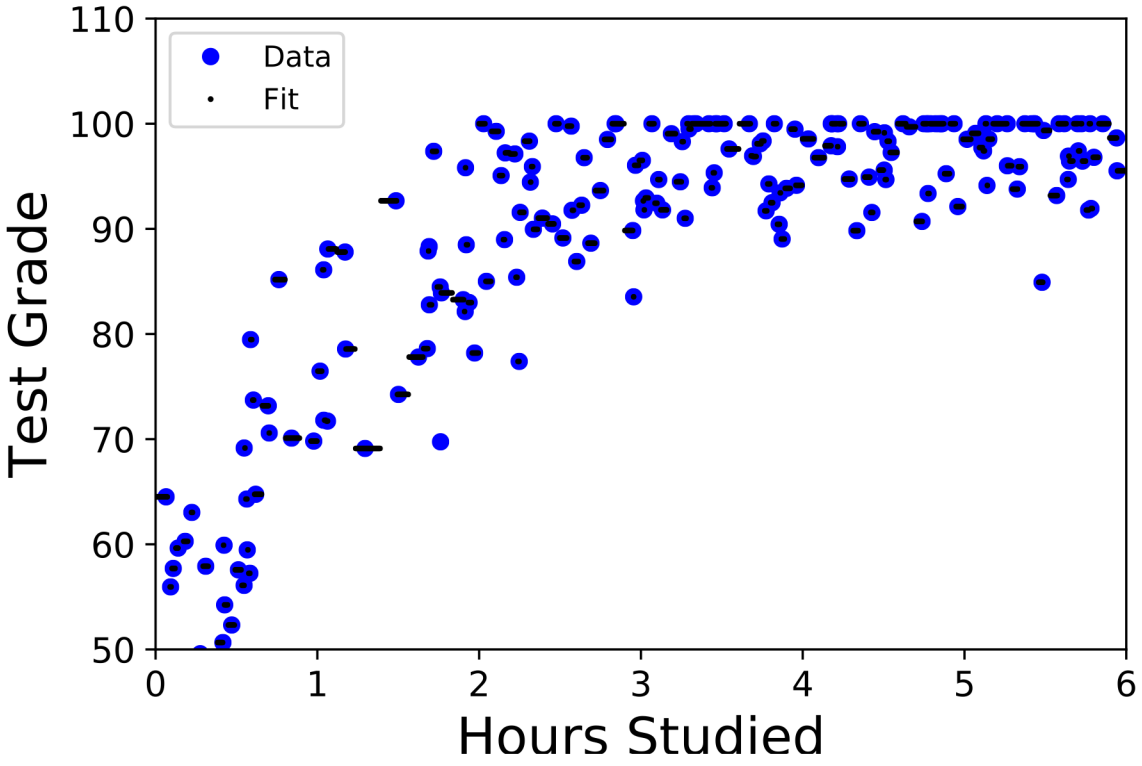
Suppose we have split our data



K-Nearest Neighbors

K=1 with 200 data points.

Training RMSE=0
Validation RMSE~=9

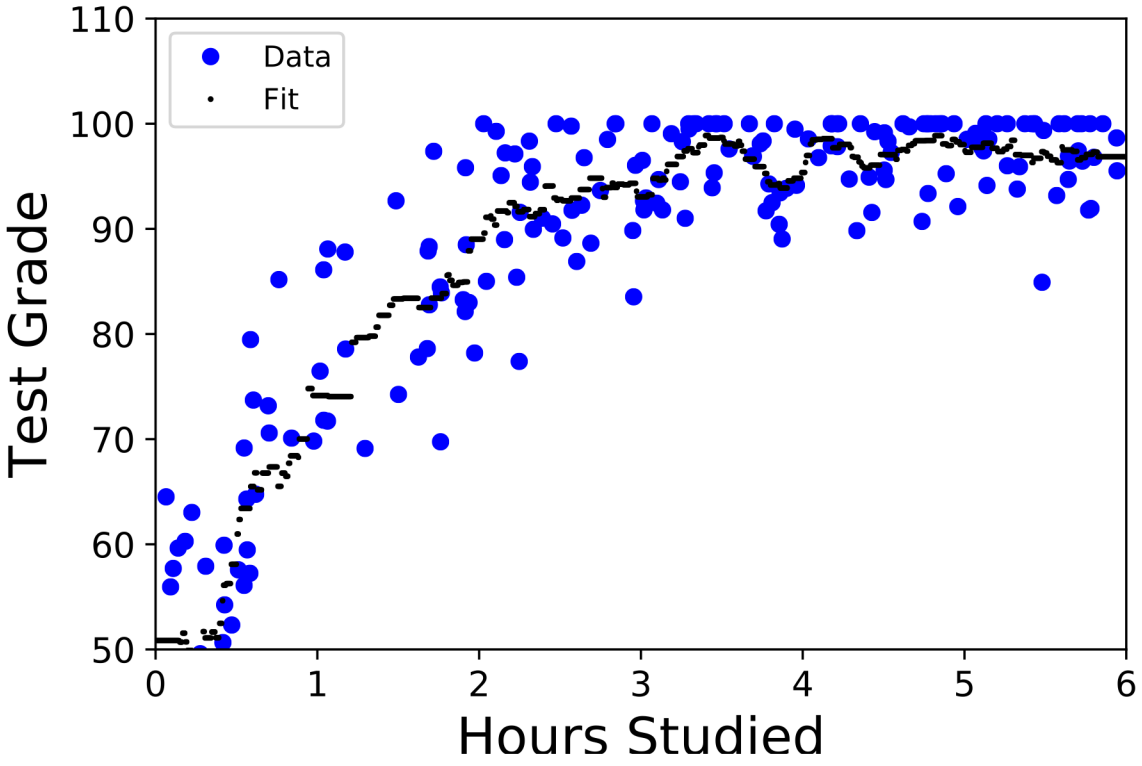


K-Nearest Neighbors

K=10 with 200 data points.

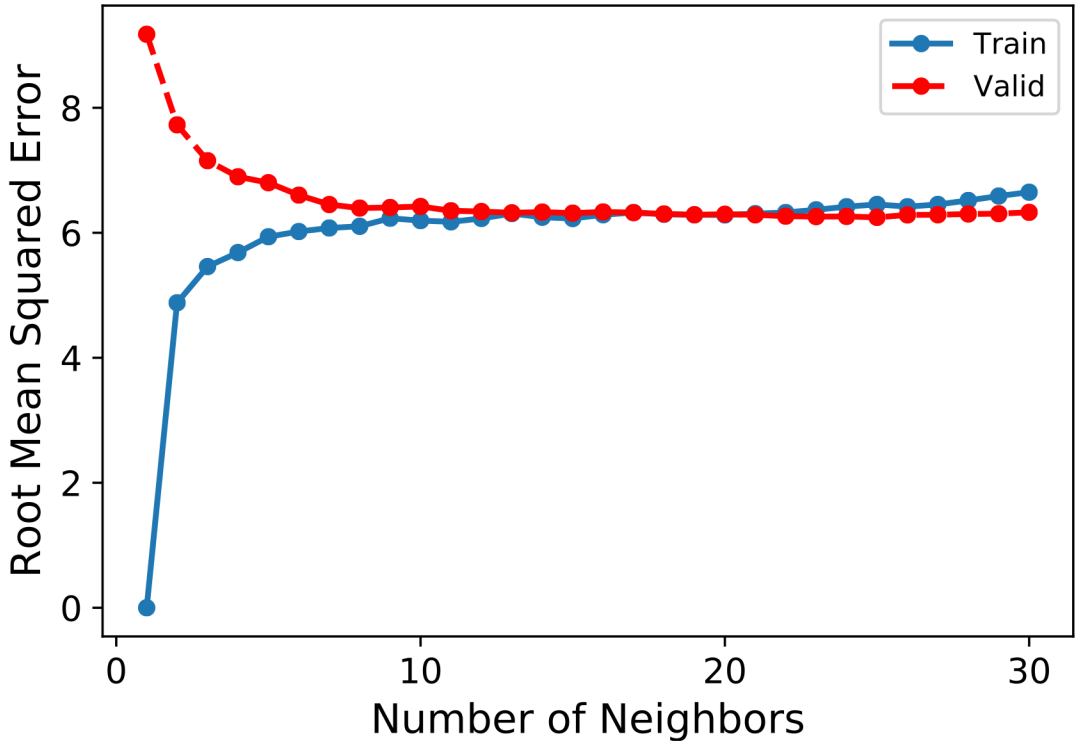
Training RMSE~=6
Validation RMSE~=6

Drastically better on validation RMSE.



How many neighbors?

We can look at the training and validation loss over all of the neighbors.



HIGHER DIMENSIONAL DATA AND CLASSIFICATION

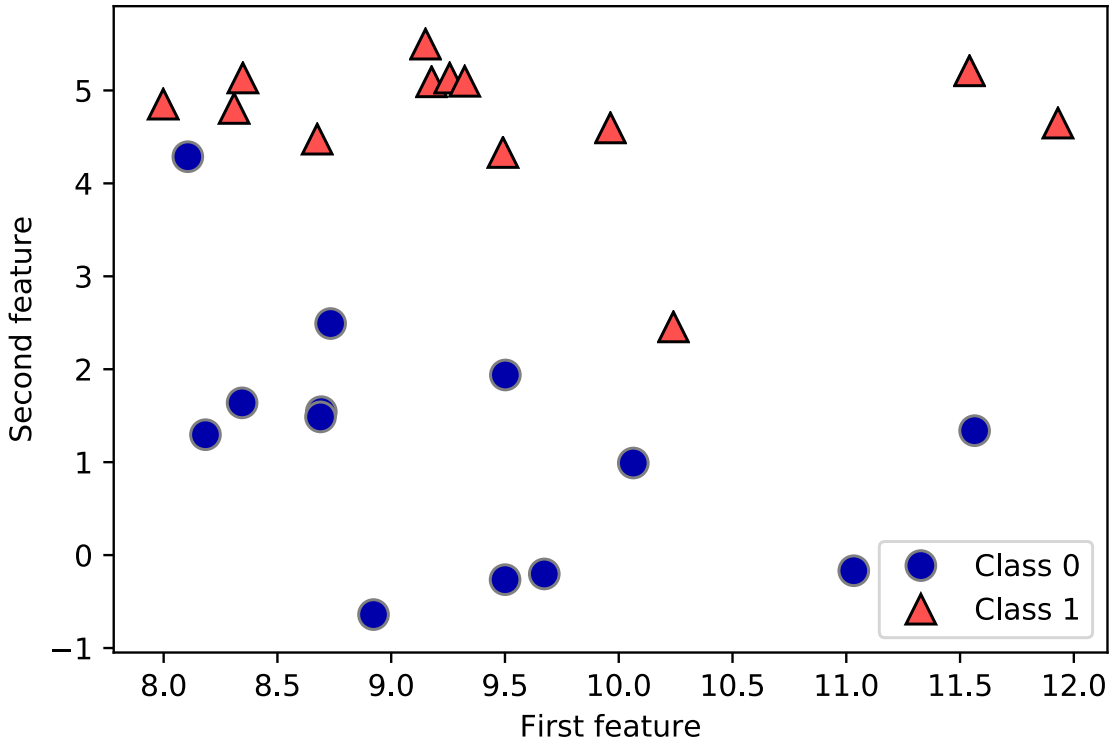
K-NN Classifier

- Suppose we wanted to predict *classes* rather than continuous values
- We can do that by predicting the *majority* class of our neighbors

Some Example Data in Two Dimensions

This is synthetic dataset that's useful for visualization.

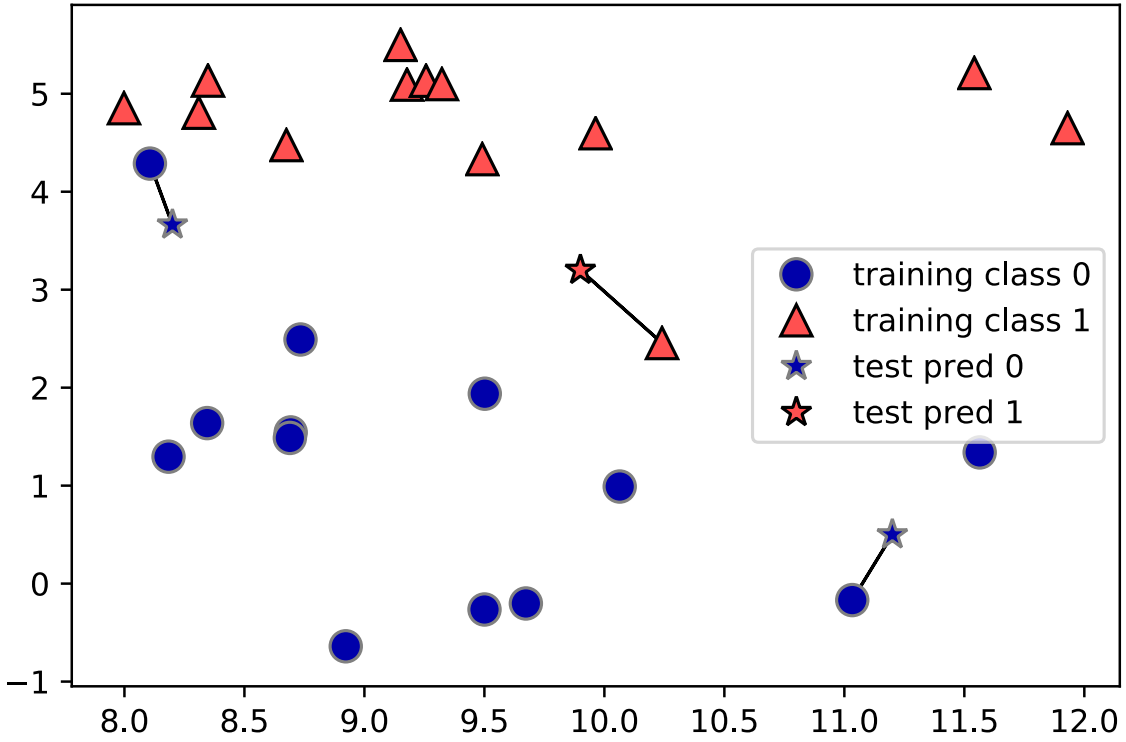
There are two classes (i.e. binary classification) represented in the dataset.



Visualization of Nearest Neighbors

Here are some example datapoints added that need predictions.

The lines represent the nearest data point in the training example, which we use to make predictions.

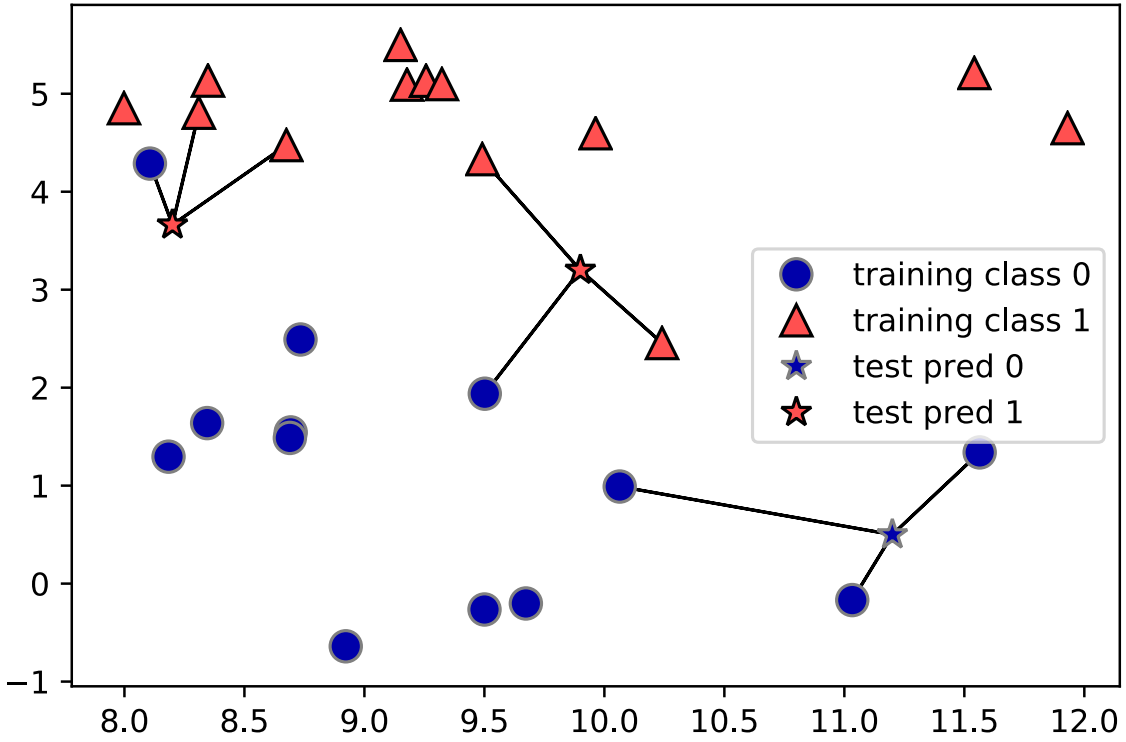


Visualization of Nearest Neighbors

Here are some example datapoints added that need predictions.

The lines represent the nearest data point in the training example, which we use to make predictions. Here there are **3** neighbors.

When we have multiple neighbors, we are going to pick the majority class.

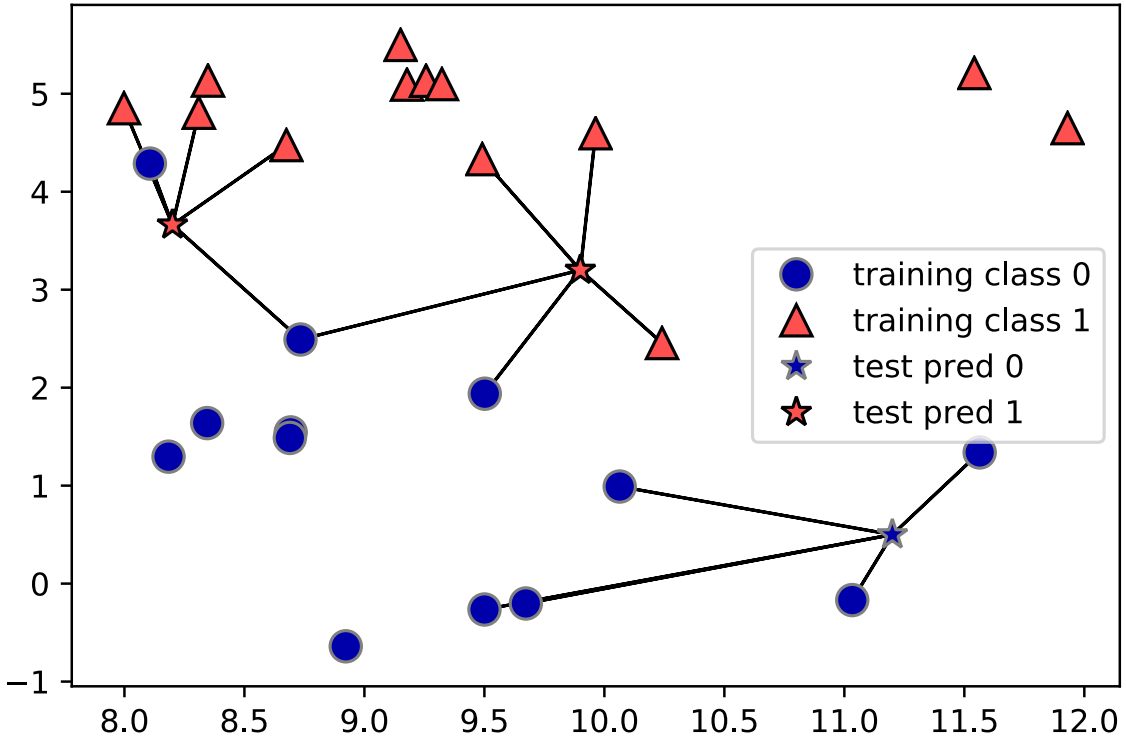


Visualization of Nearest Neighbors

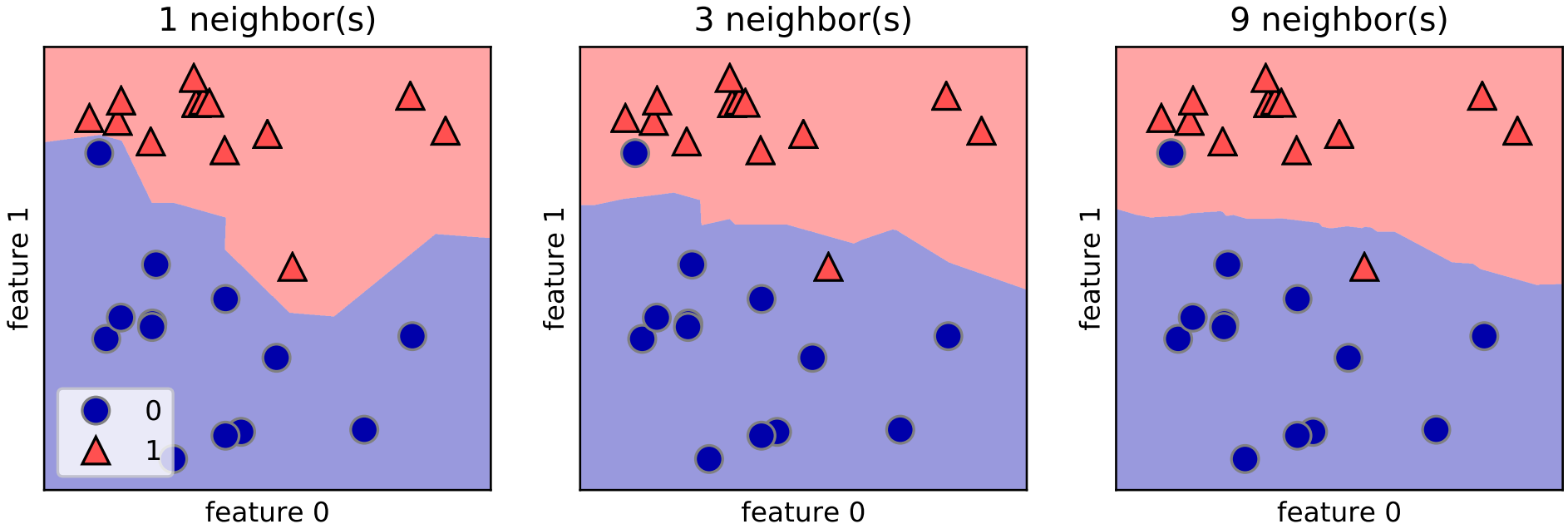
Here are some example datapoints added that need predictions.

The lines represent the nearest data point in the training example, which we use to make predictions. Here there are **5** neighbors.

When we have multiple neighbors, we are going to pick the majority class.



Visualization of Decision Boundaries



Some Comments:

- We are actually going to predict *probabilities* as in logistic regression. Estimated probability for a new data point x^* is

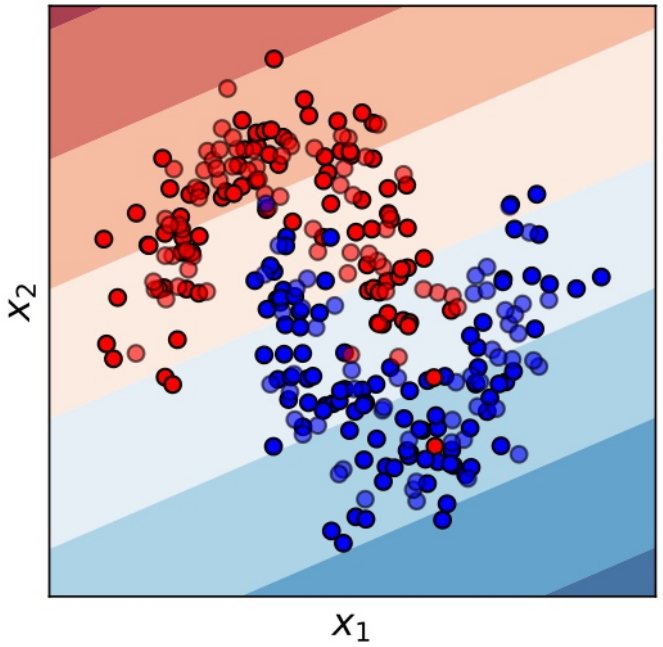
$$p(y^* = 1|x^*) = \frac{1}{\#\ of\ neighbors} \sum_{i \in neighbors} 1_{y_i=1}$$

- We also have to choose how to determine our neighbors. Typically we will use the ”Euclidean distance” or “squared distance” or “l2” norm. Suppose that x has P features:

$$d(x_a, x_b) = \|x_a - x_b\|_2 = \sqrt{\sum_{p=1}^{P} (x_{ap} - x_{bp})^2}$$

Limitations of Linear Classifiers

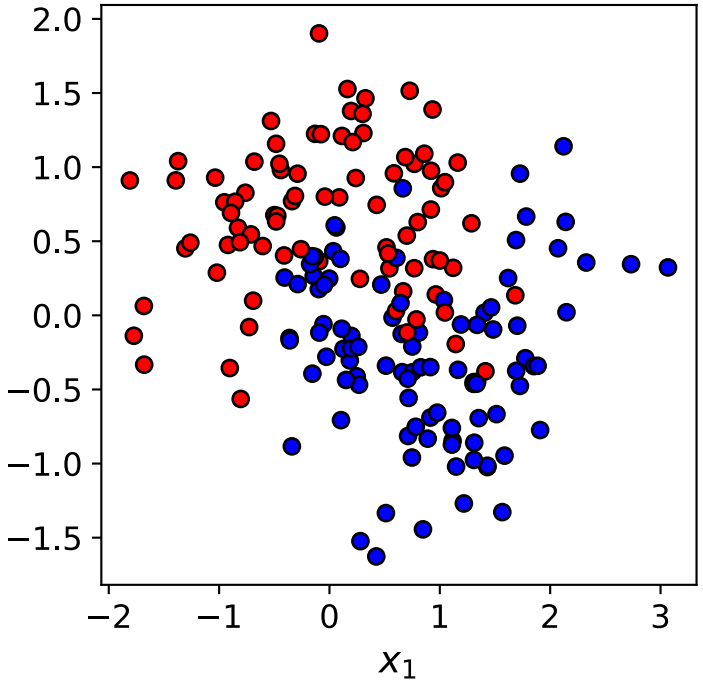
- Linear classifiers can only represent limited relationships
- Often want to use a classifier that can handle non-linearities.
- Many different ways of creating non-linear classifiers!



Let's look at applying kNN to this dataset

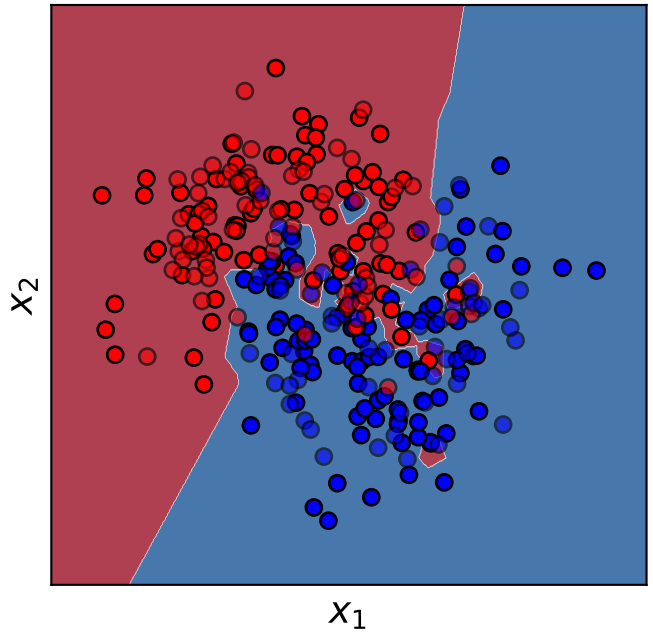
Here we have a highly nonlinear dataset.

This is an even noisier version of the previous dataset.



Let's look at applying kNN to this dataset

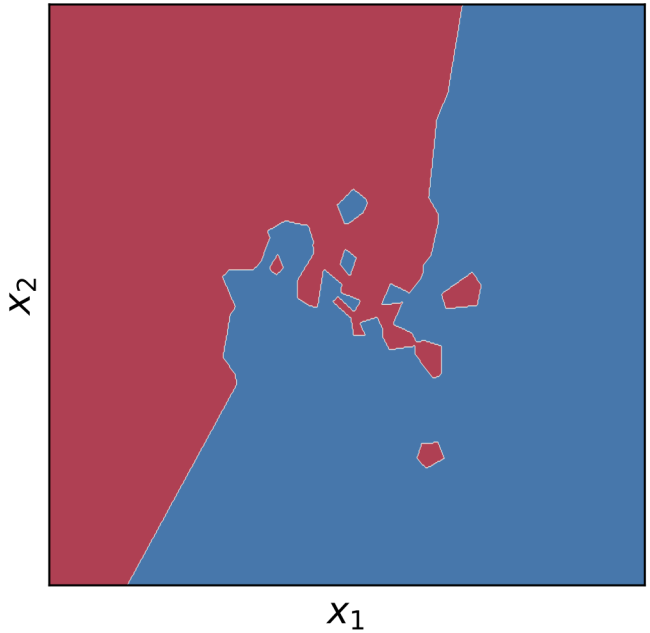
This is the result when k=1.



Let's look at applying kNN to this dataset

This is the result when k=1.

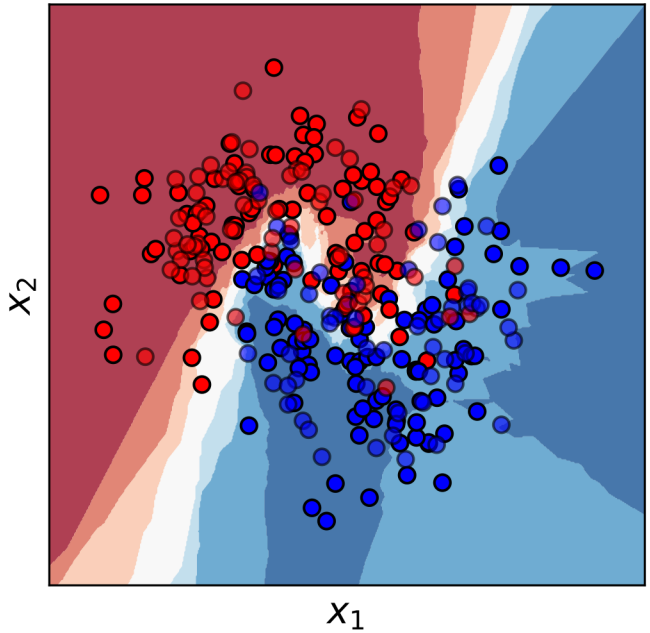
Can be helpful to look at the decision surface without the observed points.



Let's look at applying kNN to this dataset

This is the result when k=10.

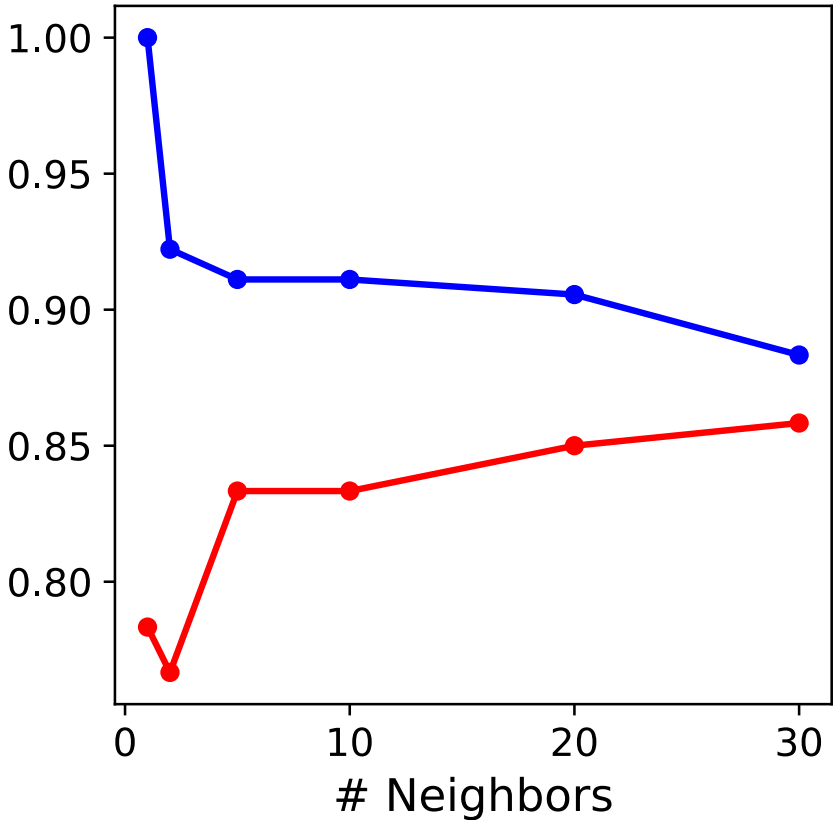
It has learned much smoother transition boundaries, probably much more realistic.



How many neighbors should we use?

We can look at the accuracy of the classifier as a function of the number of neighbors.

How many neighbors should we use?



Can we reduce the variance of our estimates?

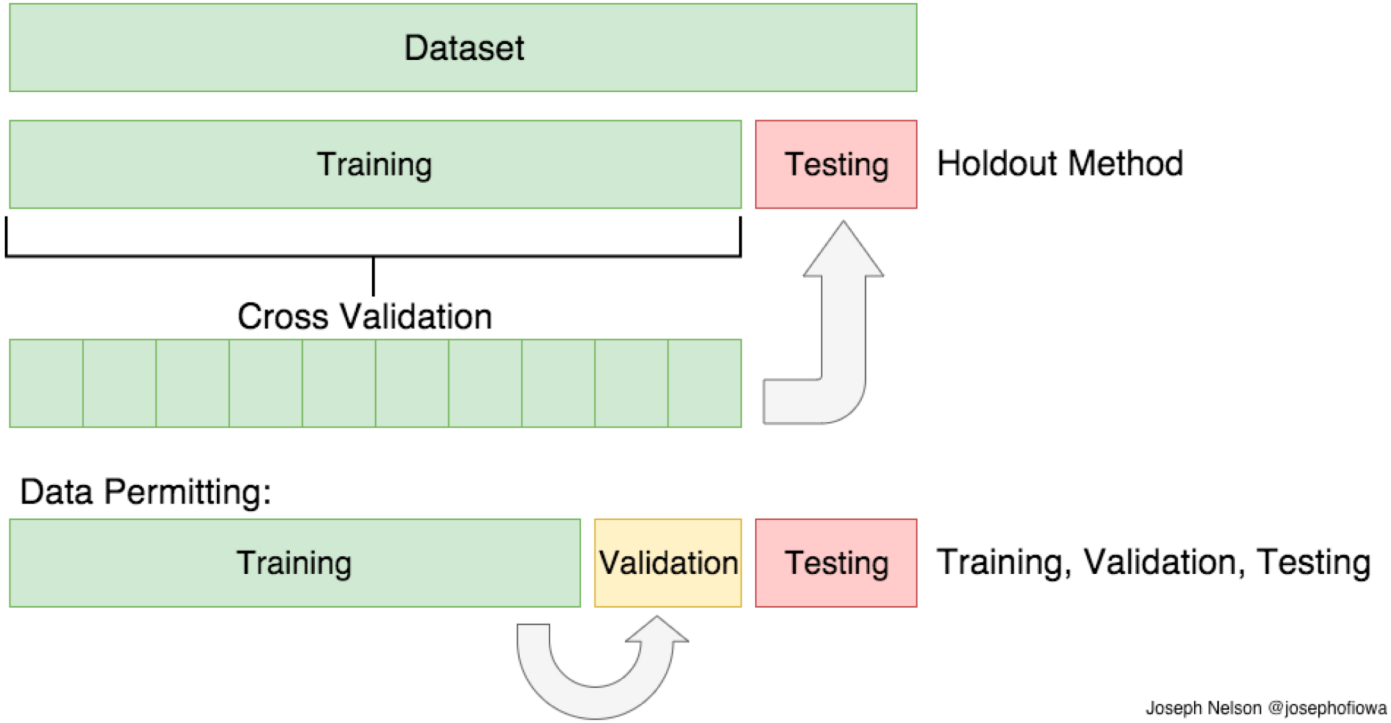
CROSS-VALIDATION

Cross-Validation

- Creating a validation set is a useful way to estimate performance prior to using the test set, and is useful for model selection (which will be cover in a few lectures).
- However, using one validation set can skew performance and its performance estimates are noisy.

One solution is to create *multiple* validation sets and keep retrain and test on multiple splits of the training data.

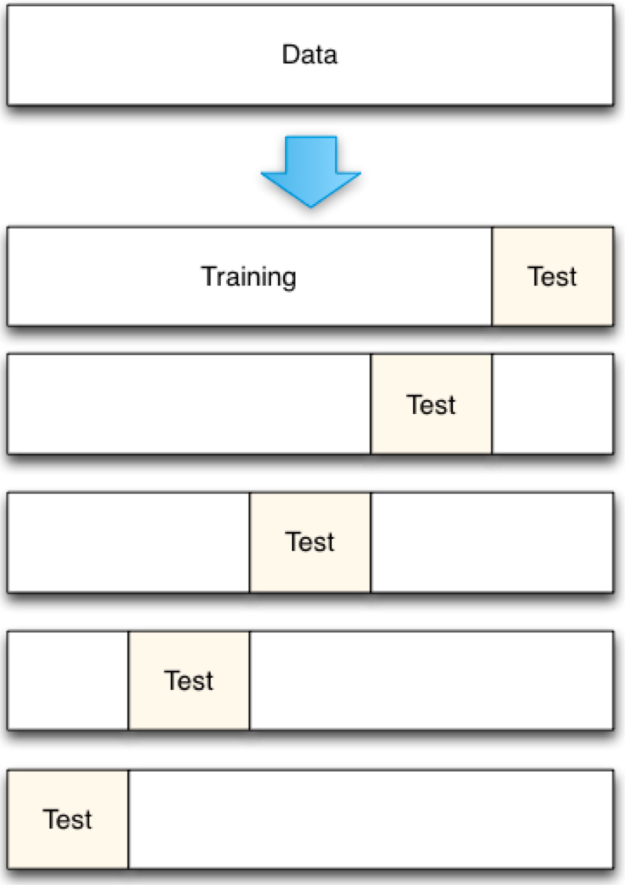
Visualization of data splitting



Cross-Validation

Visualization of 5-fold cross-validation.

Often in cross-validation, the individual splits are called the “test” set. This is weird nomenclature, I will always call this the ”validation” set.



Some comments:

- Cross-validation can reduce variance of performance estimates
- We will typically use cross-validation in this class
- Can be *very helpful* in statistical testing (will cover in unit 4).

PERFORMANCE MEASURES

What is the question that we're trying to ask?

- Effective analysis is dependent on asking the right question. What is the goal?
- Some common goals:
  - Make the best value prediction or forecasting (regression)
  - Make an effective classification (i.e. medical test, want to determine what is wrong with the patient to determine the correct treatment)
  - Grouping similar data points

Forecasting

- Typically forecasting into the future means that we want to minimize distance (i.e. L2 error).
- Implicitly this is assuming that our data follows a normal distribution from the prediction.

Classification

- We want to split data into pre-specified groups, i.e. does this patient have cancer?

One standard way to estimate performance is is use accuracy:

$$Accuracy = \frac{\#\ of\ data\ points'outcomes\ guessed\ correctly}{total\ \#\ of\ data\ points}$$

Accuracy can be very misleading…

How good does a medical test have to be?

- Suppose a patient comes into the clinic, and the doctor prescribes a test to check for lung cancer.
- Suppose that we have test that is:
  - 99% sensitive (99% of true cases are predicted as positive)
  - 99% specific (99% of negative cases are predicted as negative)
- This sounds like a great and accurate test (99% accuracy)!
- Yearly rate of new cases of lung cancer is about ~=.0007 (American Cancer Society).
- Instead, suppose we want to make a test that's even better, with 99.9% accuracy. How can we do this?

We can control sensitivity and specificity

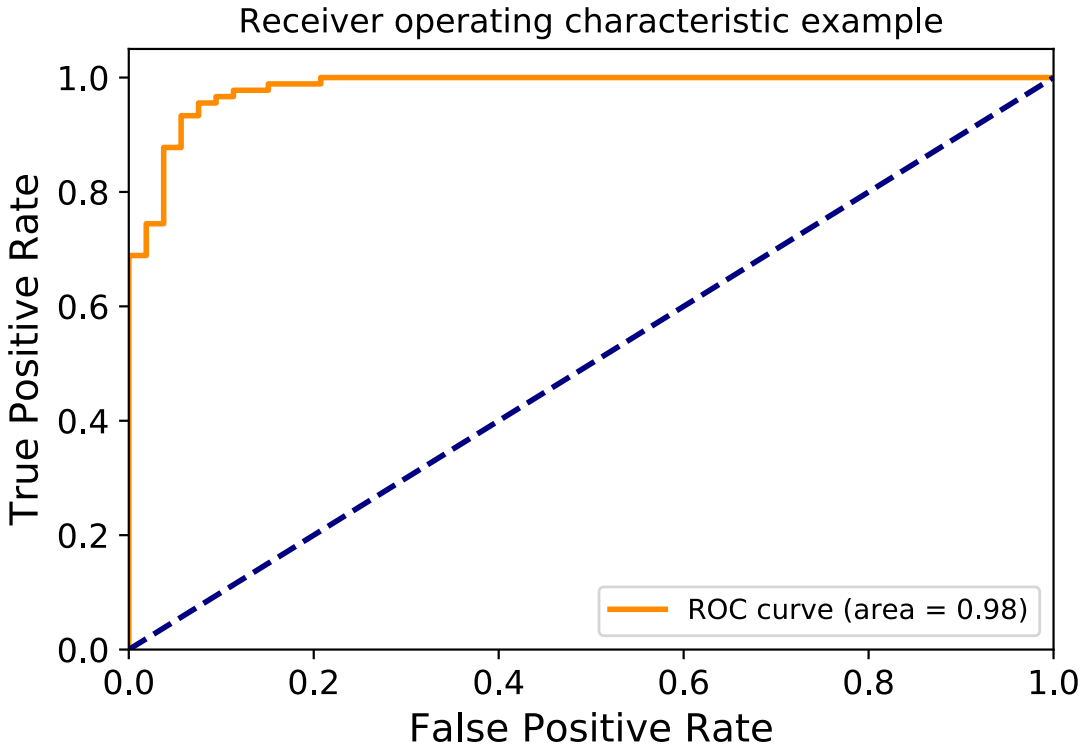
- For most classification algorithms (i.e. predicting a binary outcome), one gets the *confidence (or score)* of the prediction.
- We can change the threshold for the test to change its statistical properties:
  - We can increase true positives, but with increased false positives
  - We can decrease false positives, but with decreased true positives
  - We can use a different test with better properties…

Accuracy is *dependent* on the chosen level of sensitivity.

We can evaluate this tradeoff with something called a “Receiver Operating Characteristic” curve.

Receiver Operating Characteristic

The name comes from radar systems and detection theory, where they have similar statistical goals but very different methods.



Area Under the Curve (AUC)

- Typically “Area under the ROC Curve,” which is the integral of the curve on the ROC plot, which varies from 0 to 1. Sometimes abbreviated as AUROC. Also known as a c-statistic (concordance) in biostatistics.
- Some key properties:
  - 1 is perfectly separated (i.e. perfect prediction)
  - 0.5 is random chance.
  - 0 is perfectly anti-separated (should never happen! Something is wrong if you get anything near zero…)
  - Practically, this should be from 0.5-1, and higher is better.
  - Robust to the threshold for accuracy
  - Robust to imbalanced classes
- There are alternative ways of viewing what the AUC metric is.

AUC as a ranking metric

- *My* preferred viewpoint is to consider AUC as a rank. Consider the following protocol:
  - Draw a random sample from the positive class, and get a score (or confidence) X_1 from the classifier (or scoring algorithm)
  - Draw a random sample from the negative class, and get a score (or confidence) X_0 from the classifier (or scoring algorithm)
  - There is an equivalency here:

$$Prob(X_1 > X_0) = AUC$$

Conclusions

- K-Nearest Neighbors can be an effective classifier to capture non-linear relationships
- Can ”learn” more complex patterns as we get additional data
- Predictive performance needs to be carefully evaluated.