# Machine Learning

PHYS 453

Dr. Daugherity

# Intro to ML Theory

- Basic vocab and techniques
- Set up so we can use the tools properly
  - it is very easy to skip vital steps and end up with nonsense
  - will see how this setup looks in python with scikit-learn next
- For today we don't care how the models work.  We will start caring very soon.

Theory Introduction

# SUPERVISED LEARNING

# High-Level Overview

**Input**: represent one sample as numbers (called **features**) in an efficient way that is hopefully useful for solving the problem

**Output**: answer (the **target**) depends on our question:

- Classification: what discrete category does the sample belong to?
- Regression: predict a real number
- Reinforcement: optimize a certain outcome

**Model**: how we calculate the output from the input. Usually has lots of adjustable parameters that need to be set somehow

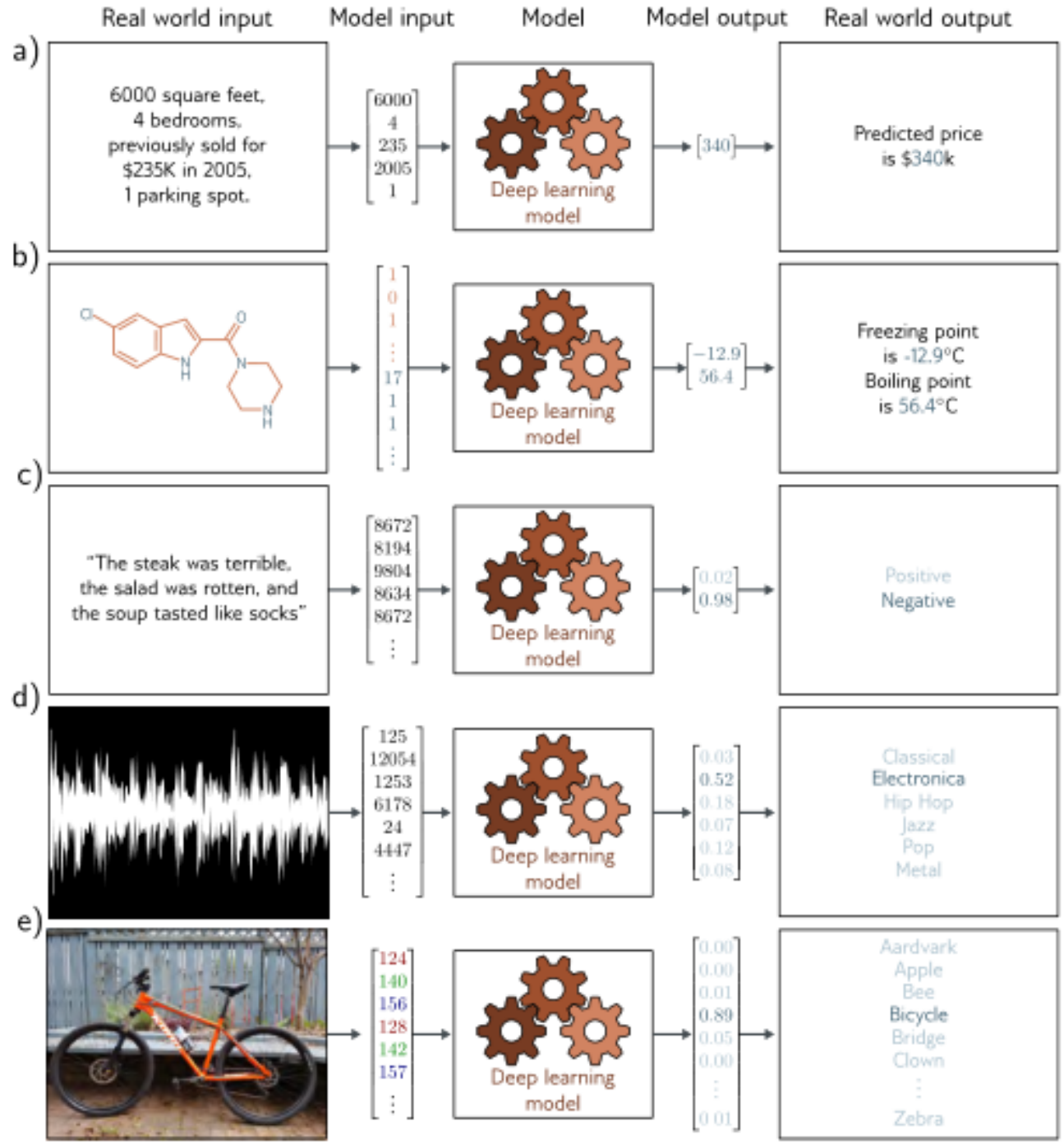



**Figure 1.2** Regression and classification problems. a) This *regression* model takes a vector of numbers that characterize a property and predicts its price. b) This *multivariate regression* model takes the structure of a chemical molecule and predicts its melting and boiling points. c) This *binary classification* model takes a restaurant review and classifies it as either positive or negative. d) This *multiclass classification* problem assigns a snippet of audio to one of $N$ genres. e) A second multiclass classification problem in which the model classifies an image according to which of $N$ possible objects it might contain.

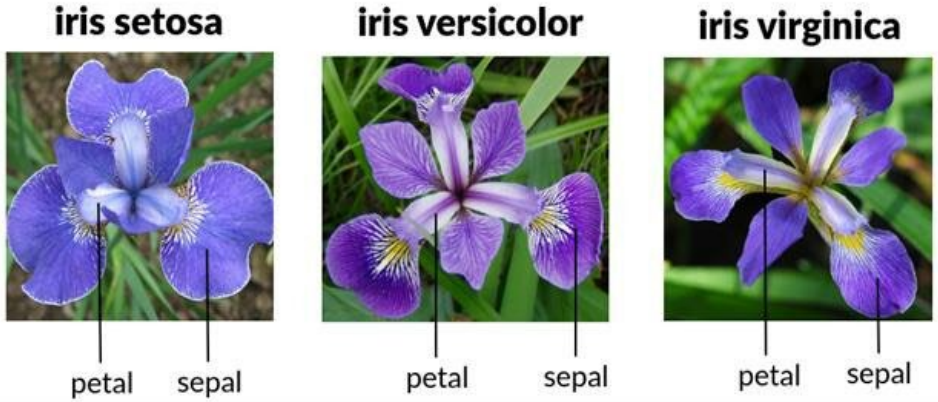
# Supervised Learning

- **Supervised Learning**: make decisions based on known examples
- **Features**: the measured data on a sample
  - stored in **2D array X**
  - think of a spreadsheet with columns of feature data, each row is a sample
- **Target**: the right answer
  - stored in **1D array y**
  - our Training Data: many samples where we already know X and y
- **Predicting**: goal is to predict y for unknown samples given only X

# Data Example

Iris dataset:

- 150 flowers
- 4 features based on length and width measurements
- target is species with 3 options




| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 |
| **...** | ... | ... | ... | ... | ... |
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | 2 |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | 2 |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | 2 |

**Features** **Target**

```
X.shape

(150, 4)

X[:5] # the first 5 flowers

array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

```
y.shape

(150,)

y

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

# Big Picture

*Typical steps in ML project:*

- **Getting Data** – in the real world cleaning the data can be really difficult
- **Preprocessing** – getting data ready to use, only part we've discussed so far is test/train split
- **Training** – choosing models and parameters, "fitting" them to data
- **Evaluating** – reporting how well it works
- **Predicting** – now that you've got a working thing, go out and use it!

# A Relevant Example

Sorting incoming fish according to species using optical sensing

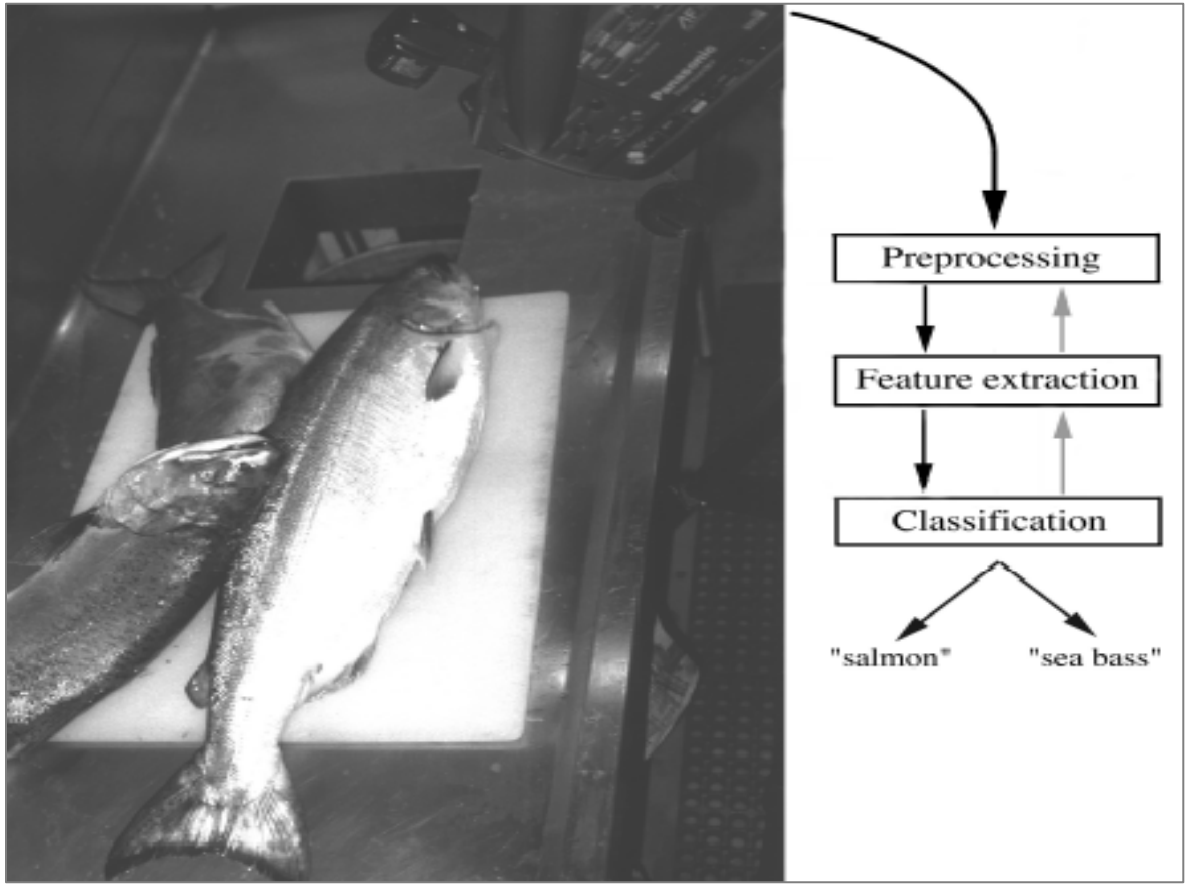
Salmon

Sea Bass

# Features

Set up a camera and take some sample images to extract **features:**

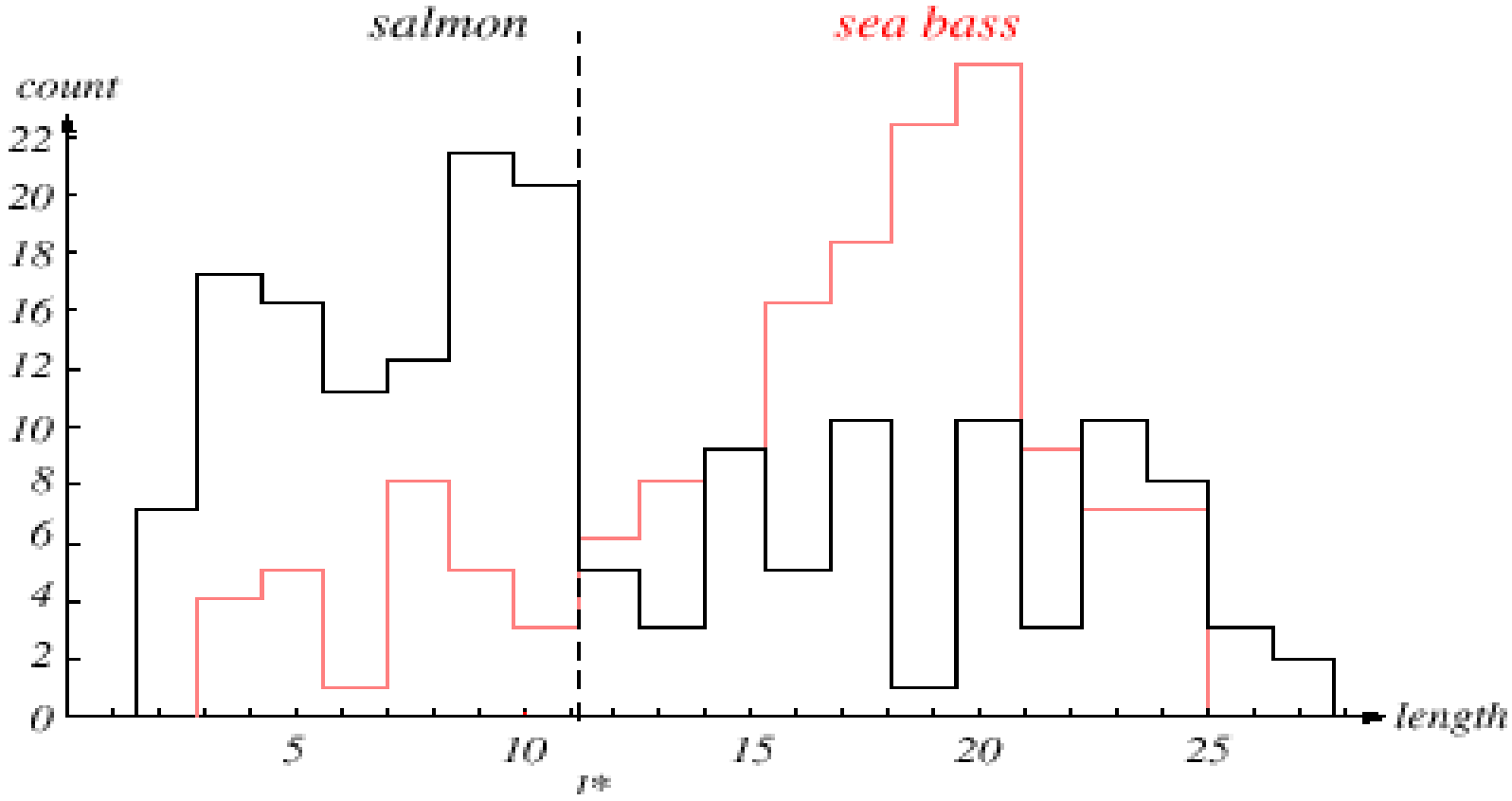
- Length
- Lightness
- Width
- Fins
- Position of the mouth
- etc…




No guarantee ahead of time what will work!

# Classification

How well does length do?



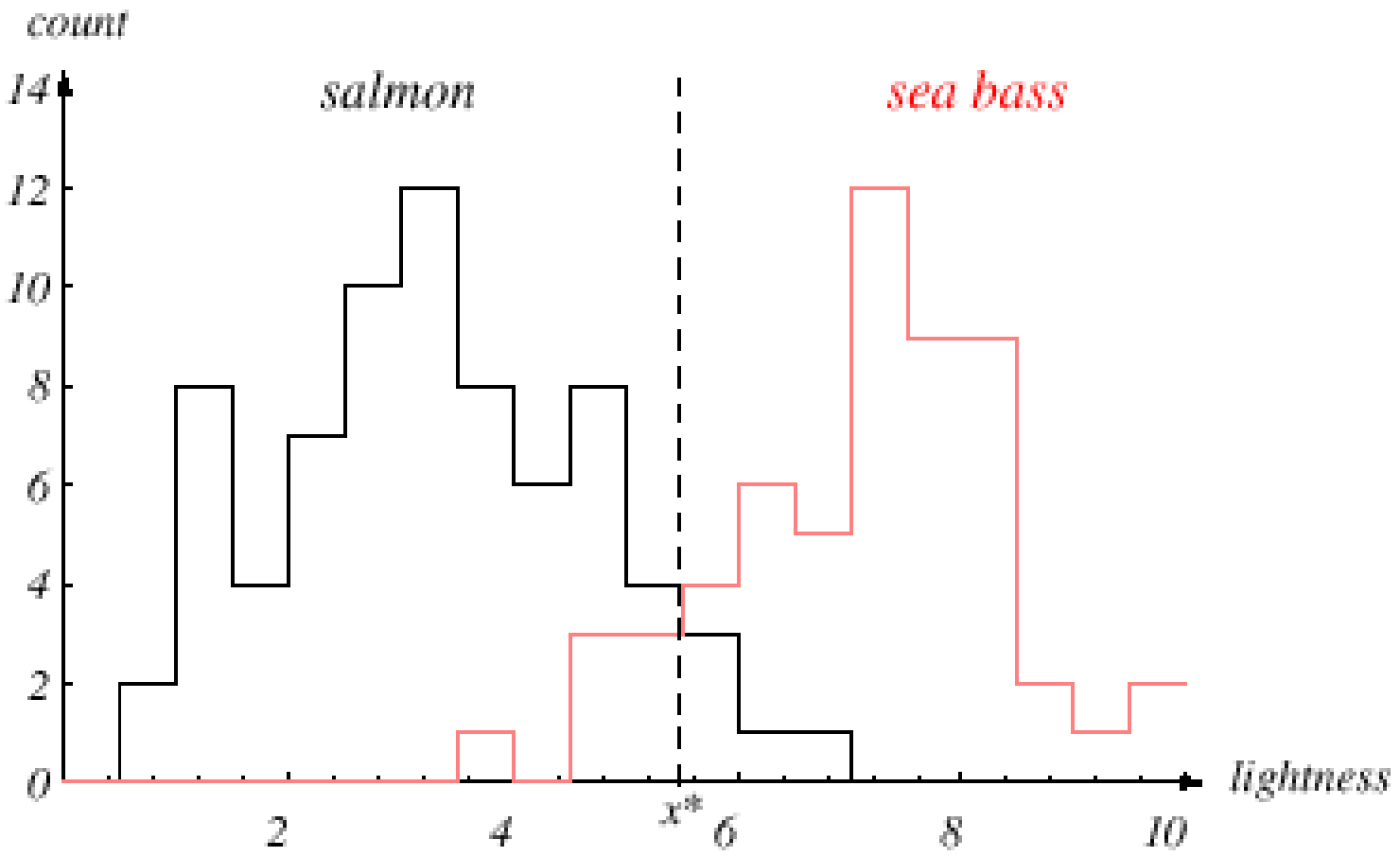

Decision Boundary – optimal classification threshold

Any fish with length < 11 = salmon, length > 11 = sea bass

# Classification

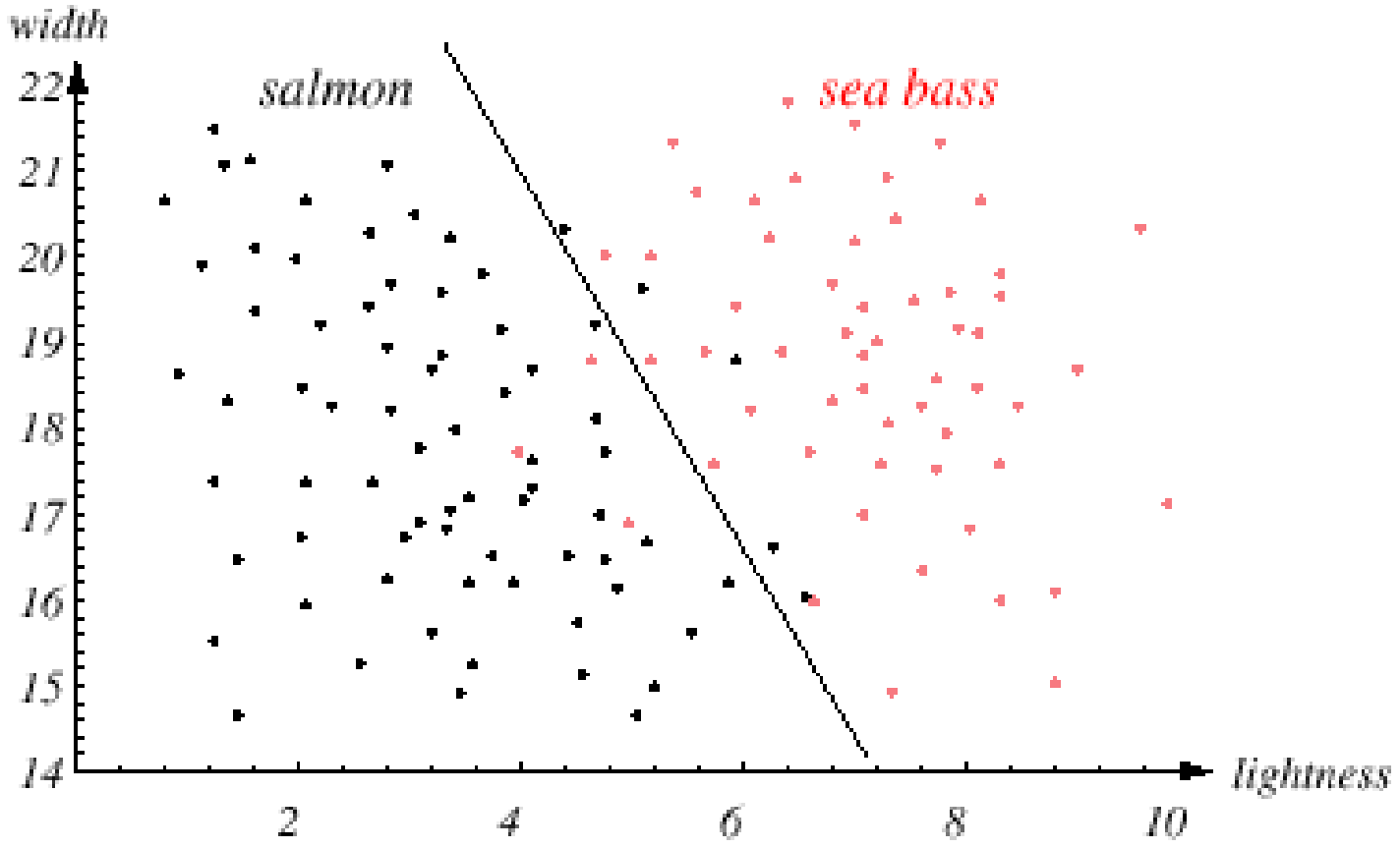
The length is a poor feature alone!

Select the lightness as a possible feature.



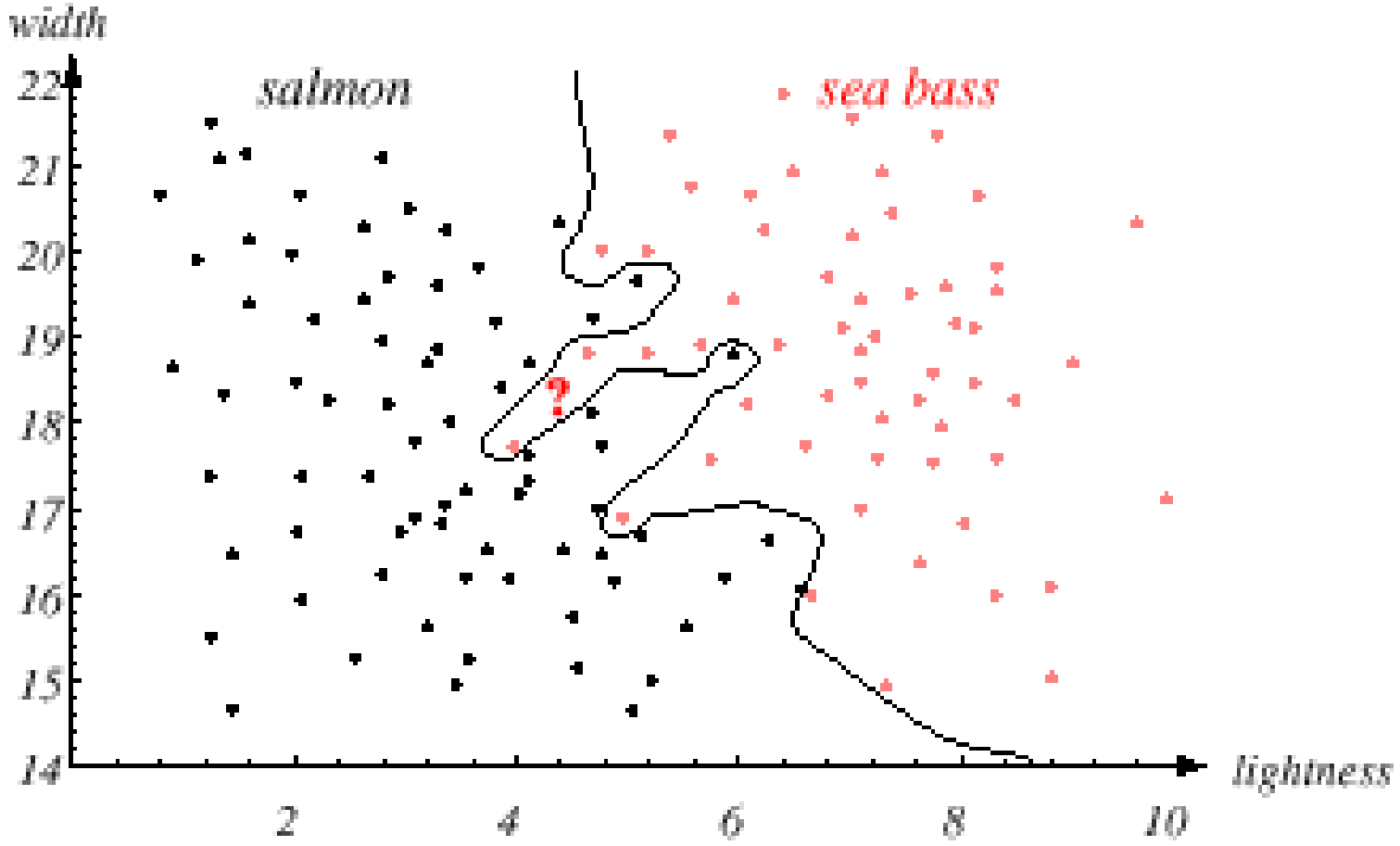
# Two Features

Where to place the decision boundary?  Should it be linear?




2D Decision Boundary

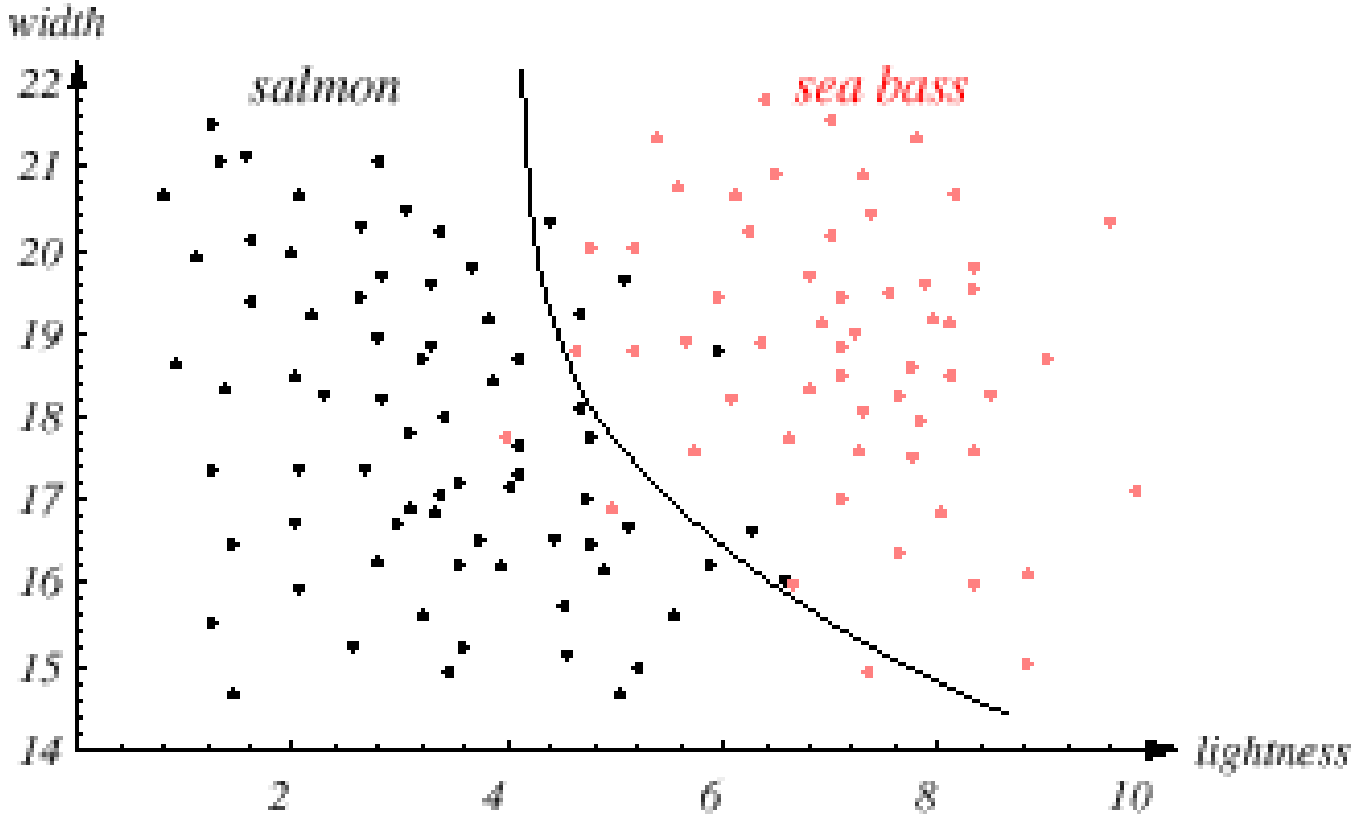
# Optimal Decision Boundary?




**Overfitting** – you nailed the training data, but do poorly on different fish. Model doesn't *generalize* well.

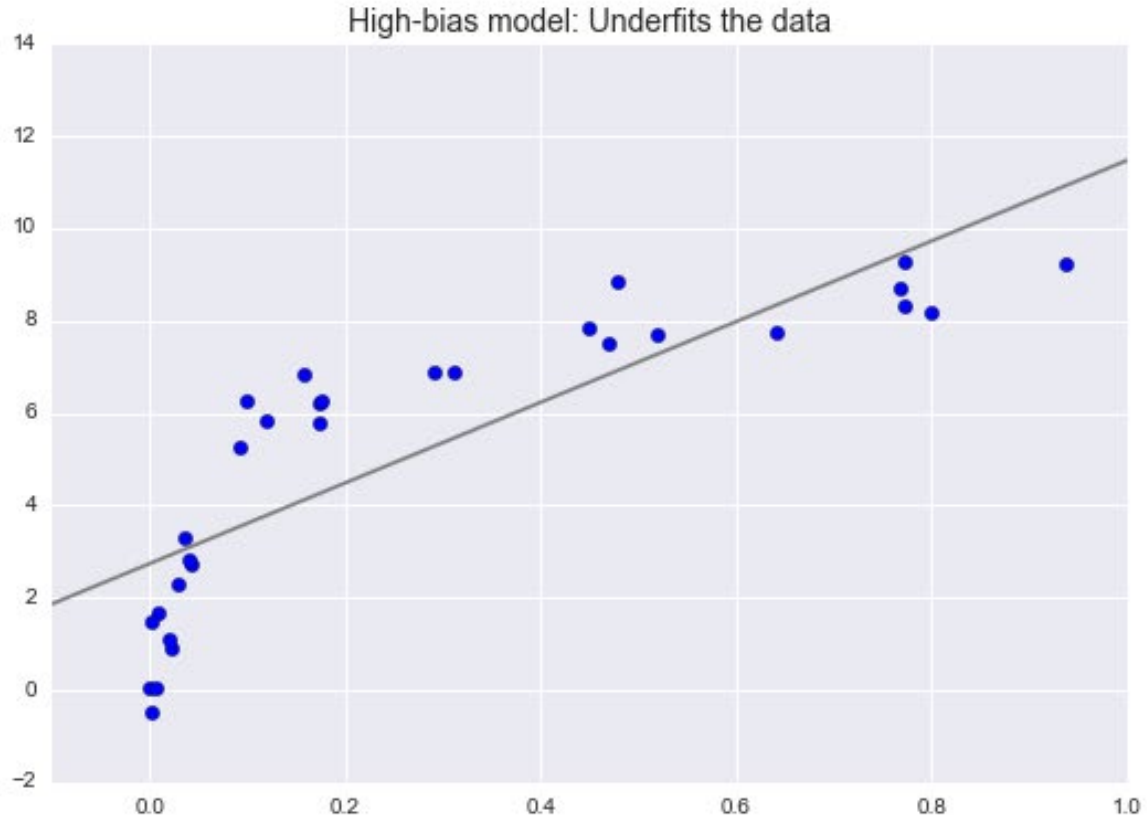
# Generalization

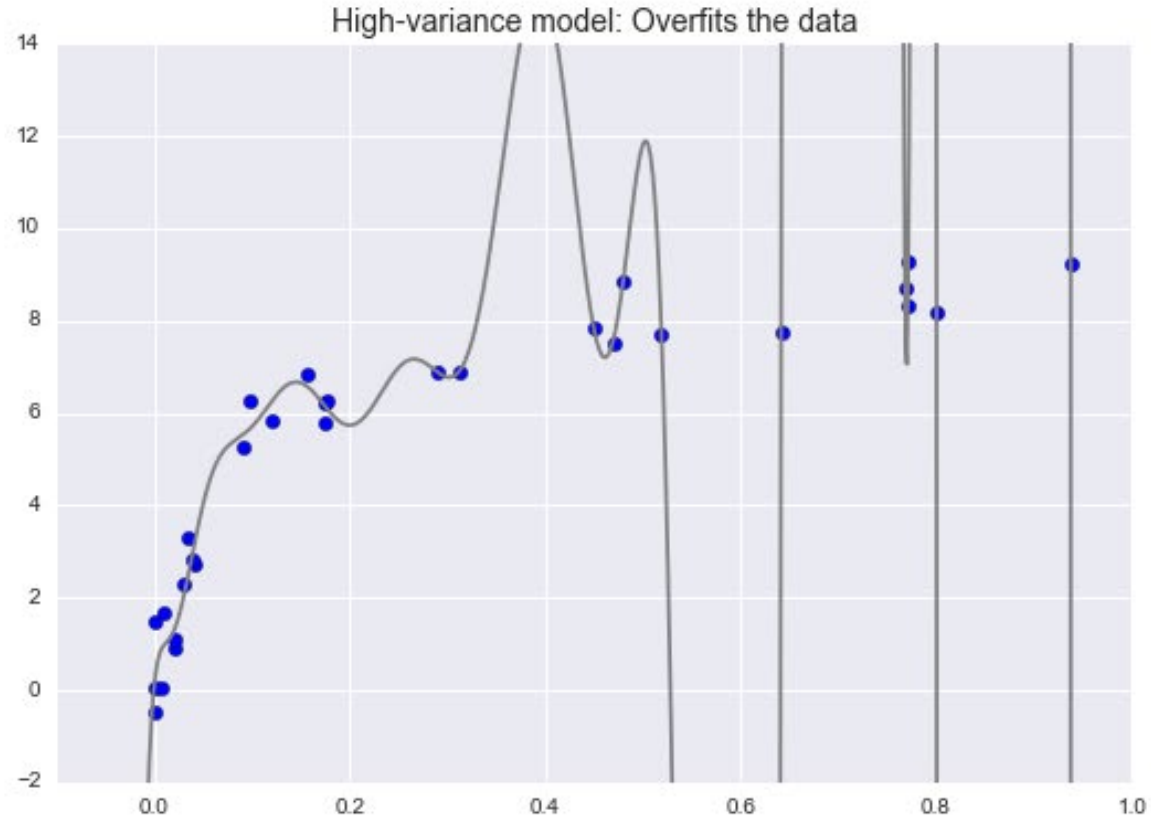
Goldilocks' third choice?



# Bias vs Variance Tradeoff

http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.03-Hyperparameters-and-Model-Validation.ipynb




**UNDERFITTING:**

- High bias: too simple to capture shape of data
- Low variance: not sensitive to statistical noise

**OVERFITTING:**

- Low bias: very flexible, can make complicated shapes
- High variance: **VERY** sensitive to statistical noise

# Evaluation

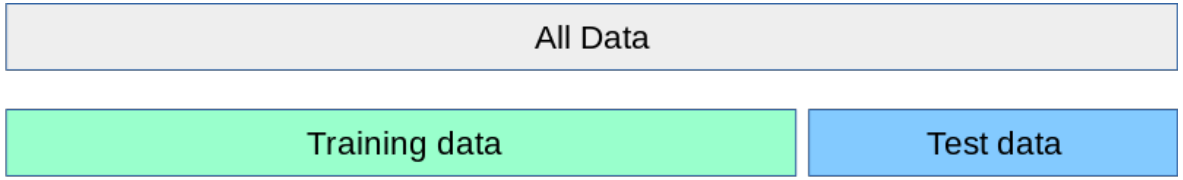
- **Overfitting** – model trained to peculiarities in the training data rather than general samples
- **Underfitting** – model not able to perform well on any data

Our best check for this is to compare the model's performance on:

1) data used for training the model
2) data the model has never seen before

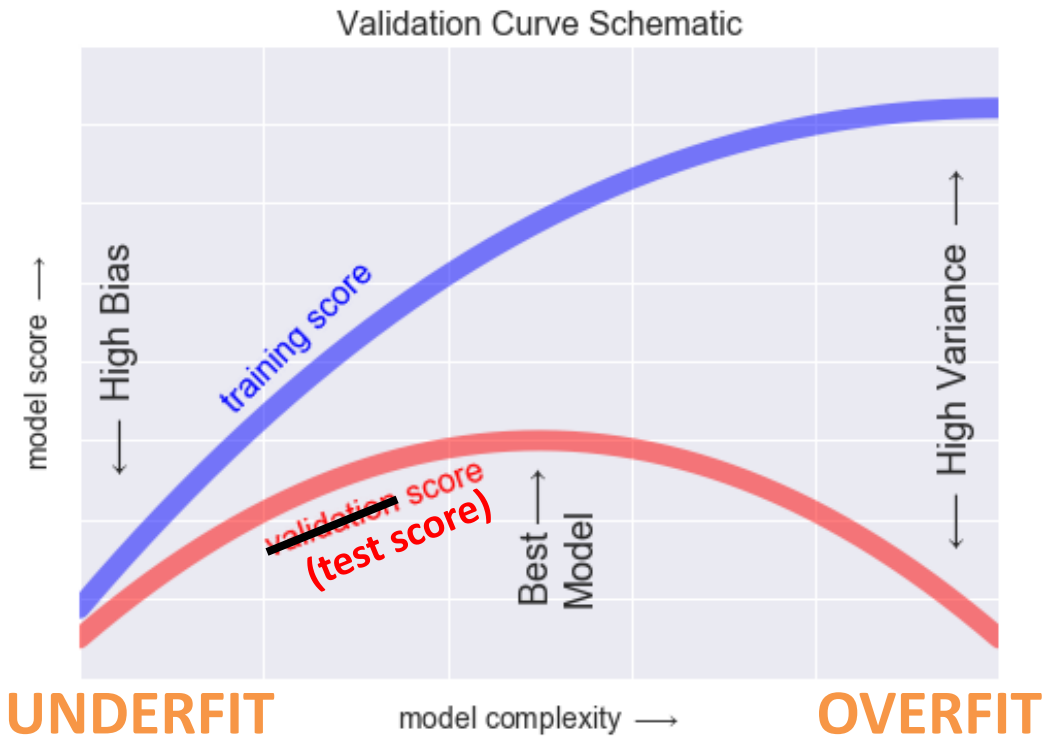
This trick is called the **test/train split**

# Test/Train Split




- The simplest and most common weapon against overfitting is the **test/train split**
- Train: use this part of the data for training
- Test: do **not** train on this part of the data, check how well our model works on data it hasn't seen yet
- **You can NOT fairly evaluate with data you trained on!**

http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.03-Hyperparameters-and-Model-Validation.ipynb



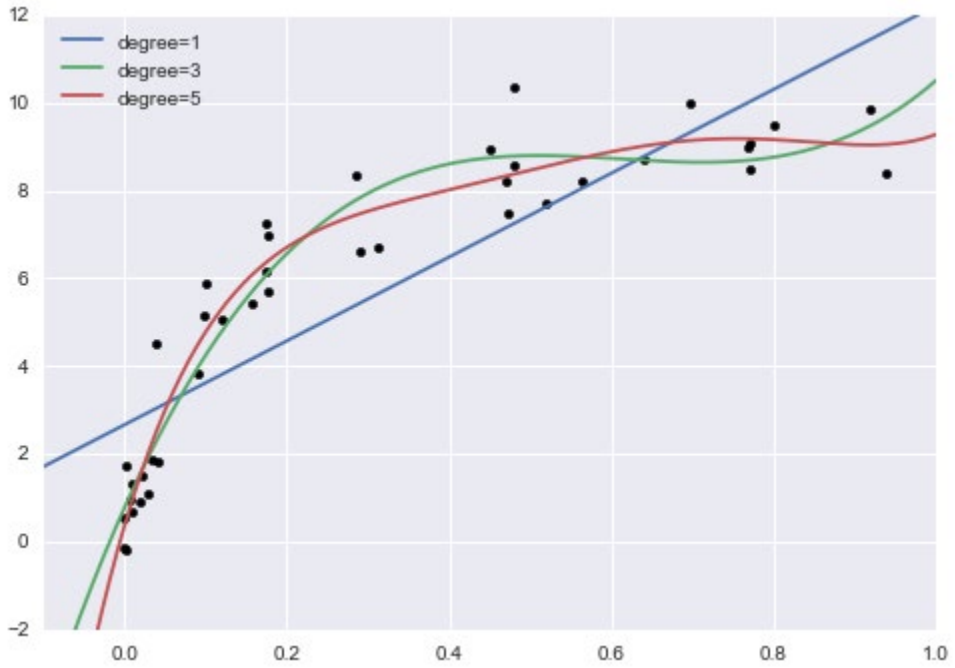

•The training score is everywhere higher than the validation score. This is generally the case: the model will be a better fit to data it has seen than to data it has not seen.

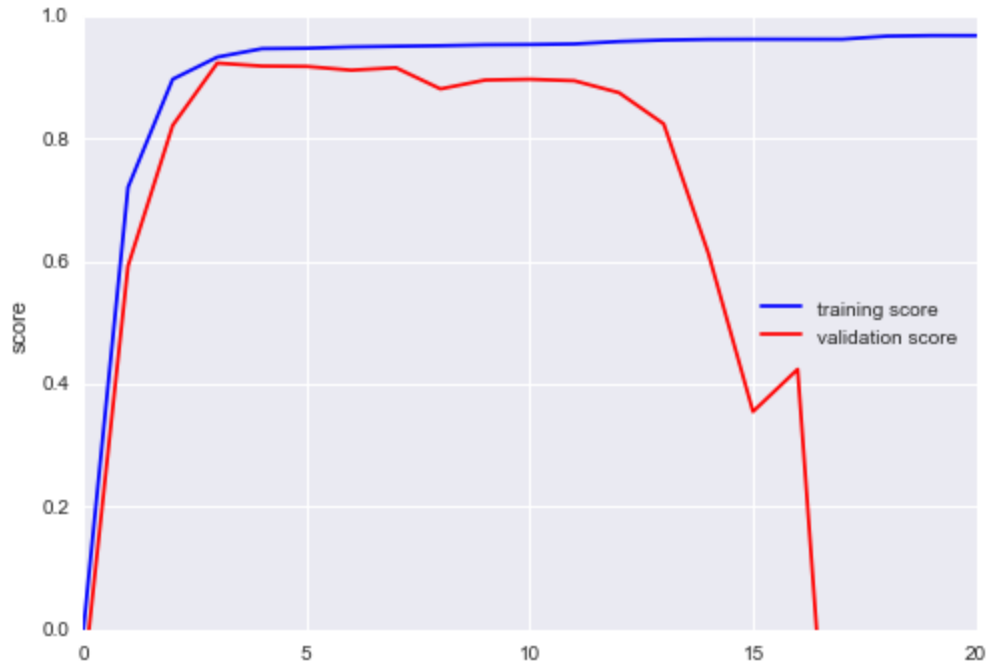
•Very **low** model complexity: the training data is **under-fit**, poor predictor both for the training data and for any previously unseen data.

•Very **high** model complexity: the training data is **over-fit**, model predicts the training data very well but fails for any previously unseen data.

•For some intermediate value, the validation curve has a maximum at best trade-off

http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.03-Hyperparameters-and-Model-Validation.ipynb



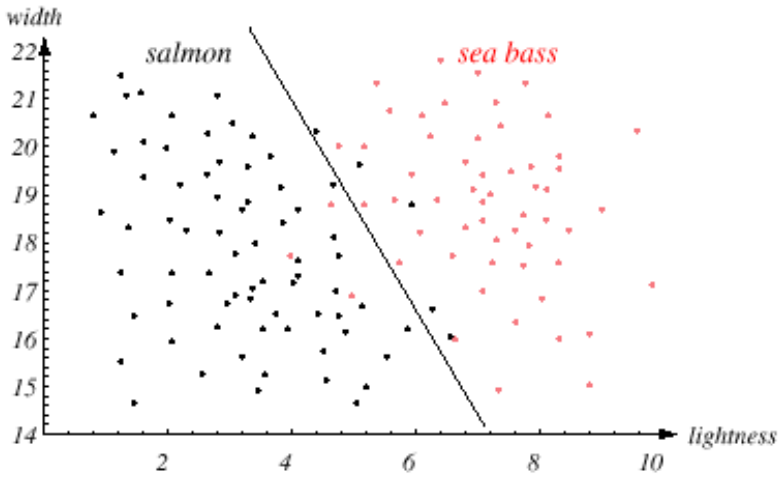

UNDERFIT

OVERFIT

Fitting data with different degree polynomials. The validation curve shows the n=3 is optimal.

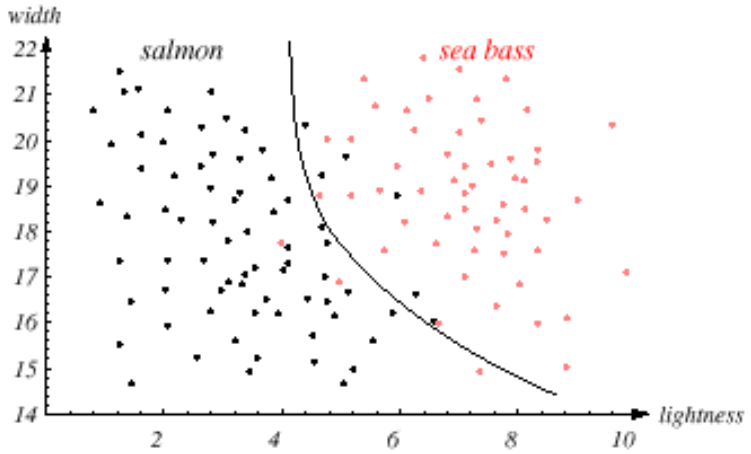
# Test/Train Split

Remember that every problem is different.  No way to tell ahead of time how a model will perform!



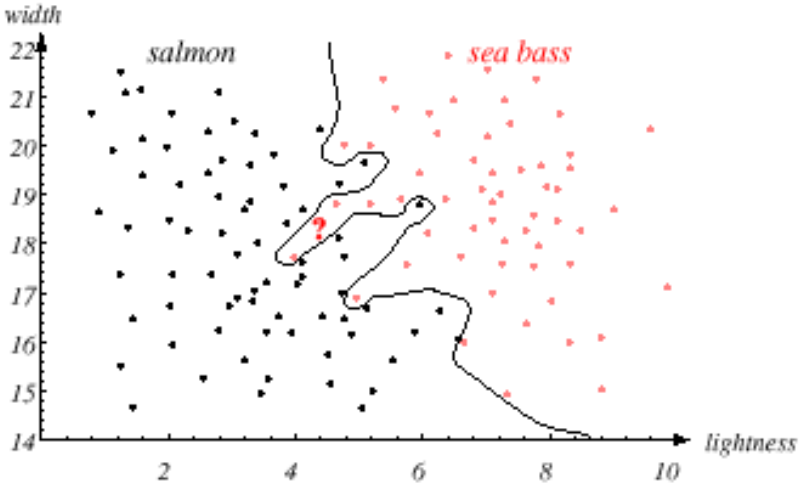

Linear

- high bias, low complexity
- underfitting




Smooth

- medium bias and complexity
- Fitting well!




Crazy

- low bias, high complexity
- overfitting

# Test/Train Split

| | **Bad Testing Score** | **Good Testing Score** |
|---|---|---|
| **Bad Training Score** | UNDERFITTING<br>High Bias / Low Variance | Generally impossible |
| **Good Training Score** | OVERFITTING<br>Low Bias / High Variance | It works! |

# Summary

- Features X
- Target y
- Underfitting/Fitting/Overfitting
- Bias/Variance
- Test/Train Split