

COMP541

DEEP LEARNING

© Lecture #2 – Machine Learning Overview

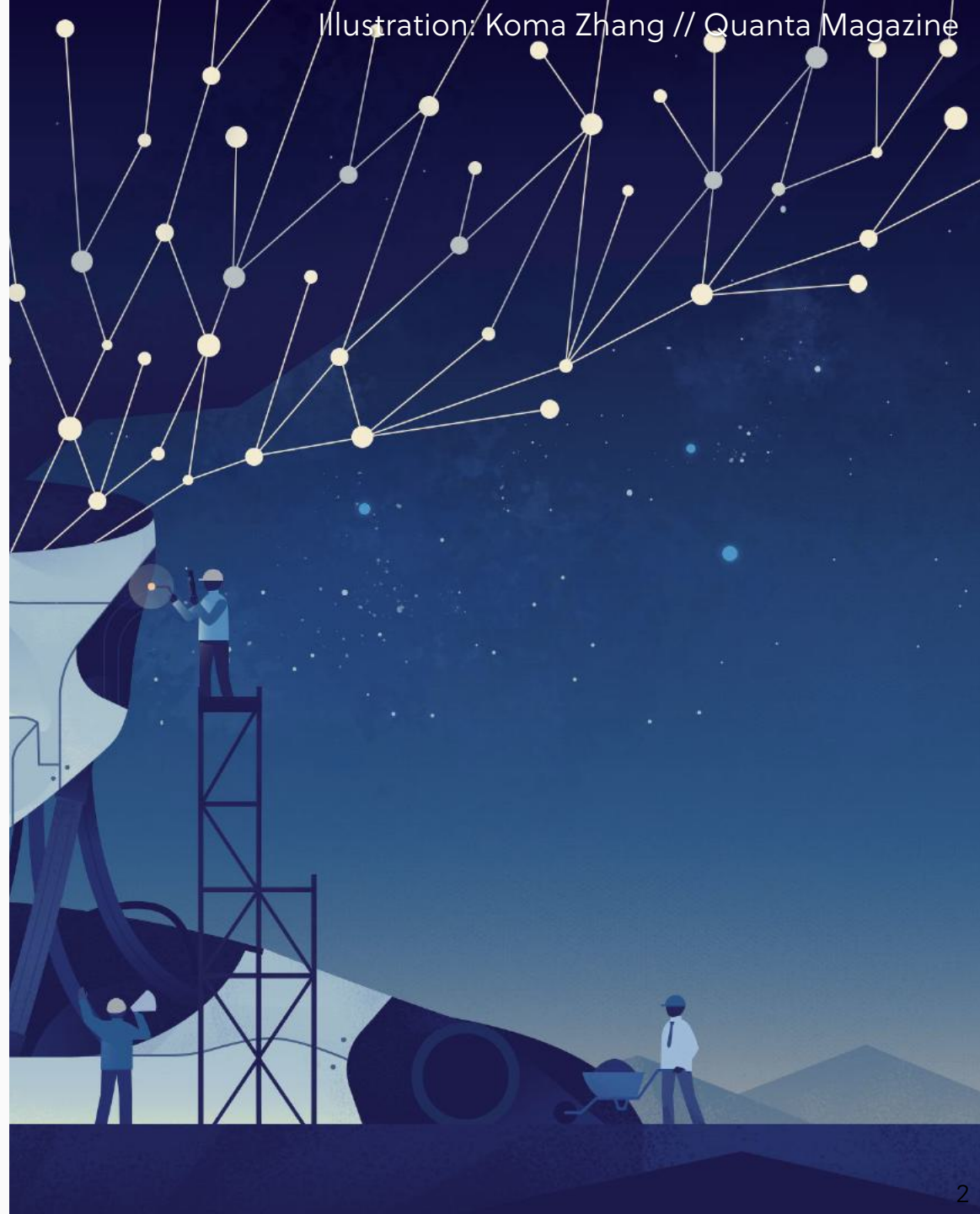


KOÇ
UNIVERSITY

Aykut Erdem // Koç University // Fall 2025

Previously on COMP541

- what is deep learning
- a brief history of deep learning
- compositionality
- end-to-end learning
- distributed representations



Good news, everyone!

- Tutorials will start this week!
- My office hours will be on Wednesdays btw 17:30-18:30
- The min. attendance requirement is reduced to 70%
See the announcement by Academic Planning and Development Management (shared on September 26)



Lecture overview

- what is learning?
- types of machine learning problems
- image classification
- linear regression
- generalization
- cross-validation
- maximum likelihood estimation

Disclaimer: Much of the material and slides for this lecture were borrowed from

- Bernhard Schölkopf's MLSS 2017 lecture,
- Tommi Jaakkola's 6.867 class,
- Fei-Fei Li and Andrej Karpathy's CS231n class
- Justin Johnson's EECS598 class

What is learning?

Two definitions of learning

- “Learning is the acquisition of knowledge about the world.”
Kupfermann (1985)
- “Learning is an adaptive change in behavior caused by experience.”
Shepherd (1988)

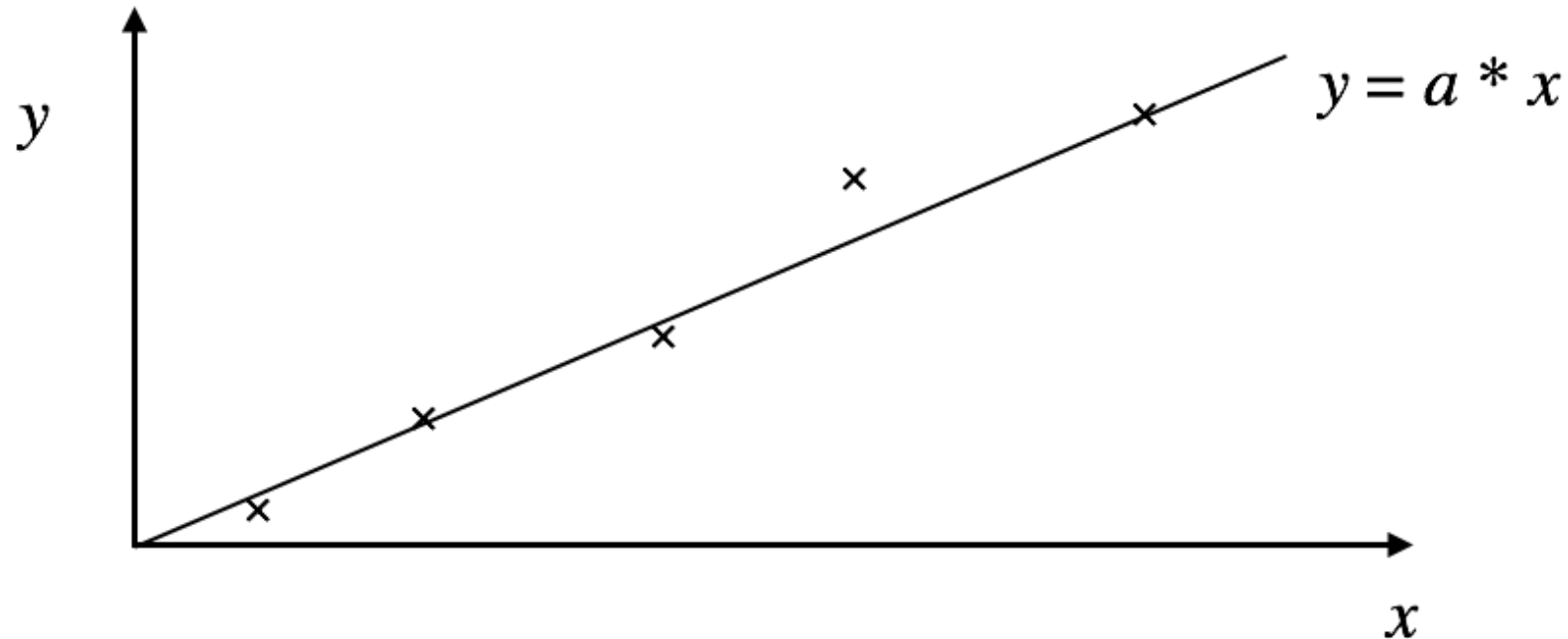
Empirical Inference

- Drawing conclusions from empirical data (observations, measurements)
- Example 1: scientific inference



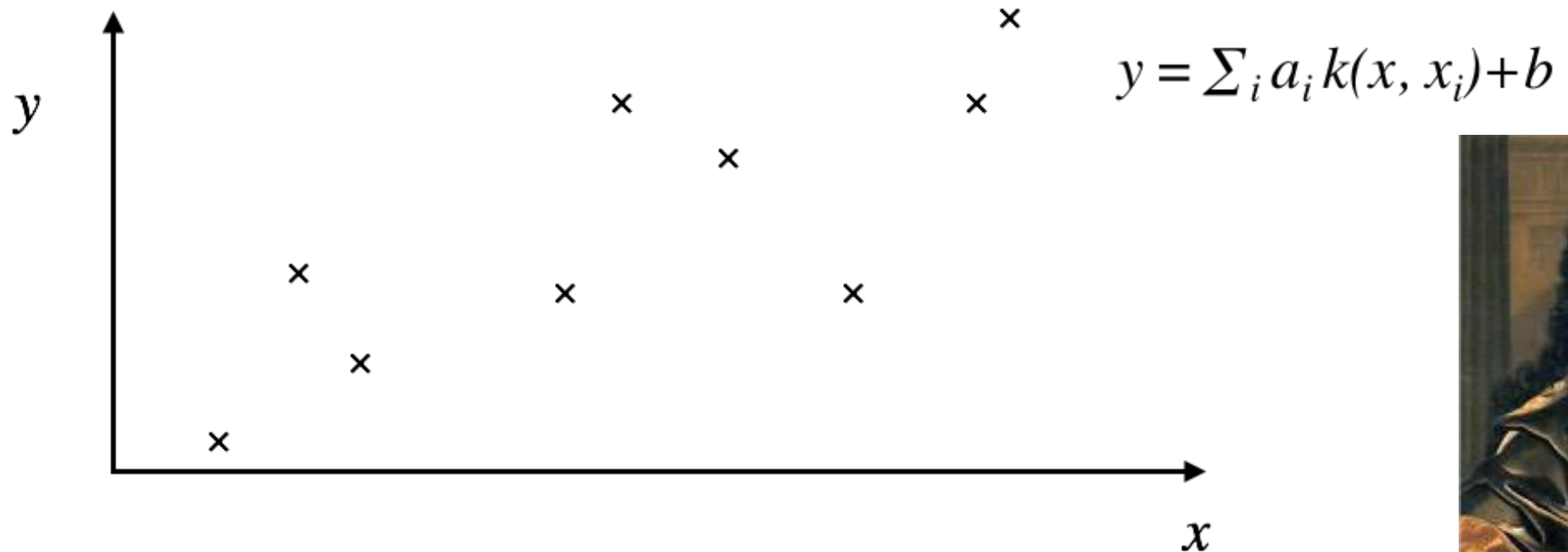
Empirical Inference

- Drawing conclusions from empirical data (observations, measurements)
- Example 1: scientific inference



Empirical Inference

- Drawing conclusions from empirical data (observations, measurements)
- Example 1: scientific inference

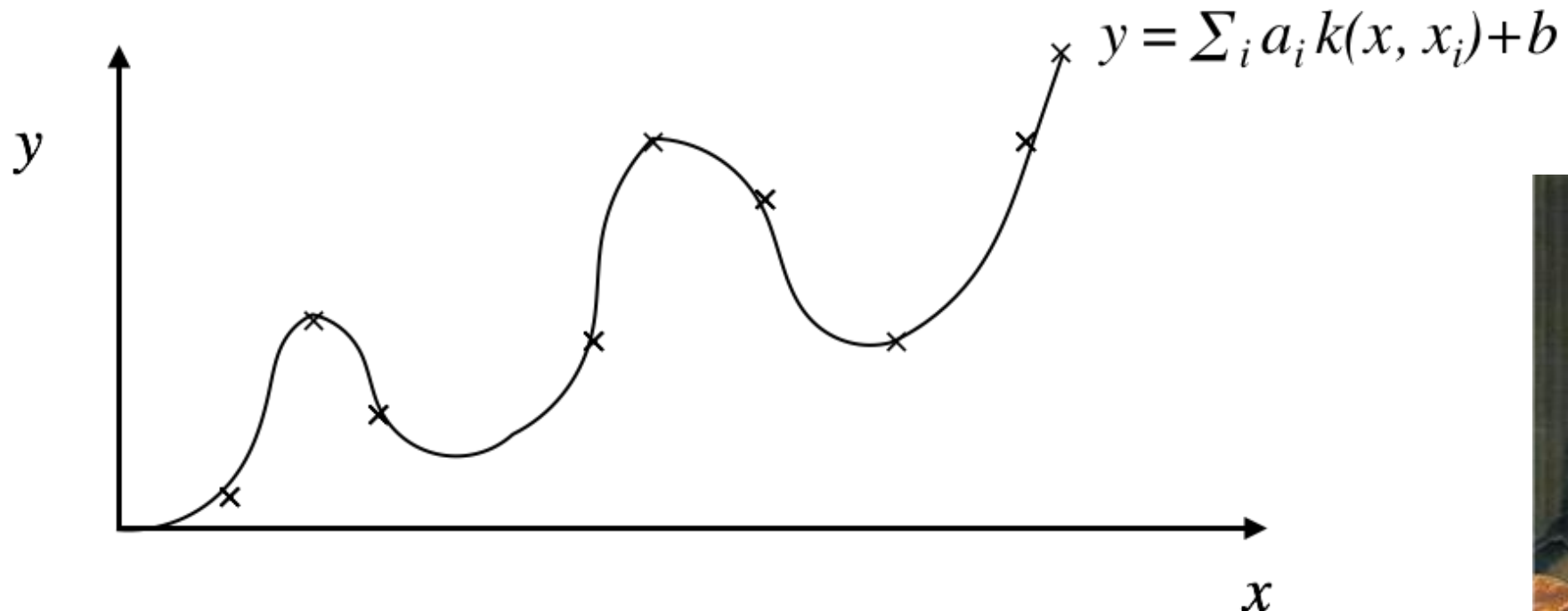


Leibniz, Weyl, Chaitin



Empirical Inference

- Drawing conclusions from empirical data (observations, measurements)
- Example 1: scientific inference

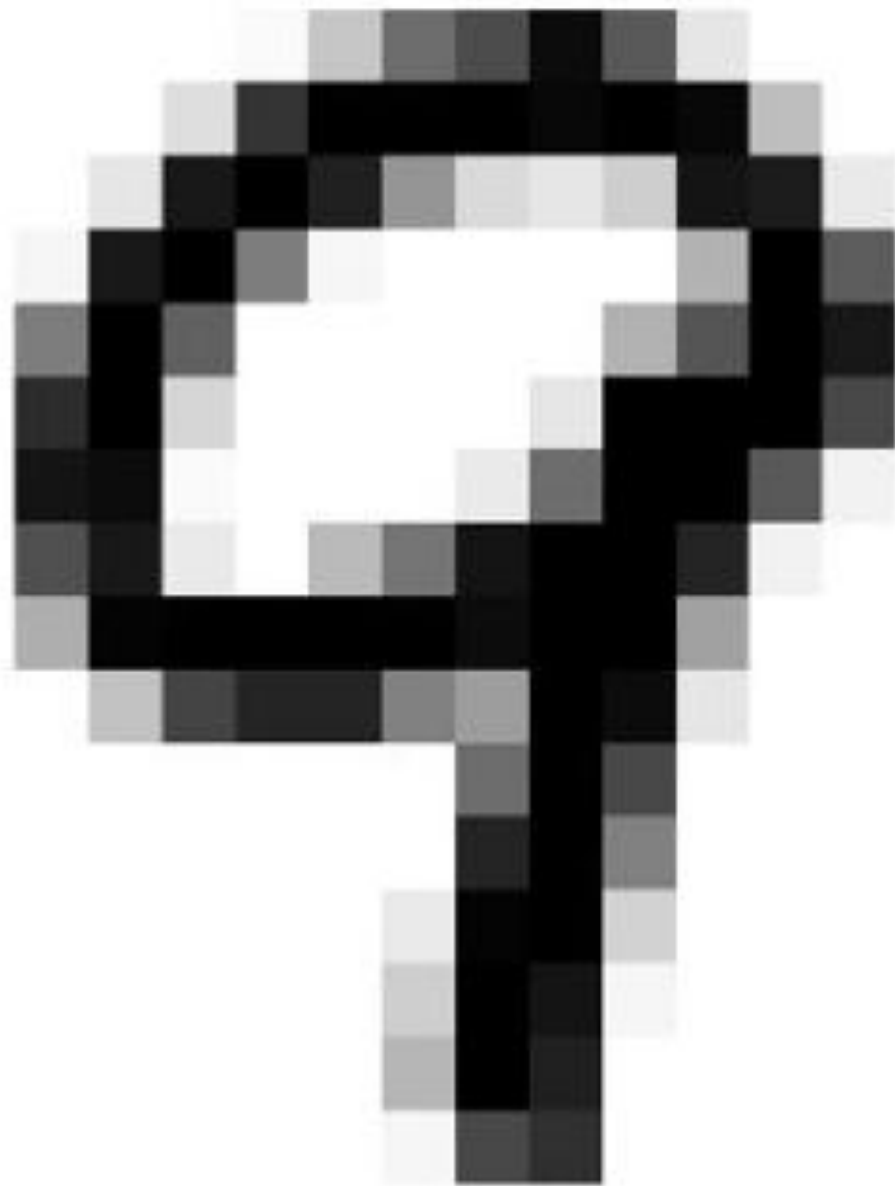


Leibniz, Weyl, Chaitin



Empirical Inference

- Example 2: perception



9



9





8



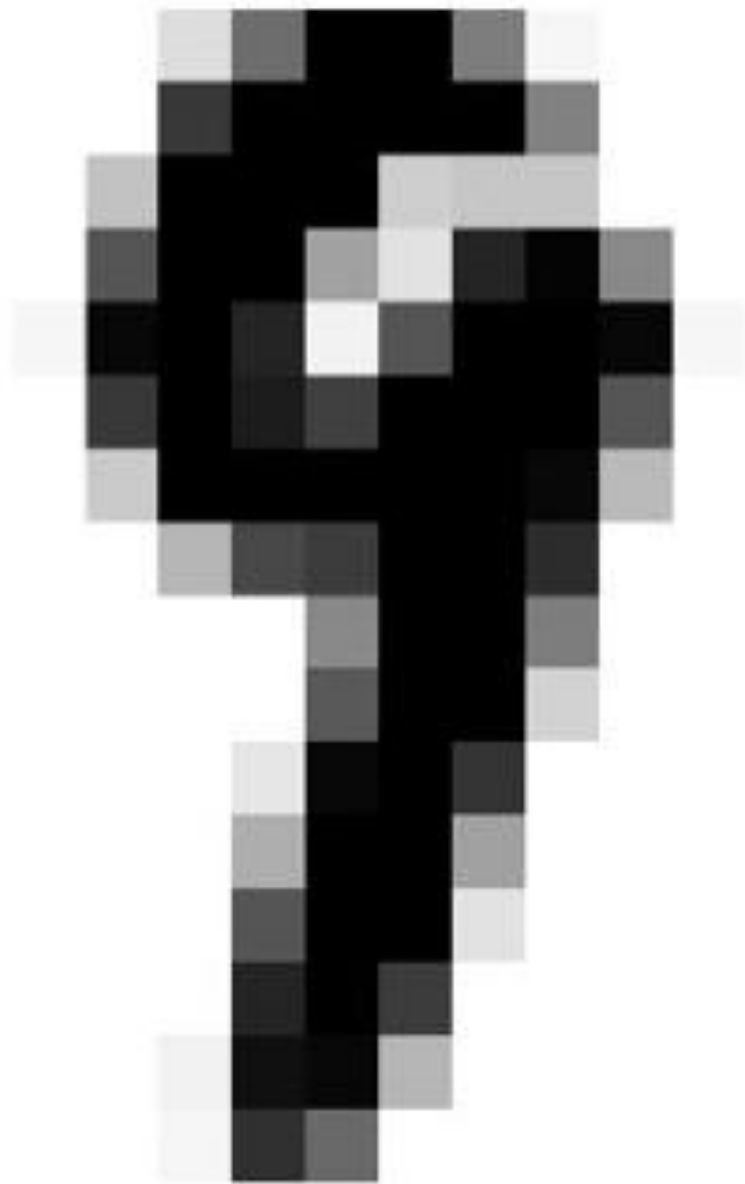
9



8



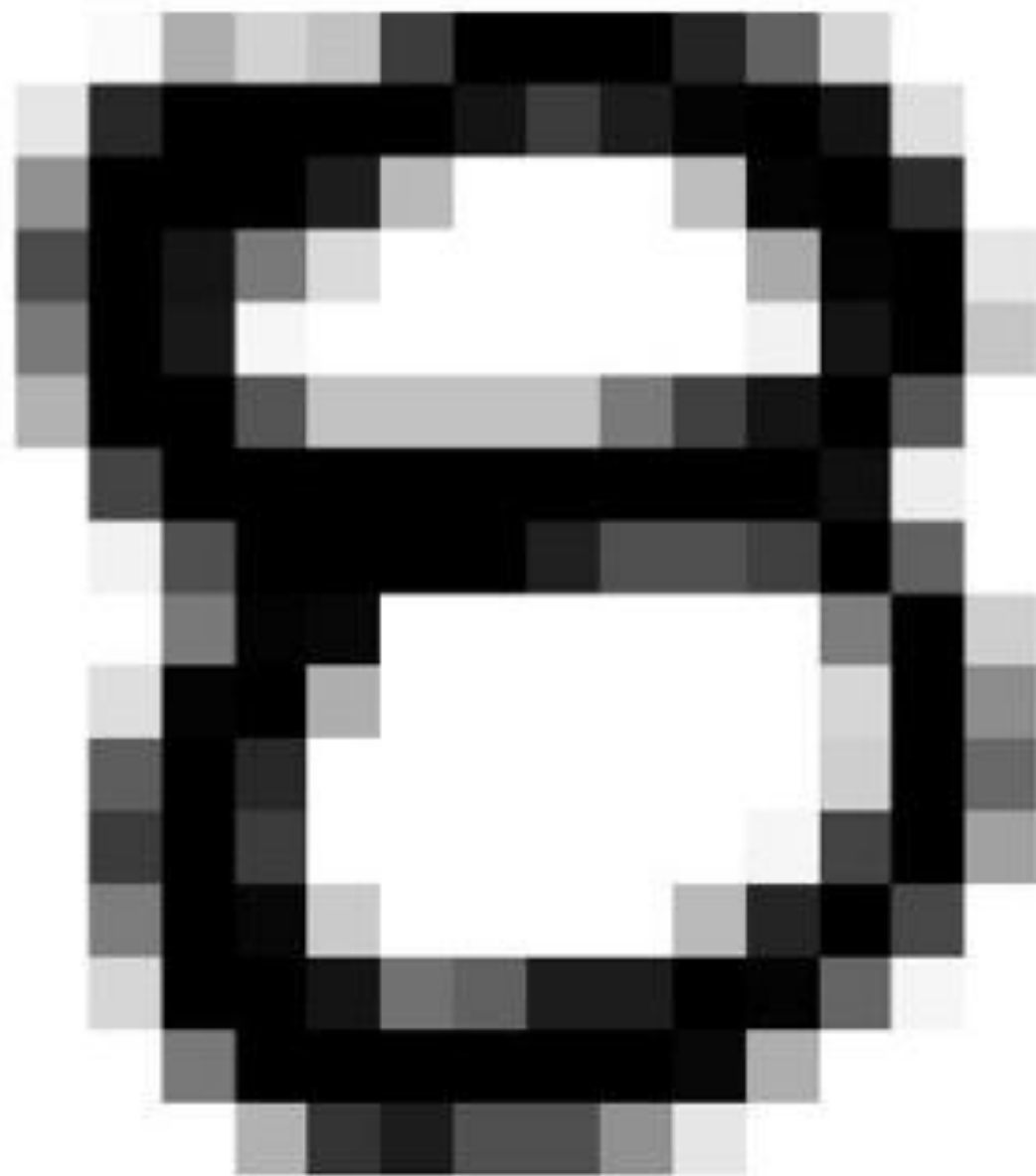
8



9



8



8



9









8



9





8





8





8



8



The choice of representation may determine whether the learning task is very easy or very difficult!

Generalization

- *observe*

- What's next?

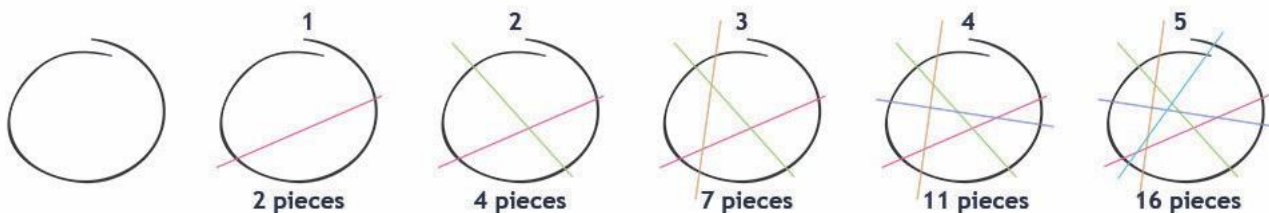
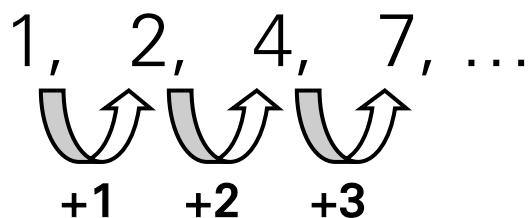


Image credit: mathspice.com

- 1, 2, 4, 7, 11, 16, ...: $a_{n+1} = a_n + n$ ("lazy caterer's sequence")

- 1, 2, 4, 7, 12, 20, ...: $a_{n+2} = a_{n+1} + a_n + 1$

- 1, 2, 4, 7, 13, 24, ...: "Tribonacci"-sequence

- 1, 2, 4, 7, 14, 28 : divisors of 28

- 1, 2, 4, 7, 1, 1, 5, ... : decimal expansions of $\pi = 3.14159...$ and $e = 2.718...$ interleaved (thanks to O. Bousquet)

- don't need e : 1247 appears at position 16992 in π

- [The On-Line Encyclopedia of Integer Sequences](#): > 1300 hits...



Generalization, II

- Question: which continuation is correct ("generalizes")?
- *Answer?* There's no way to tell (*"induction problem"*)
- Question of statistical learning theory: how to come up with a law that generalizes (*"demarcation problem"*)

Types of ML problems

Types of machine learning problems

Based on the information available:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

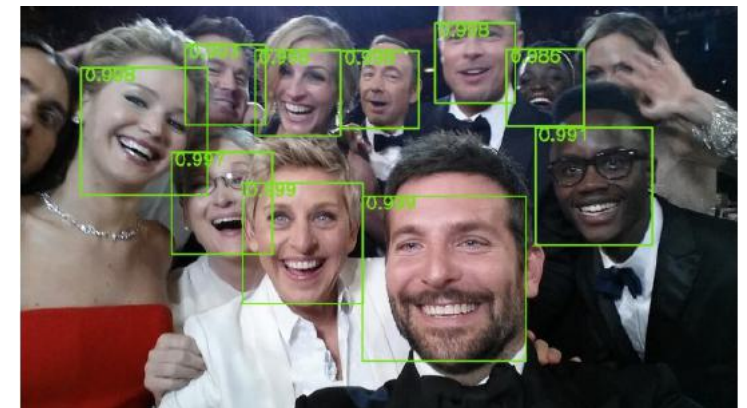
Supervised learning

- **Input:** $\{(\mathbf{x}, y)\}$
- **Task:** Predict target y from input $\mathbf{x}$
 - **Classification:** Discrete output
 - **Regression:** Real-valued output



→ cat

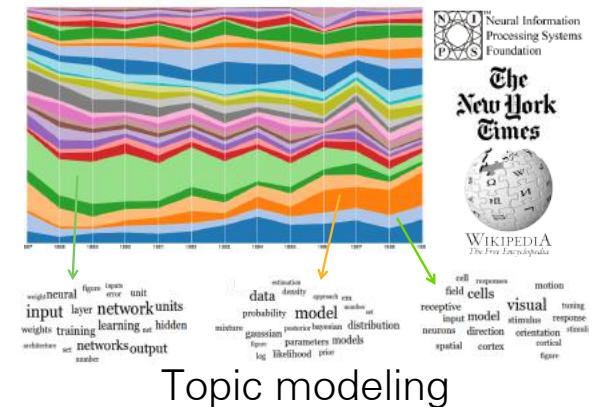
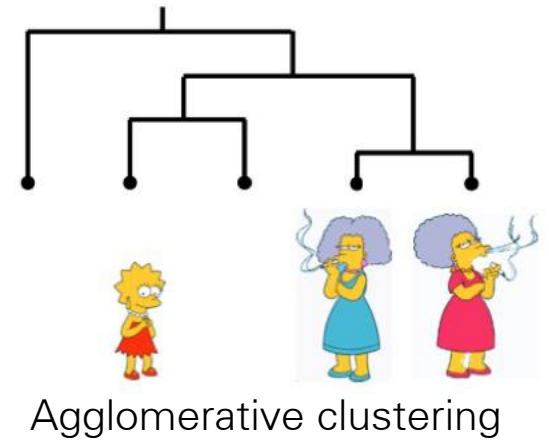
Image classification



Face detection

Unsupervised learning

- **Input:** $\{\mathbf{x}\}$
- **Task:** Reveal structure in the observed data
 - **Clustering:** Partition data into groups
 - **Feature extraction:** Learning meaningful features automatically
 - **Dimensionality reduction:** Learning a lower-dimensional representation of input



Semi-supervised learning

- **Input:**

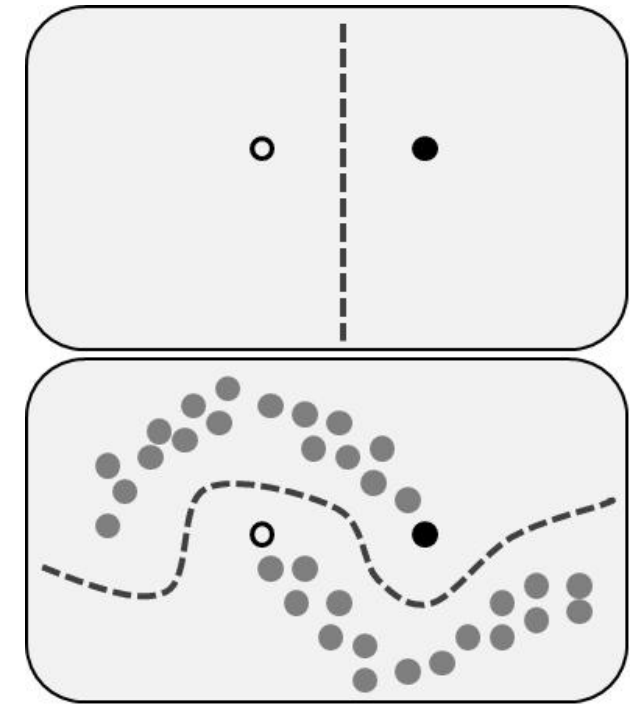
Few labeled examples $\{(\mathbf{x}, y)\}$

Many unlabeled examples $\{\mathbf{x}\}$

- **Task:** Predict target y from input $\mathbf{x}$

- Classification: Discrete output
- Regression: Real-valued output

Try to improve predictions based on examples by making use of the additional “unlabeled” examples



interactive segmentation

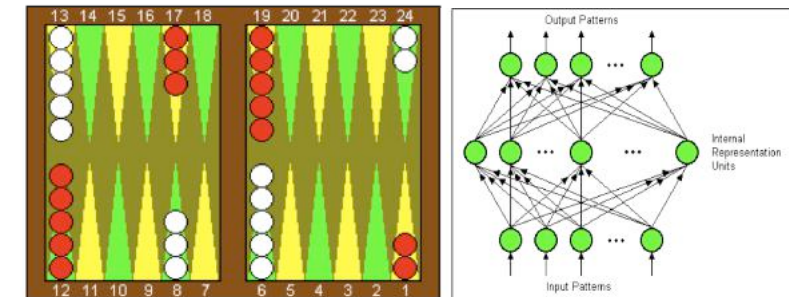
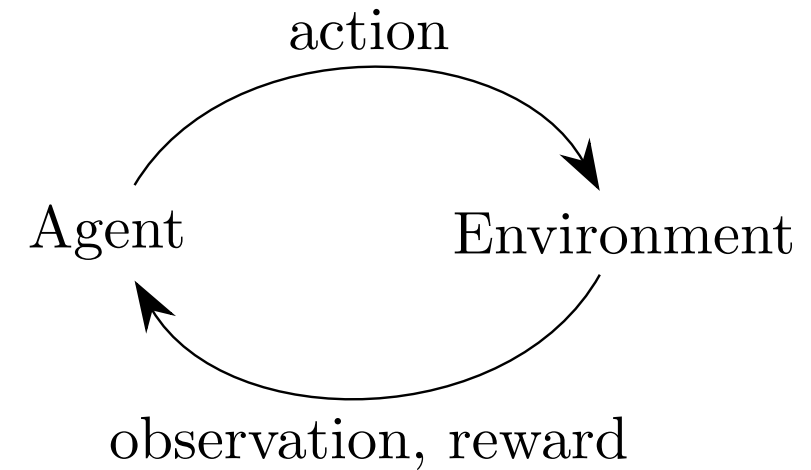
Reinforcement learning

- **Input:**

Interaction with an environment;
the agent receives a numerical
reward signal

- **Task:** A way of behaving that is very rewarding
in the long run

- Goal is to estimate and maximize the
long-term cumulative reward



TD-Gammon (Tesauro, 1990-1995)

Types of machine learning problems

How Much Information Does the Machine Need to Predict?

Y LeCun

■ “Pure” Reinforcement Learning (cherry)

- ▶ The machine predicts a scalar reward given once in a while.
- ▶ **A few bits for some samples**

■ Supervised Learning (icing)

- ▶ The machine predicts a category or a few numbers for each input
- ▶ Predicting human-supplied data
- ▶ **10→10,000 bits per sample**

■ Unsupervised/Predictive Learning (cake)

- ▶ The machine predicts any part of its input for any observed part.
- ▶ Predicts future frames in videos
- ▶ **Millions of bits per sample**



“If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake.”

– Yann LeCun
NIPS 2016 Keynote

■ (Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)

Image classification

- non-parametric vs. parametric models
- nearest neighbor classifier
- hyperparameter
- cross-validation

Image Classification: a core task in Computer Vision

Input: image

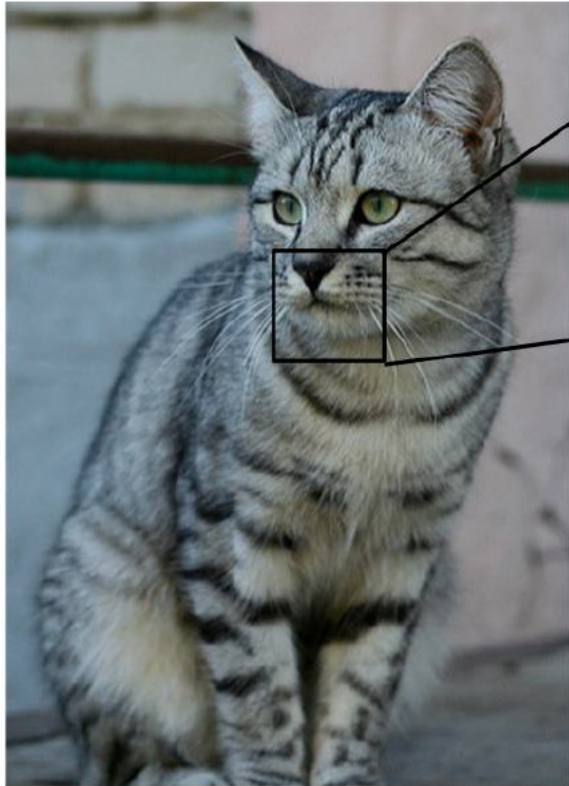


Output: Assign image to one of a fixed set of categories



cat
bird
deer
dog
truck

The problem: Semantic Gap



This image by Nikita is
licensed under [CC-BY 2.0](#)

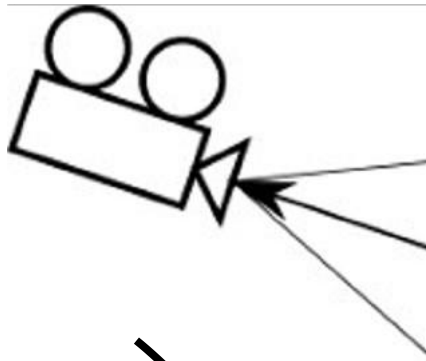
```
[[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]
 [ 91 98 102 106 104 79 98 103 99 105 123 136 110 105 94 85]
 [ 76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]
 [ 99 81 81 93 120 131 127 100 95 98 102 99 96 93 101 94]
 [106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]
 [114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
 [133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]
 [128 137 144 140 109 95 86 70 62 65 63 63 60 73 86 101]
 [125 133 140 137 119 121 117 94 65 79 80 65 54 64 72 98]
 [127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]
 [115 114 109 123 150 148 131 118 113 109 100 92 74 65 72 78]
 [ 89 93 90 97 108 147 131 118 113 114 113 109 106 95 77 80]
 [ 63 77 86 81 77 79 102 123 117 115 117 125 125 130 115 87]
 [ 62 65 82 89 78 71 80 101 124 126 119 101 107 114 131 119]
 [ 63 65 75 88 89 71 62 81 120 138 135 105 81 98 110 118]
 [ 87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]
 [118 97 82 86 117 123 116 66 41 51 95 93 89 95 102 107]
 [164 146 112 80 82 120 124 104 76 48 45 66 88 101 102 109]
 [157 170 157 120 93 86 114 132 112 97 69 55 70 82 99 94]
 [130 120 134 161 139 100 109 118 121 134 114 87 65 53 69 86]
 [128 112 96 117 150 144 120 115 104 107 102 93 87 81 72 79]
 [123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]
 [122 121 102 80 82 86 94 117 145 148 153 102 58 78 92 107]
 [122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255].

e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint Variation



[105	112	108	111	104	99	106	99	96	103	112	119	104	97	93	87]	
[	91	98	102	106	104	79	98	103	99	105	123	136	110	105	94	85]
[	76	85	90	105	128	105	87	96	95	99	115	112	106	103	99	85]
[	99	81	81	93	120	131	127	100	95	98	102	99	96	93	101	94]
[106	91	61	64	69	91	88	85	101	107	109	98	75	84	96	95]	
[114	108	85	55	55	69	64	54	64	87	112	129	98	74	84	91]	
[133	137	147	103	65	81	88	65	52	54	74	84	102	93	85	82]	
[120	137	144	140	109	95	86	70	62	65	63	63	60	73	86	101]	
[125	133	148	137	119	121	117	94	65	79	80	65	54	64	72	90]	
[127	125	131	147	133	127	126	131	111	96	89	75	61	64	72	84]	
[115	114	109	123	150	148	131	118	113	109	100	92	74	65	72	78]	
[	89	93	98	97	108	147	131	118	113	114	113	109	106	95	77	80]
[	63	77	86	81	77	79	102	123	117	115	117	125	125	130	115	87]
[	62	65	82	89	78	71	88	101	124	126	119	101	107	114	131	119]
[	63	65	75	88	89	71	62	81	120	138	135	105	81	98	110	118]
[	87	65	71	87	106	95	69	45	76	130	126	107	92	94	105	112]
[118	97	82	86	117	123	116	66	41	51	95	93	89	95	102	107]	
[164	146	112	80	82	120	124	104	76	48	45	66	88	101	102	109]	
[157	170	157	120	93	86	114	132	112	97	69	55	70	82	99	94]	
[130	128	134	161	139	100	109	118	121	134	114	87	65	53	69	86]	
[128	112	96	117	150	144	120	115	104	107	102	93	87	81	72	79]	
[123	107	96	86	83	112	153	149	122	109	104	75	80	107	112	99]	
[122	121	102	80	82	86	94	117	145	148	153	102	58	78	92	107]	
[122	164	148	103	71	56	78	83	93	103	119	139	102	61	69	84]]	

All pixels change when the camera moves!

Challenges: Intraclass Variation



Challenges: Fine-Grained Categories

Main Coon



Ragdoll



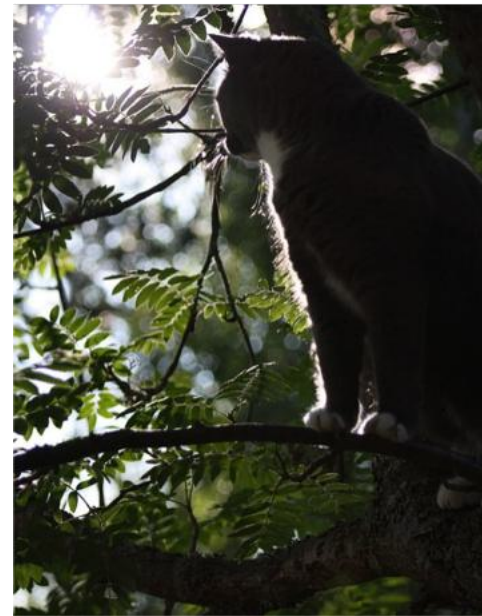
American Shorthair



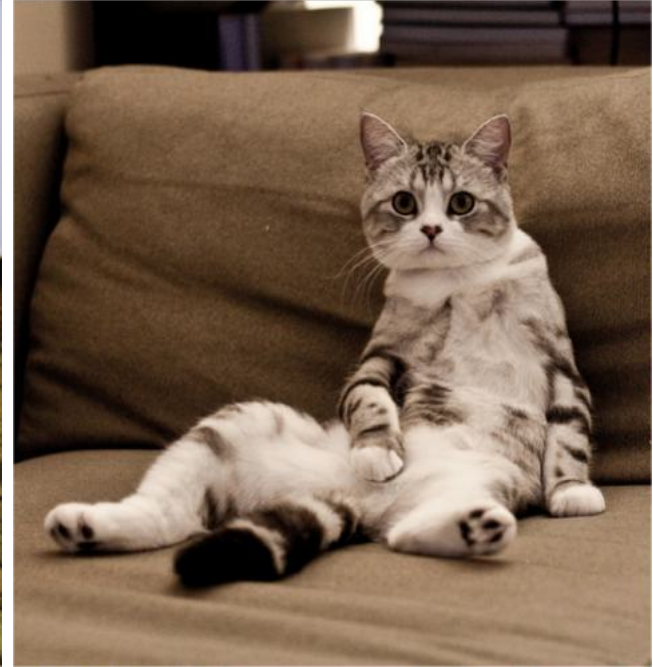
Challenges: Background clutter



Challenges: Illumination Changes



Challenges: Deformation



Challenges: Occlusion



Image Classification: Very Useful!

Medical Imaging

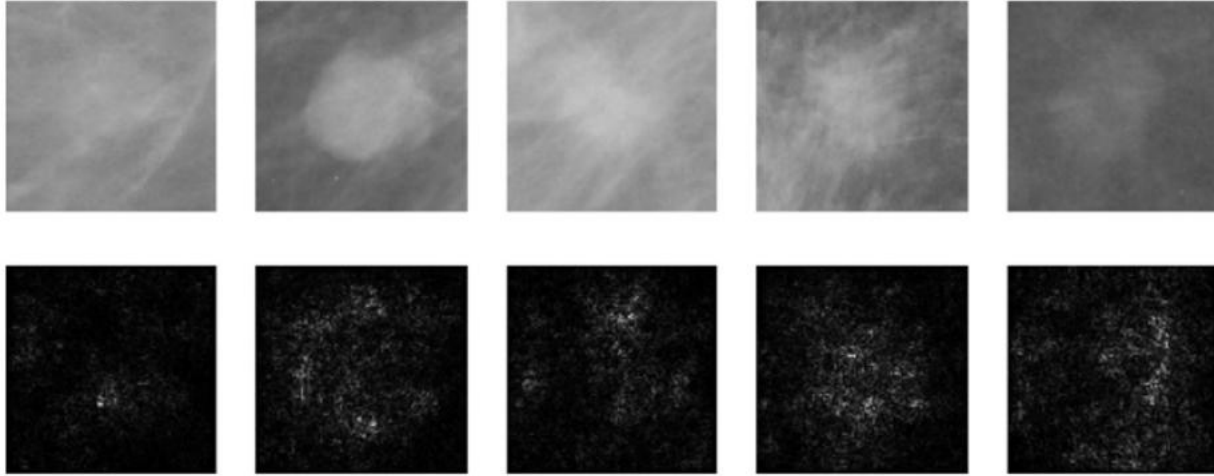
Benign

Benign

Malignant

Malignant

Benign



Galaxy Classification



Whale recognition



[Kaggle Challenge](#)

An image classifier

```
def classify_image(image):  
    # Some magic here?  
    return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

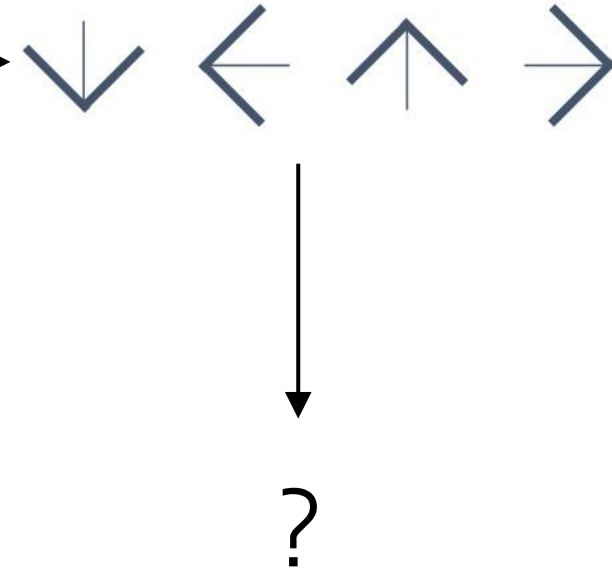
You could try ...



Find edges



Find corners



Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```
def train(images, labels):  
    # Machine learning!  
    return model
```

```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```

Example training set

airplane



automobile



bird



cat



deer



First classifier: Nearest Neighbor Classifier

```
def train(images, labels):  
    # Machine learning!  
    return model
```



Memorize all data
and labels

```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```



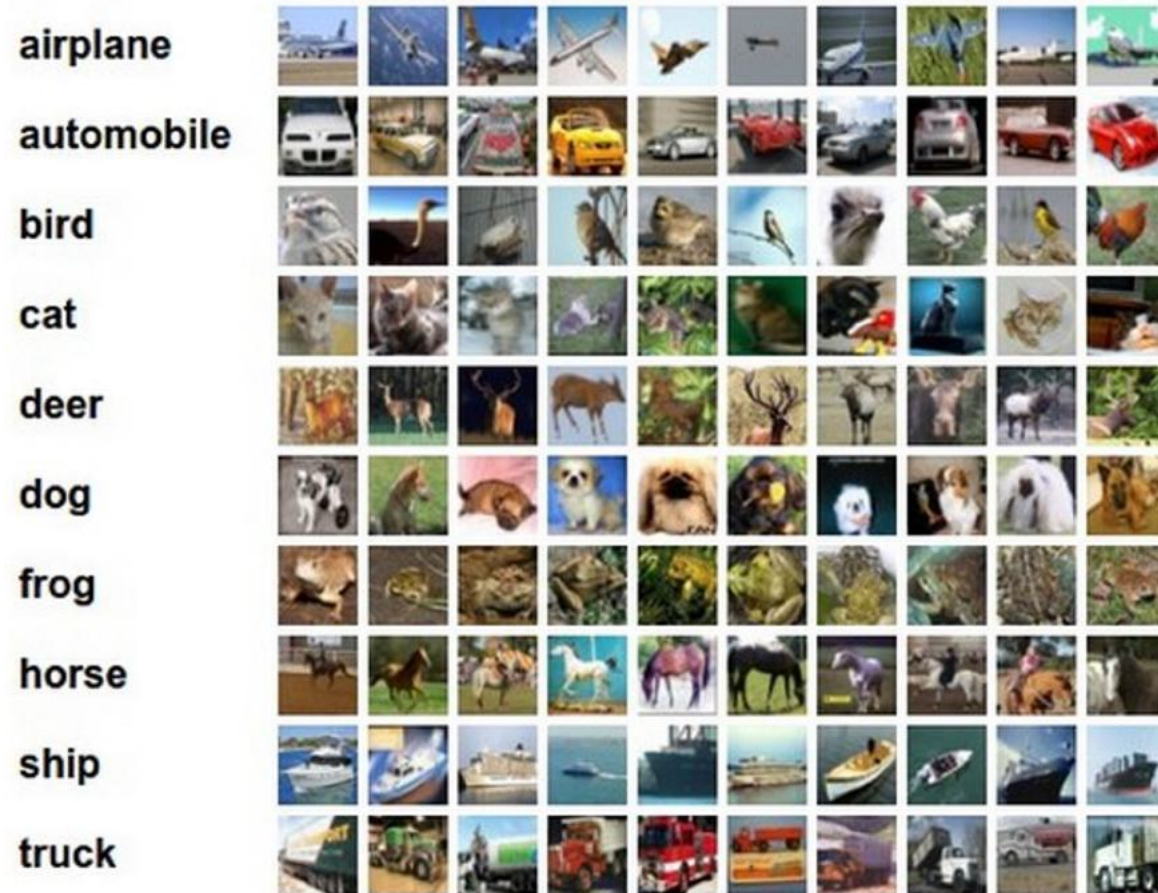
Predict the label of
the most similar
training image

Example dataset: **CIFAR-10**

10 labels

50,000 training images, each image is tiny: 32x32

10,000 test images.

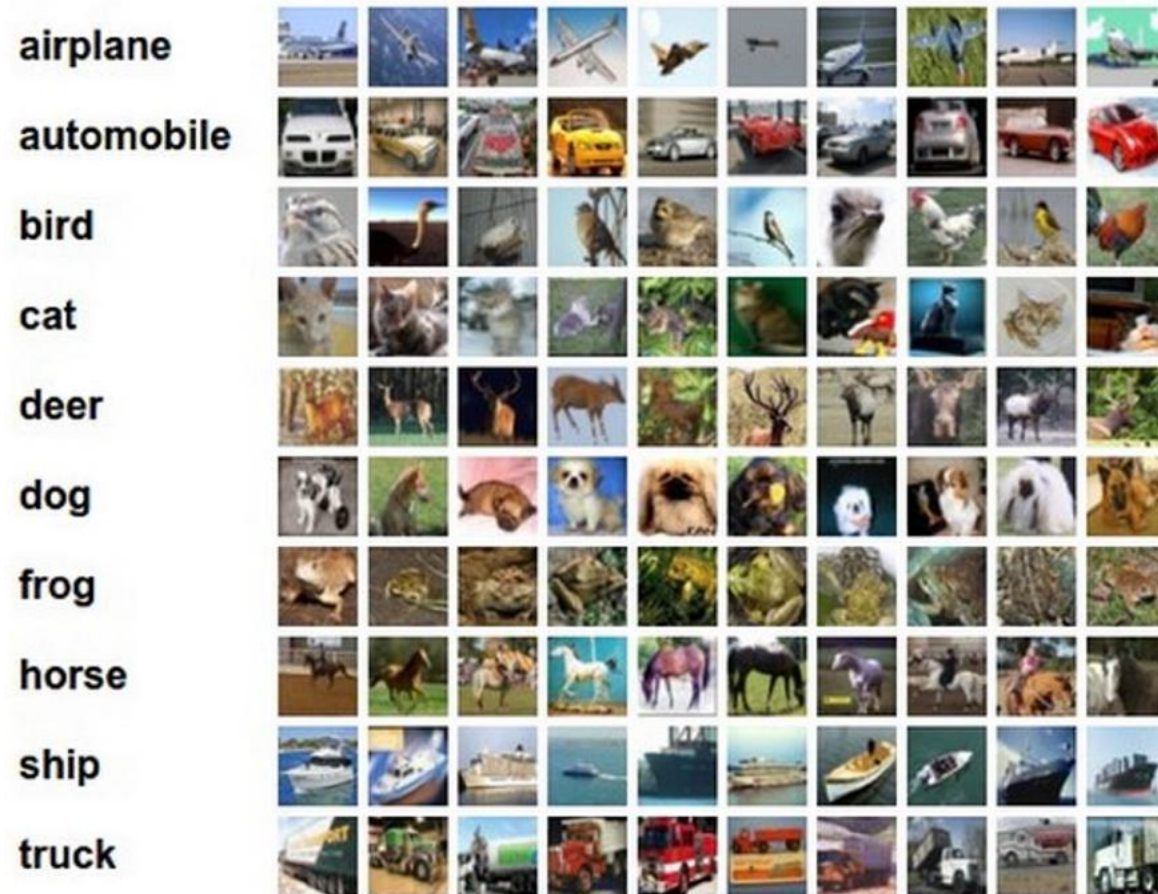


Example dataset: **CIFAR-10**

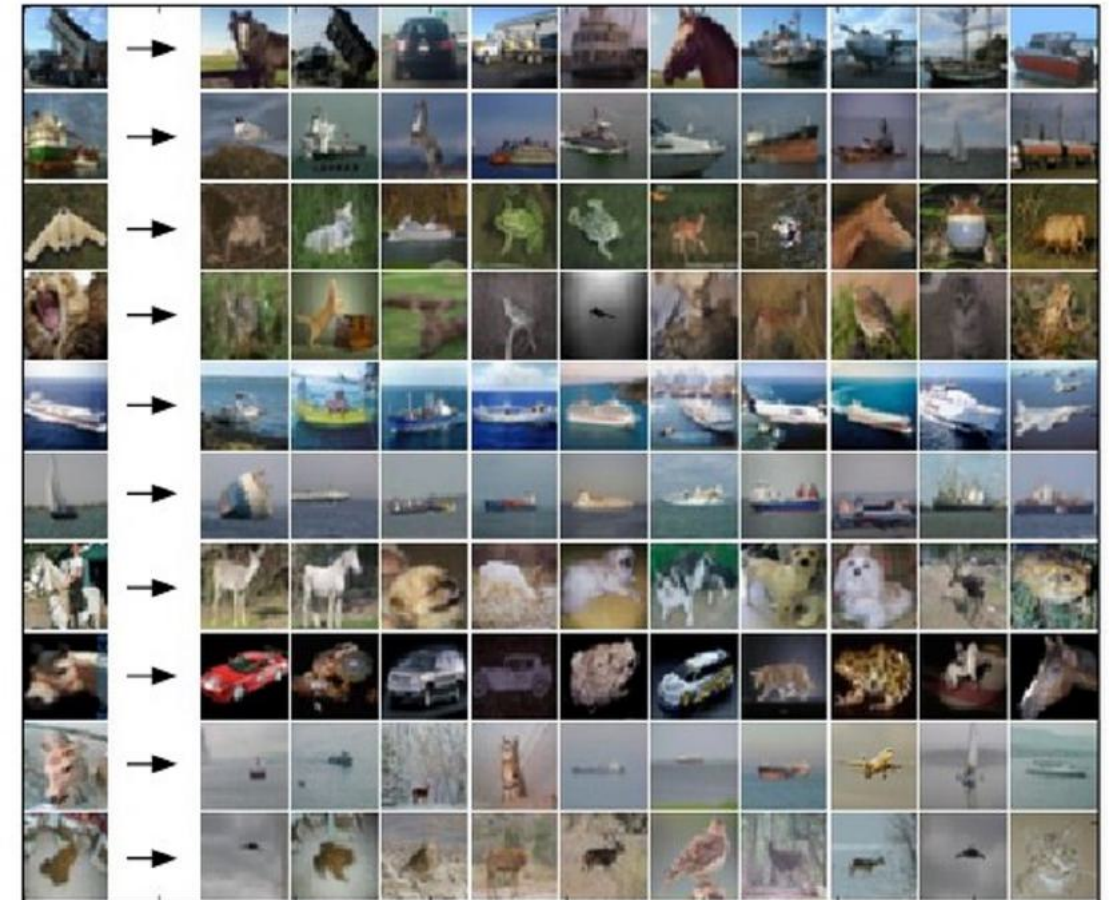
10 labels

50,000 training images

10,000 test images.



For every test image (first column),
examples of nearest neighbors in rows




```

import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

```

Nearest Neighbor classifier

Nearest Neighbor classifier

```
import numpy as np
```

```
class NearestNeighbor:
```

```
    def __init__(self):
```

```
        pass
```

```
    def train(self, X, y):
```

```
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
```

```
        # the nearest neighbor classifier simply remembers all the training data
```

```
        self.Xtr = X
```

```
        self.ytr = y
```

```
    def predict(self, X):
```

```
        """ X is N x D where each row is an example we wish to predict label for """
```

```
        num_test = X.shape[0]
```

```
        # lets make sure that the output type matches the input type
```

```
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)
```

```
        # loop over all test rows
```

```
        for i in xrange(num_test):
```

```
            # find the nearest training image to the i'th test image
```

```
            # using the L1 distance (sum of absolute value differences)
```

```
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
```

```
            min_index = np.argmin(distances) # get the index with smallest distance
```

```
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example
```

```
        return Ypred
```

Memorize training data

Nearest Neighbor classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

For every test image:

- Find the nearest train image
- Return the label of nearest training image

```

import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

```

Nearest Neighbor classifier

Q: how does the classification speed depend on the size of the training data?


```

import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

```

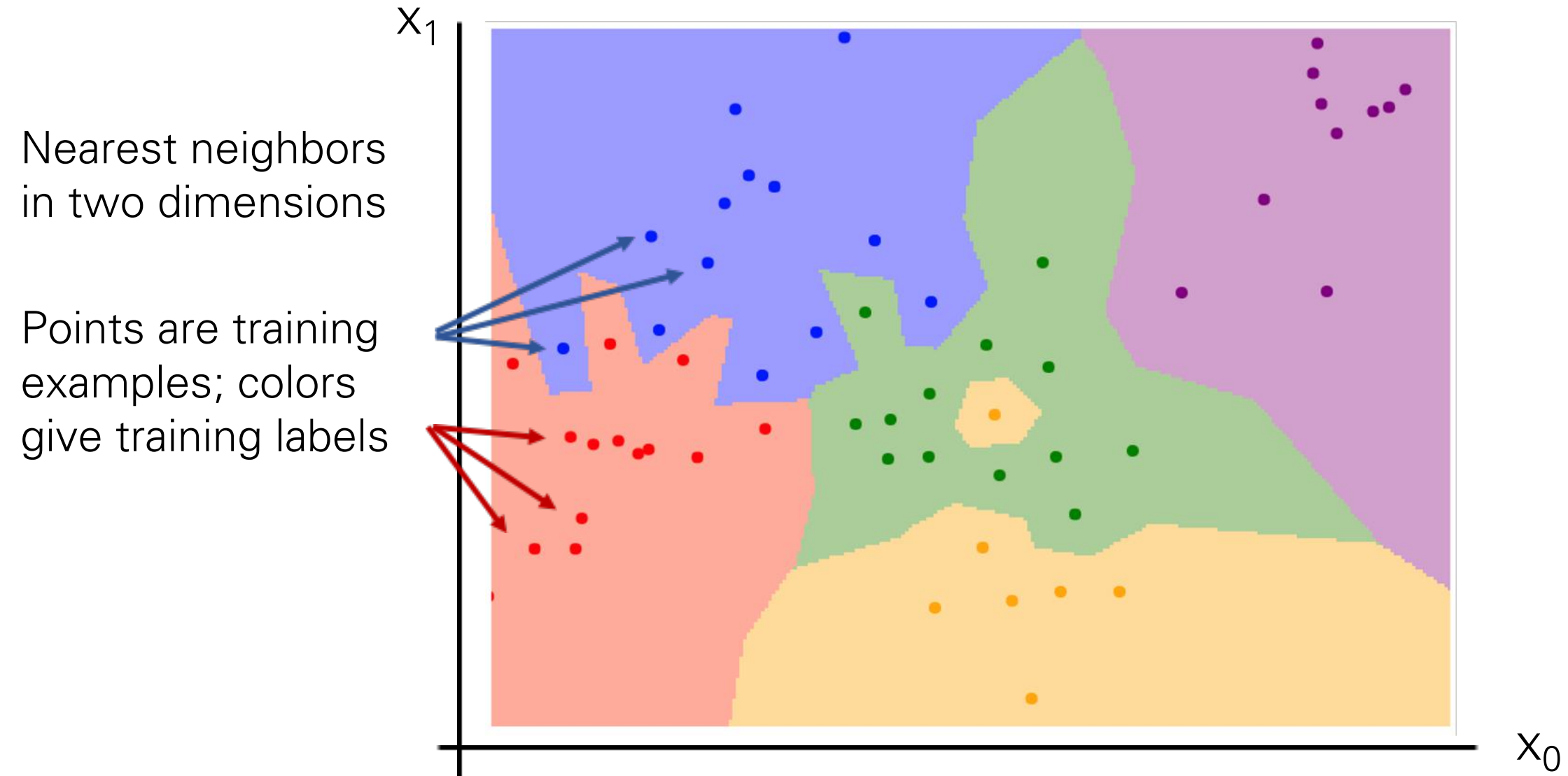
Nearest Neighbor classifier

Q: how does the classification speed depend on the size of the training data? **linearly** :(

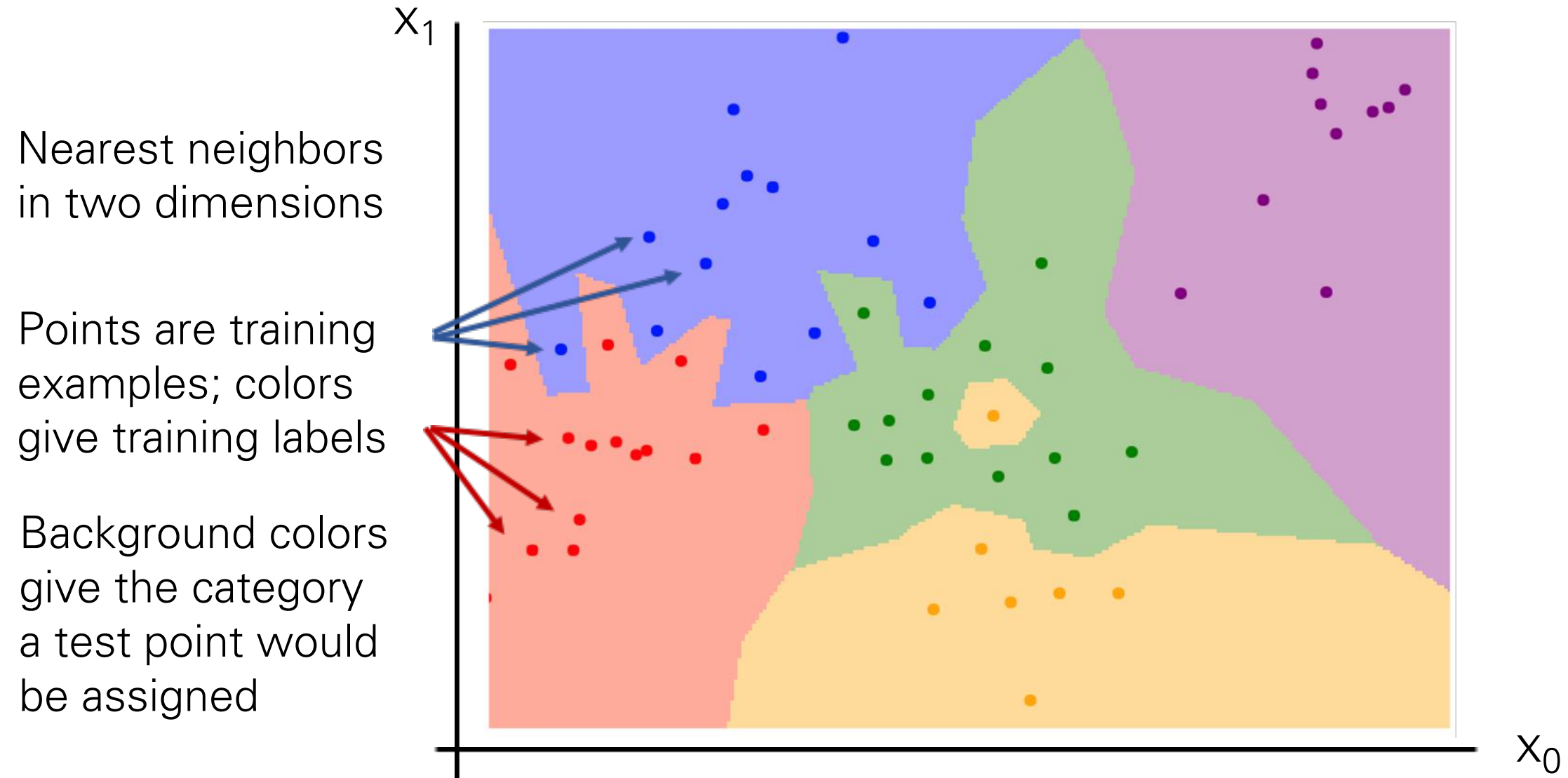
This is **backwards**:

- test time performance is usually much more important in practice.
- Deep Neural Networks flip this: expensive training, cheap test evaluation

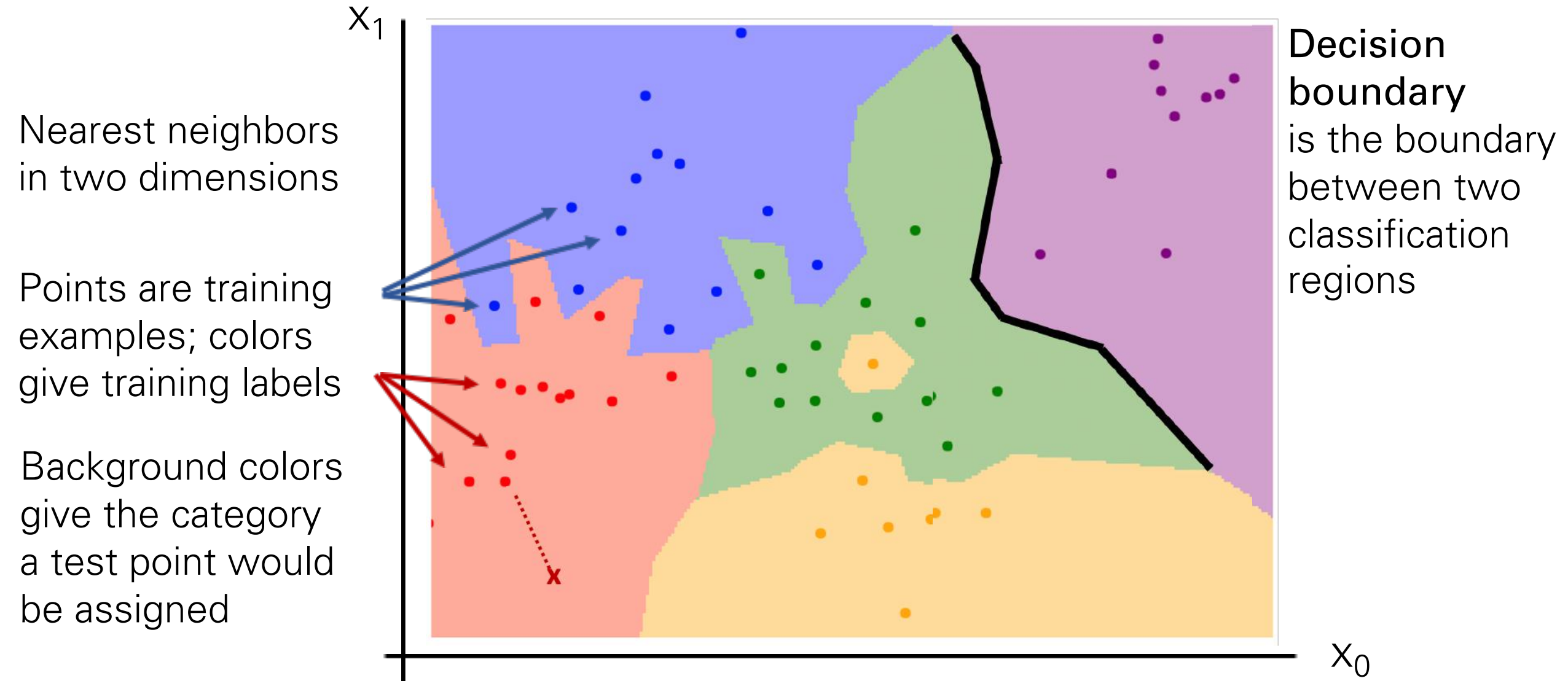
Nearest Neighbor Decision Boundaries



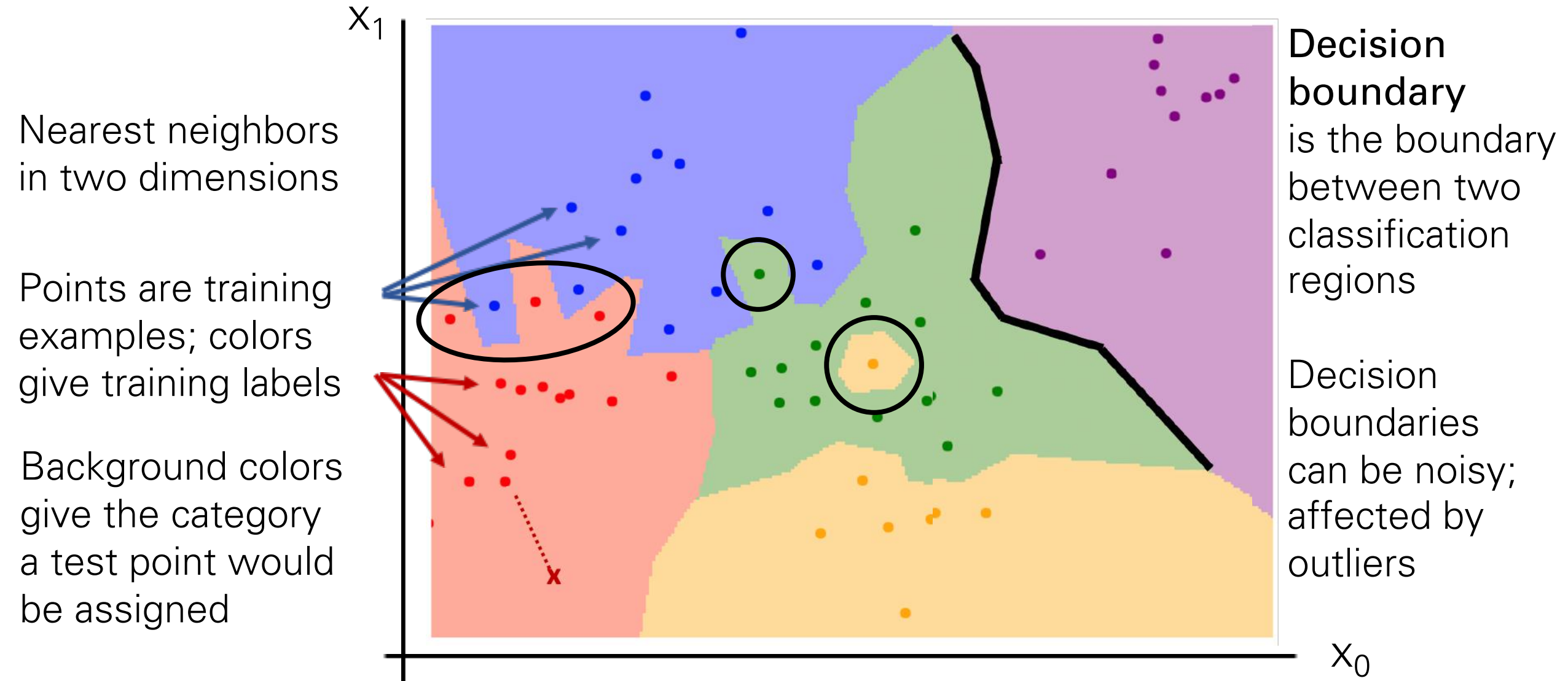
Nearest Neighbor Decision Boundaries



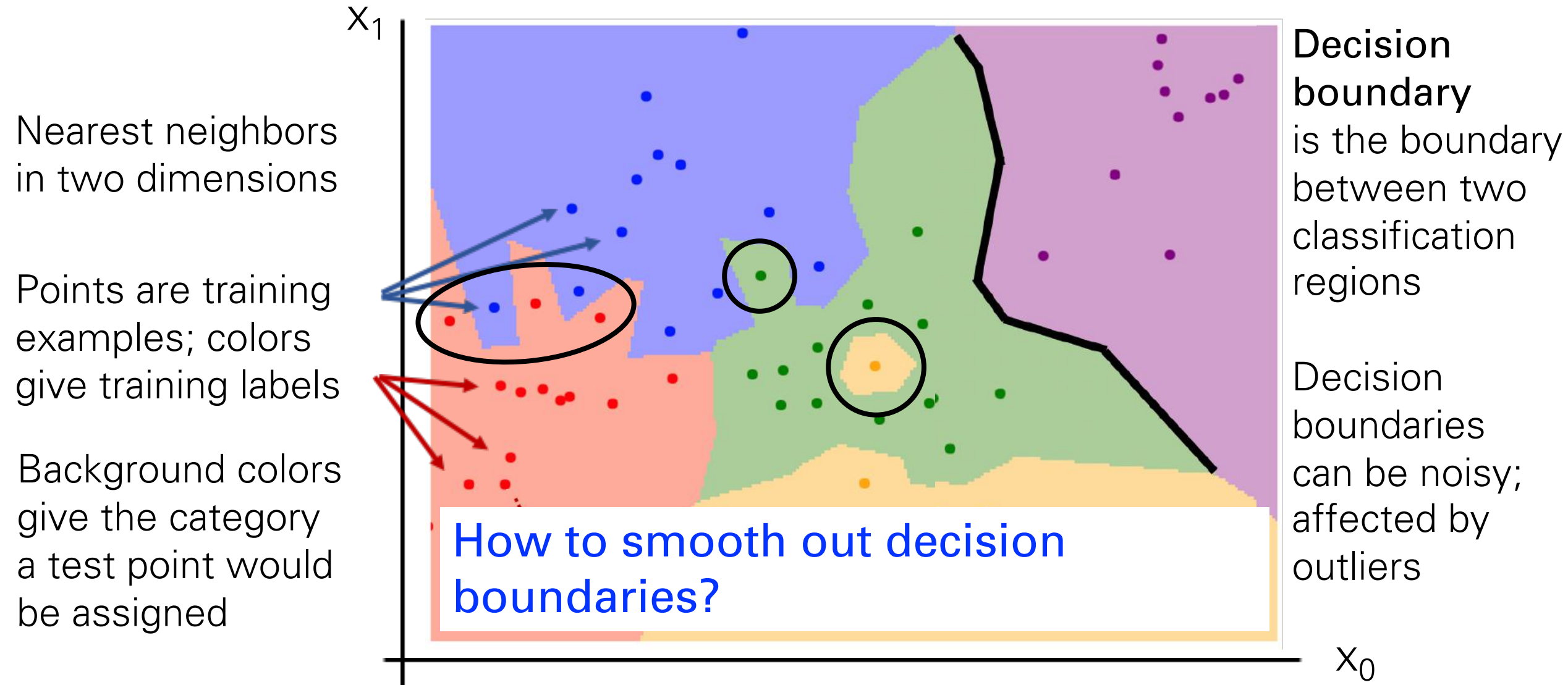
Nearest Neighbor Decision Boundaries



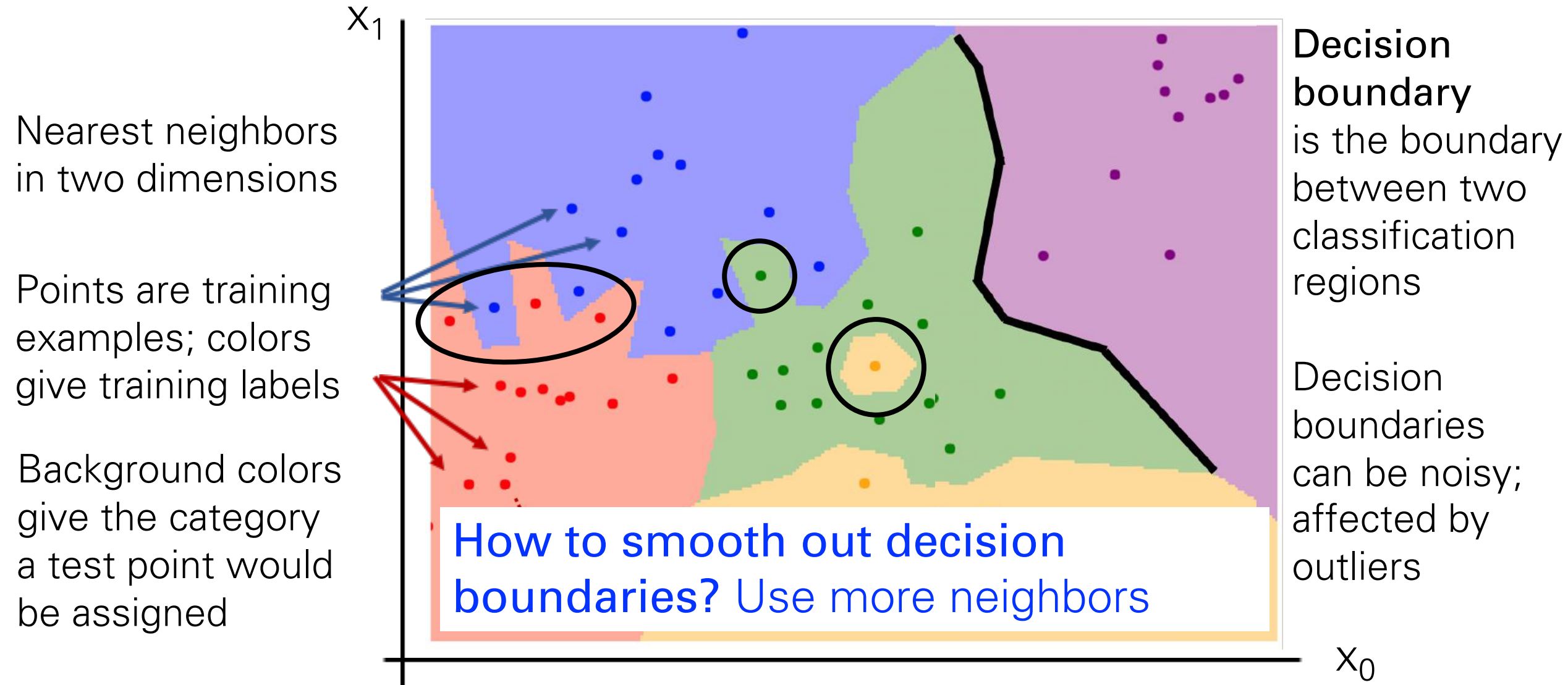
Nearest Neighbor Decision Boundaries



Nearest Neighbor Decision Boundaries



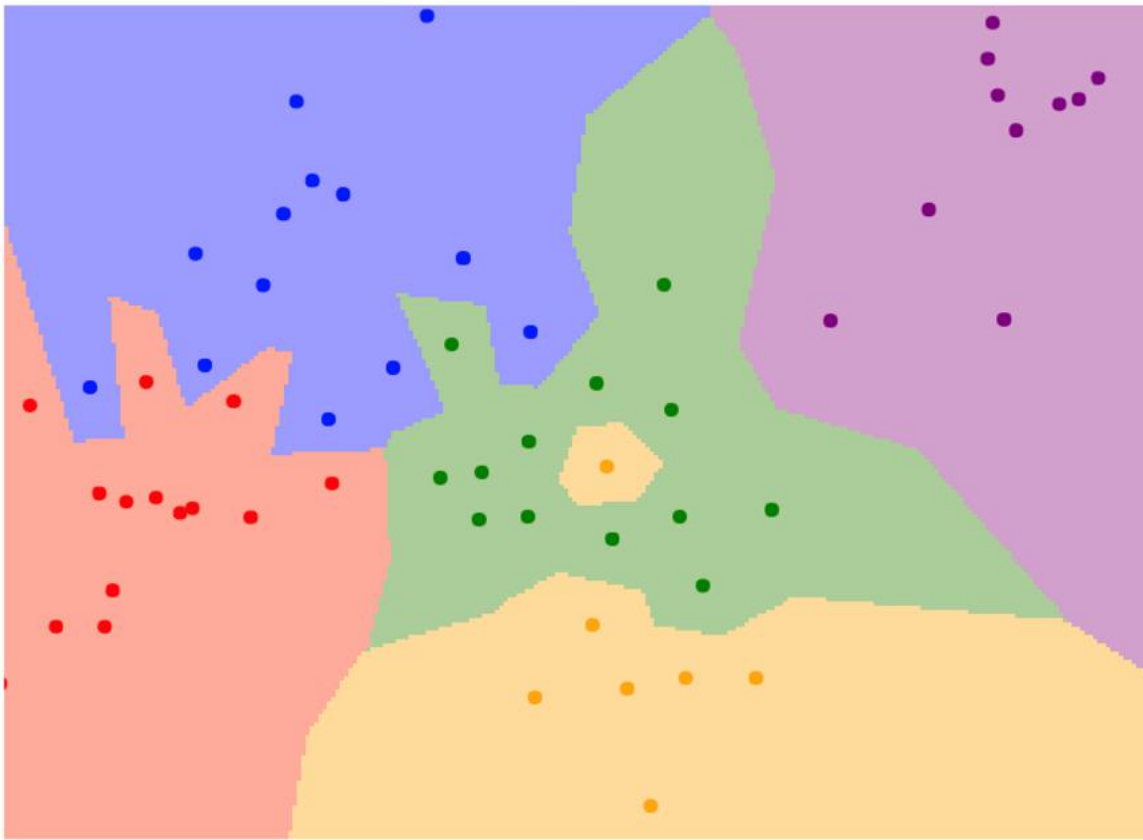
Nearest Neighbor Decision Boundaries



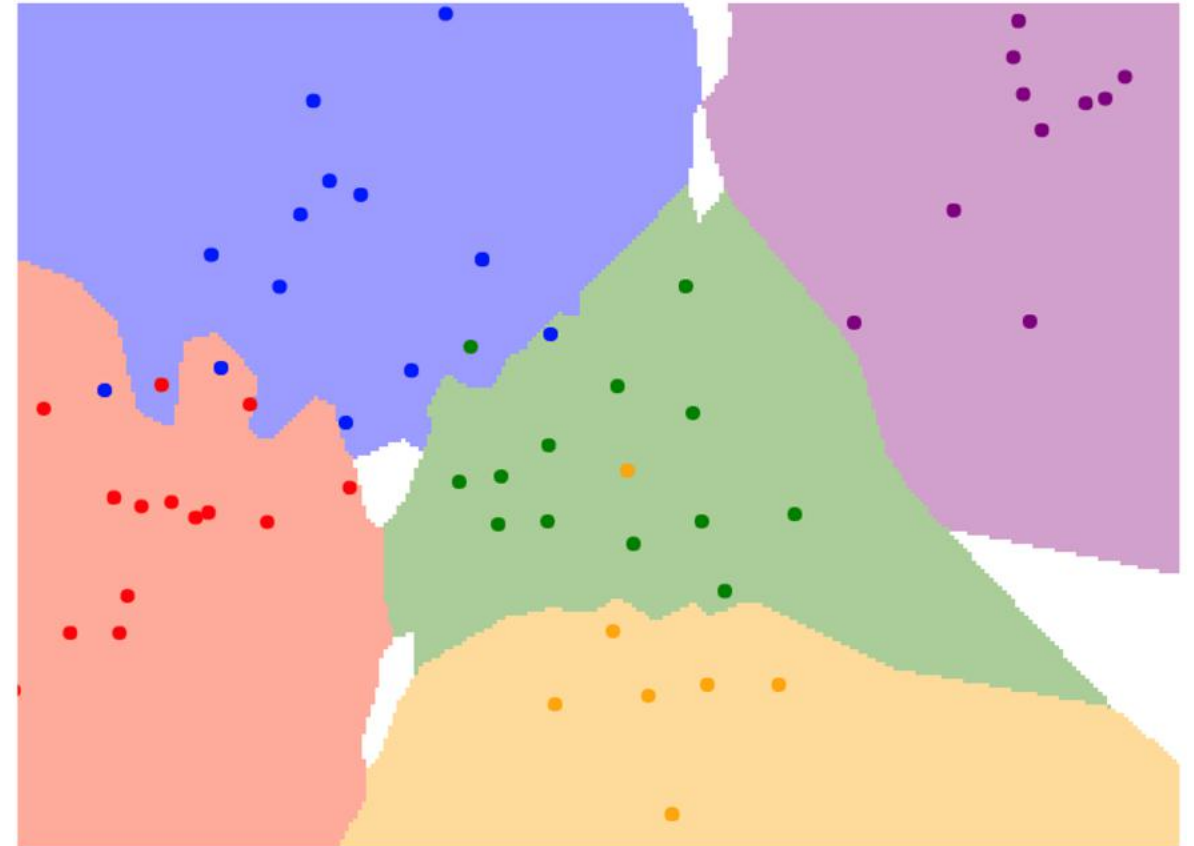
K-Nearest Neighbors

Instead of copying label from nearest neighbor, take **majority vote** from K closest points

$K = 1$



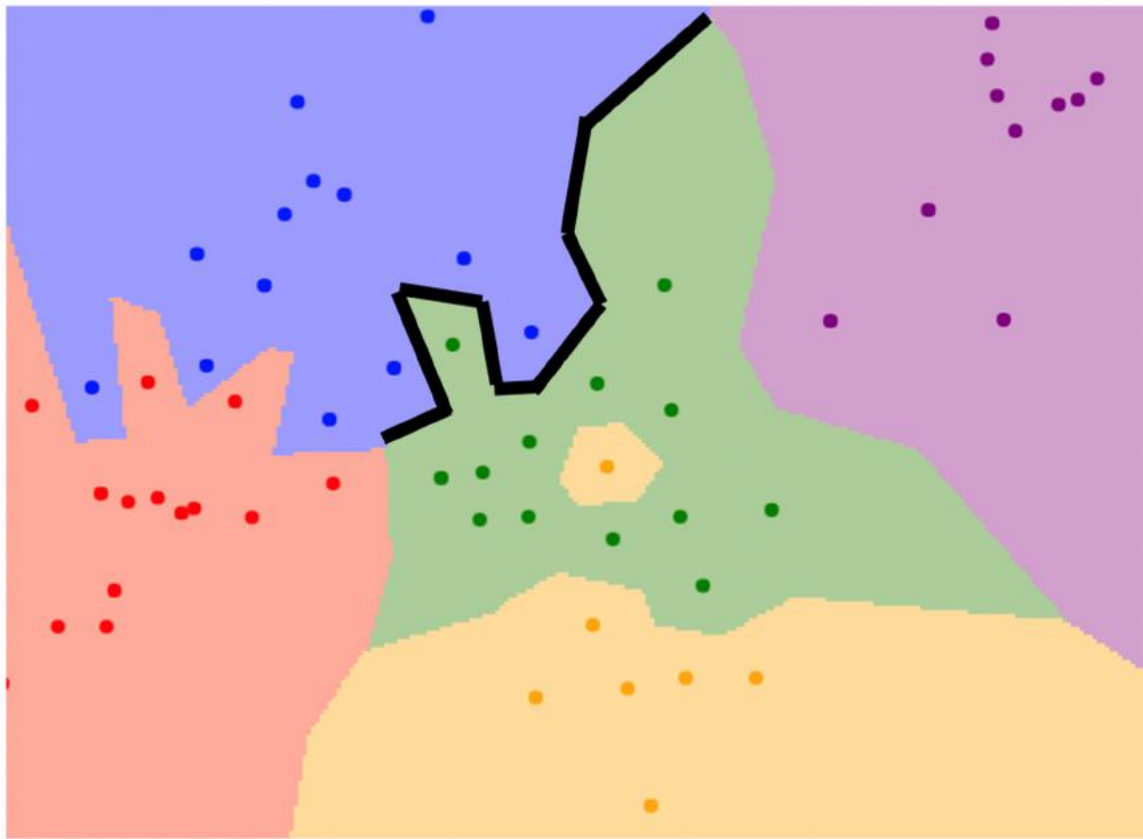
$K = 3$



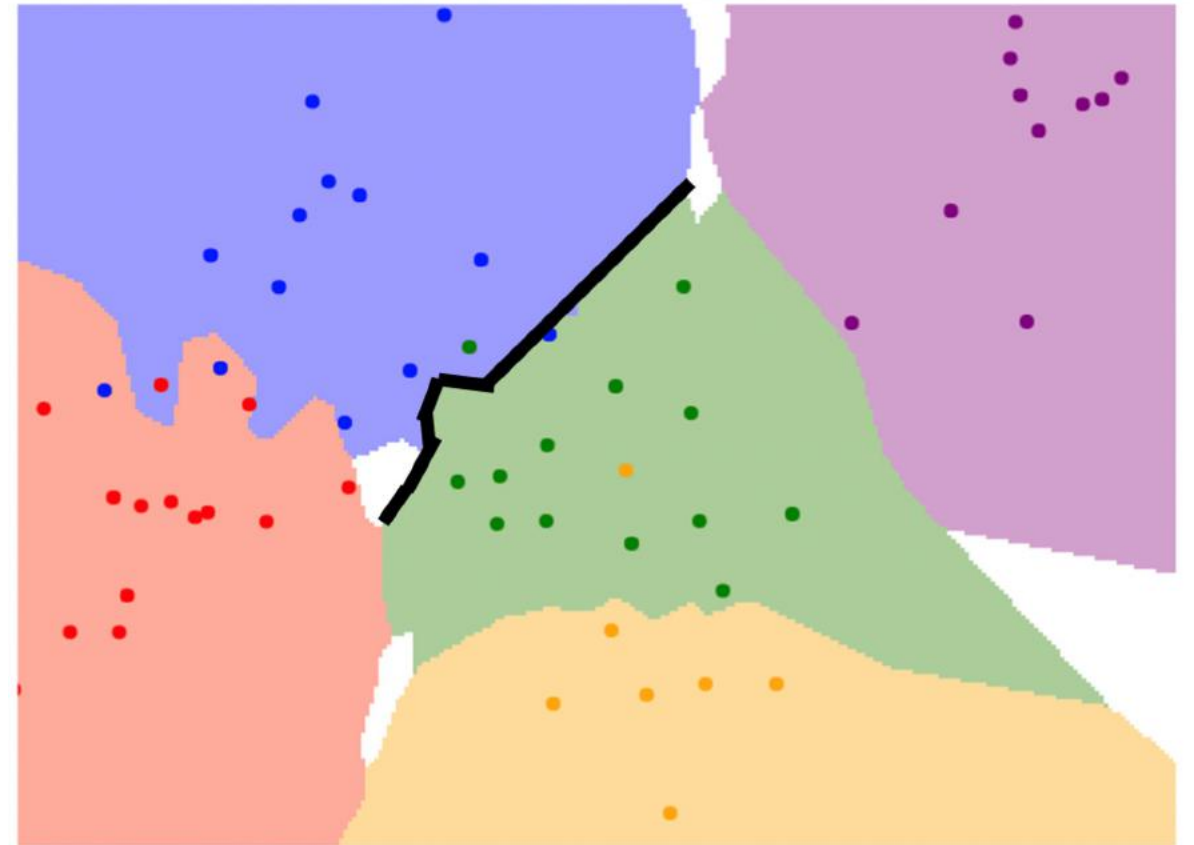
K-Nearest Neighbors

Using more neighbors helps
smooth out rough decision
boundaries

$K = 1$



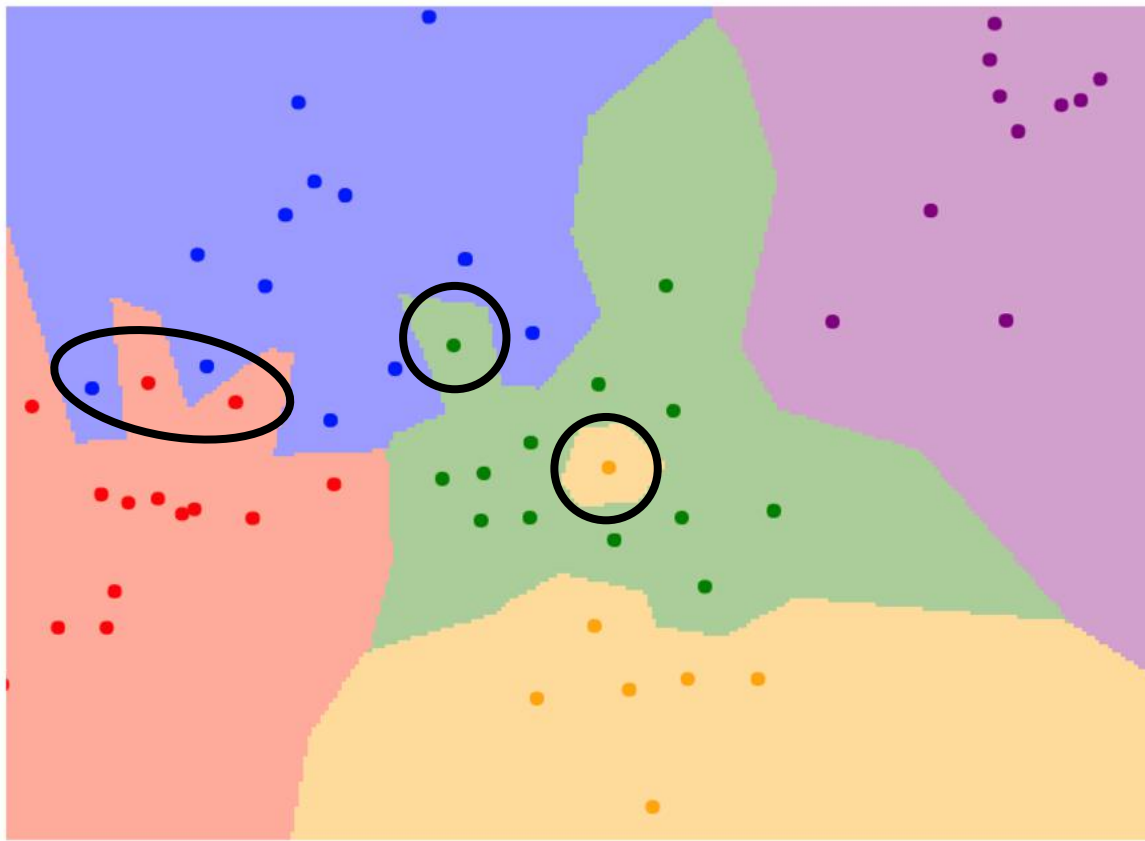
$K = 3$



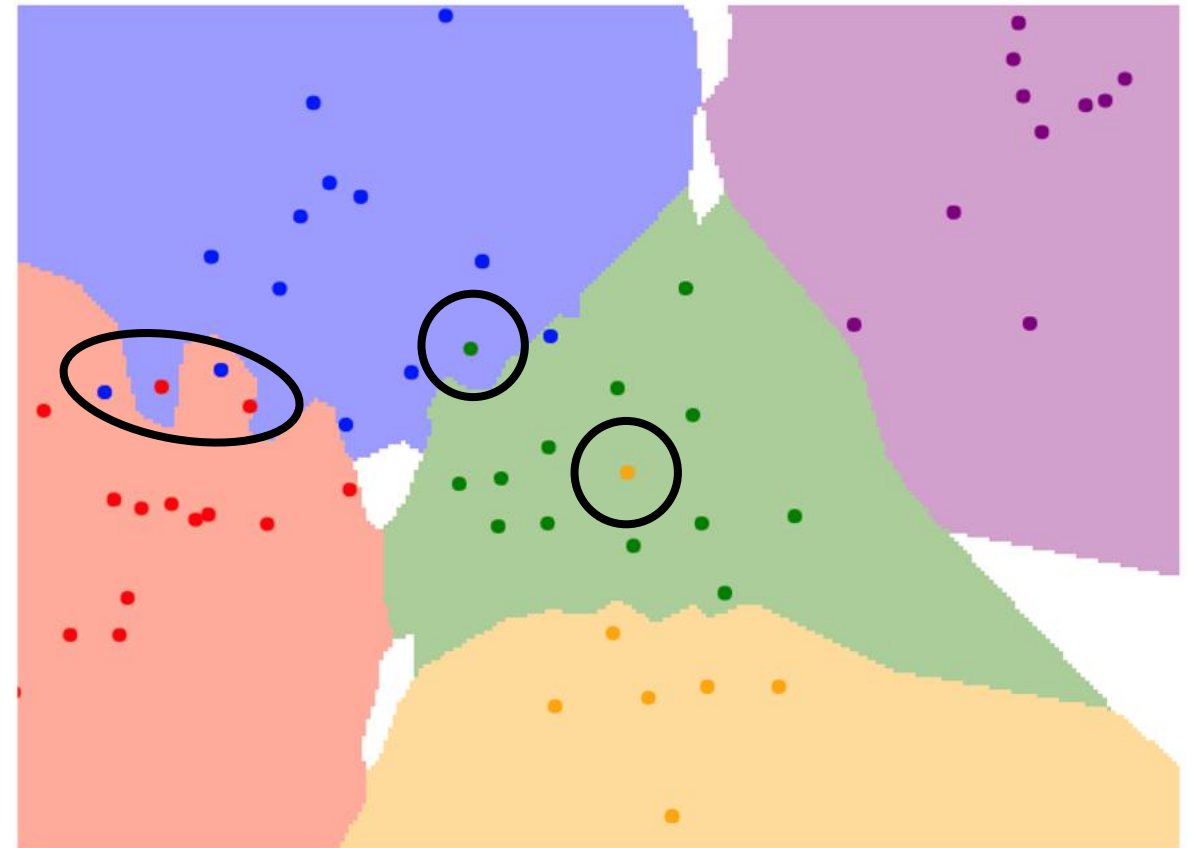
K-Nearest Neighbors

Using more neighbors helps
reduce the effect of outliers

$K = 1$



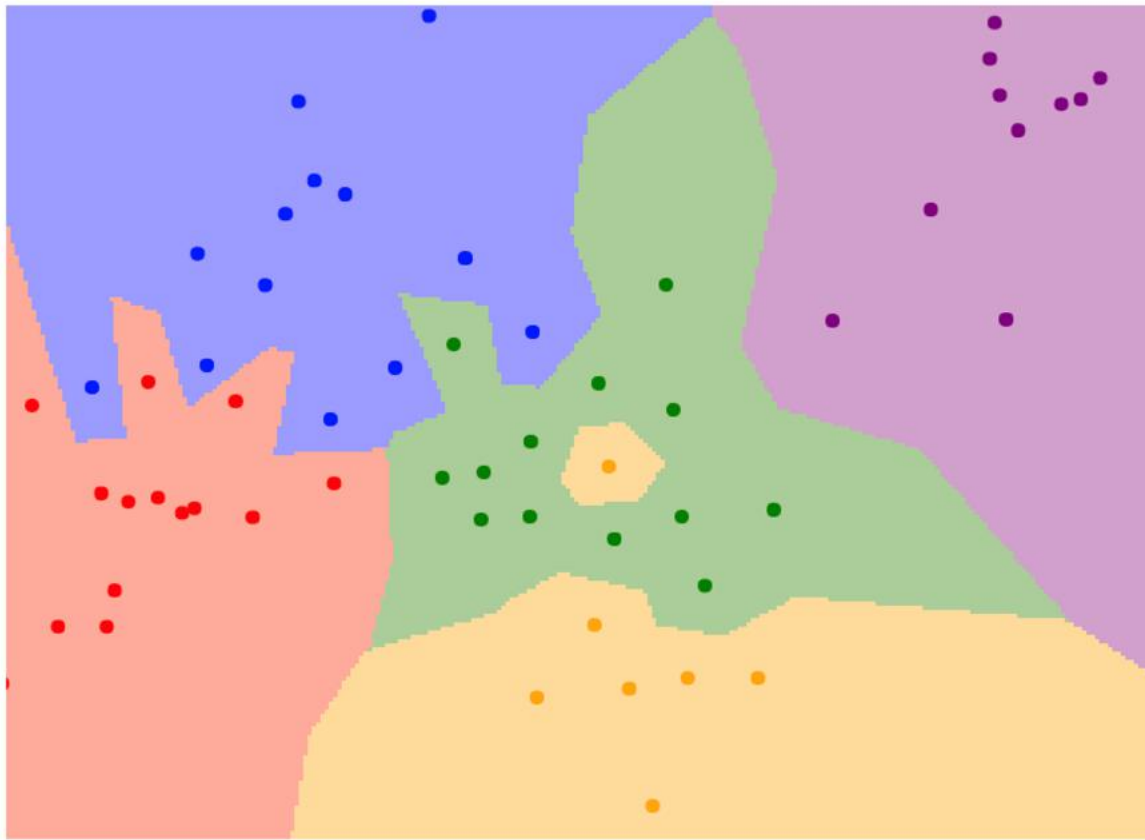
$K = 3$



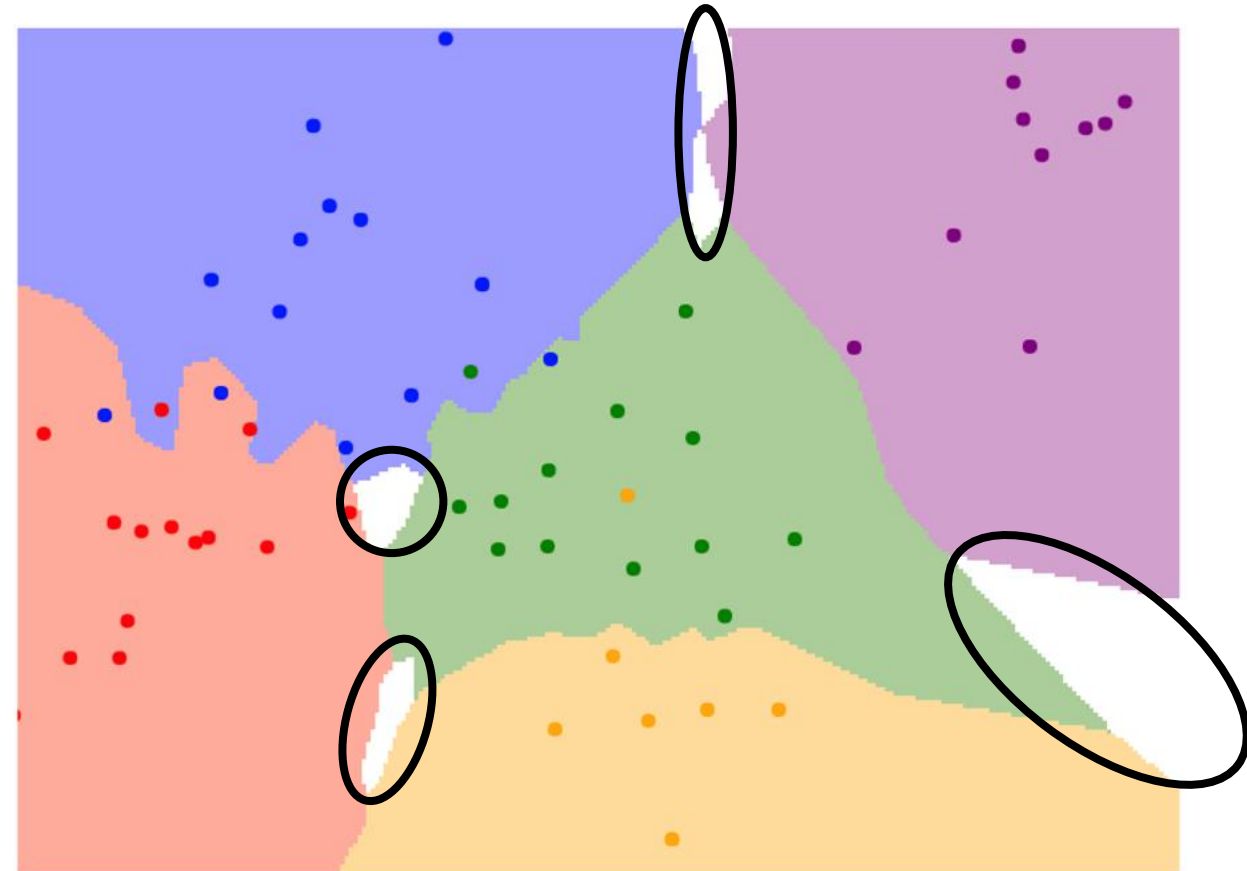
K-Nearest Neighbors

When $K > 1$ there can be ties between classes.
Need to break somehow!

$K = 1$



$K = 3$



How do we compare the images? What is the **distance metric**?

L1 distance: $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

add → 456

The choice of distance is a **hyperparameter**
common choices:

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

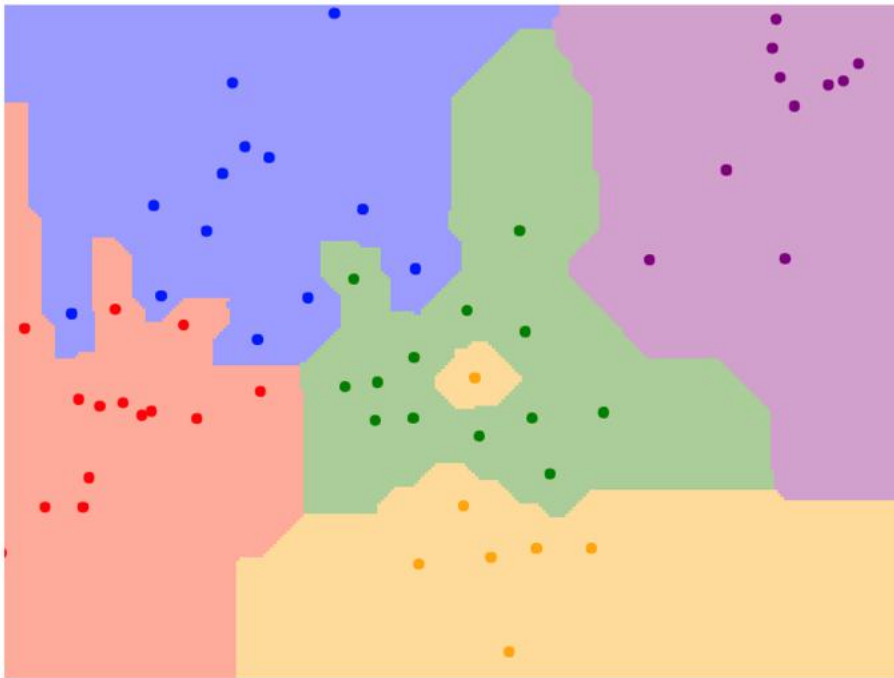
K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

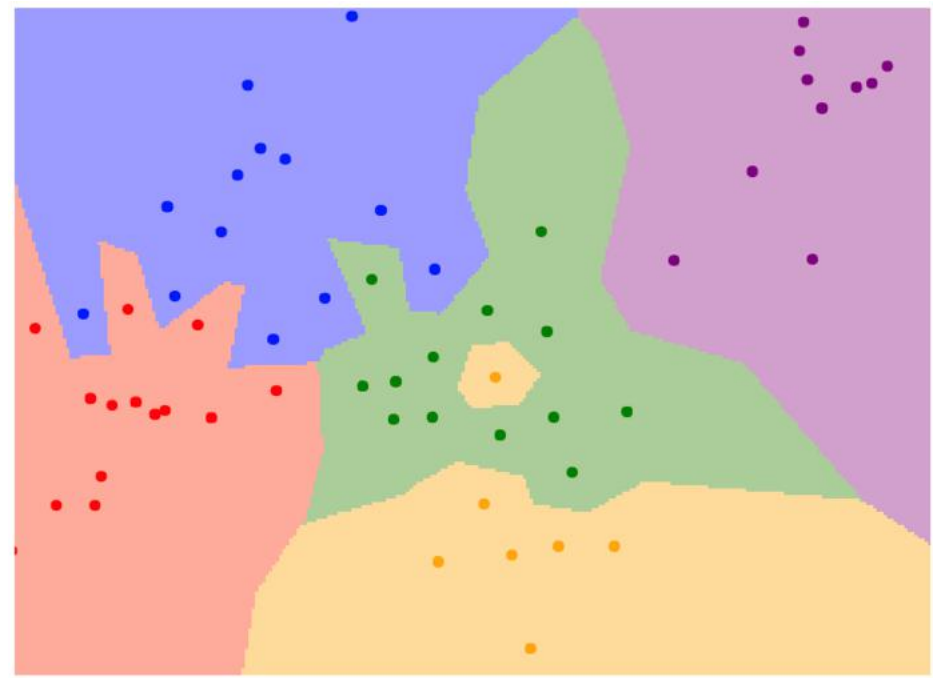
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K = 1

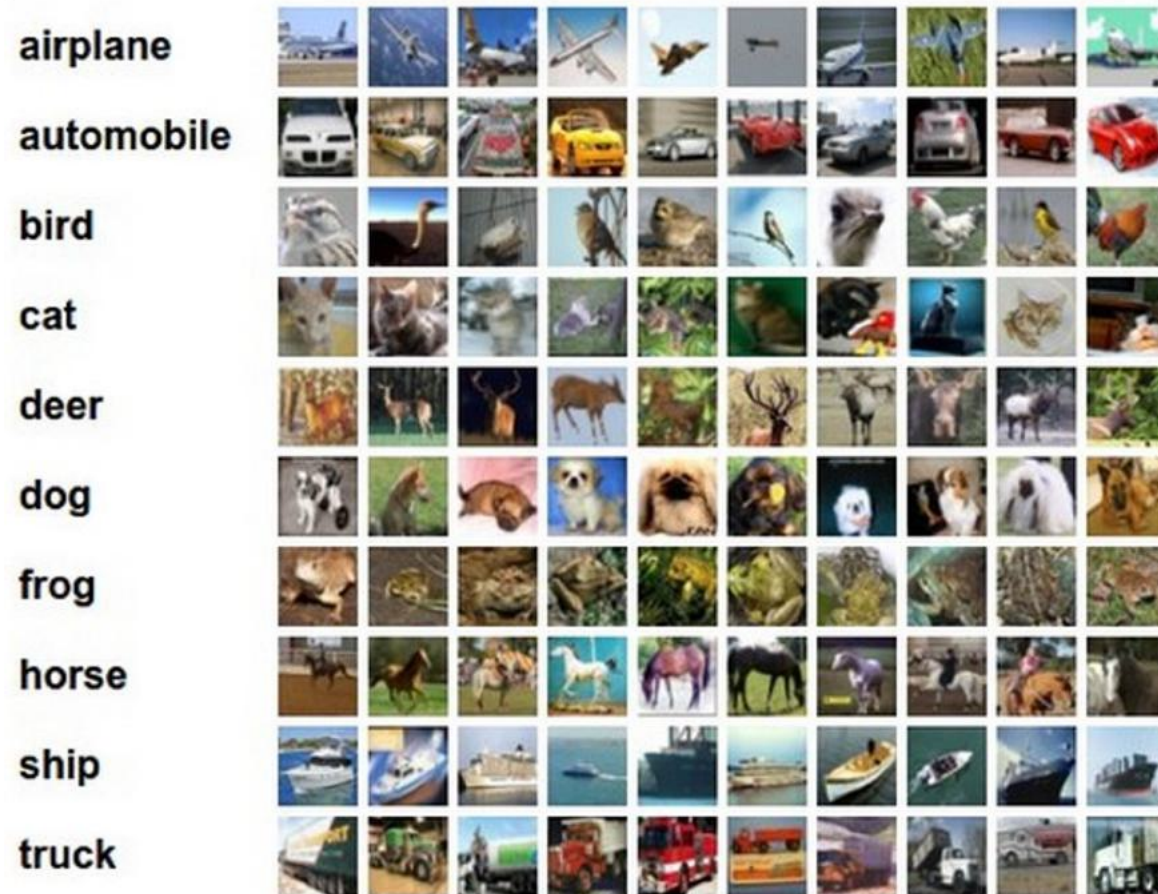


Example dataset: **CIFAR-10**

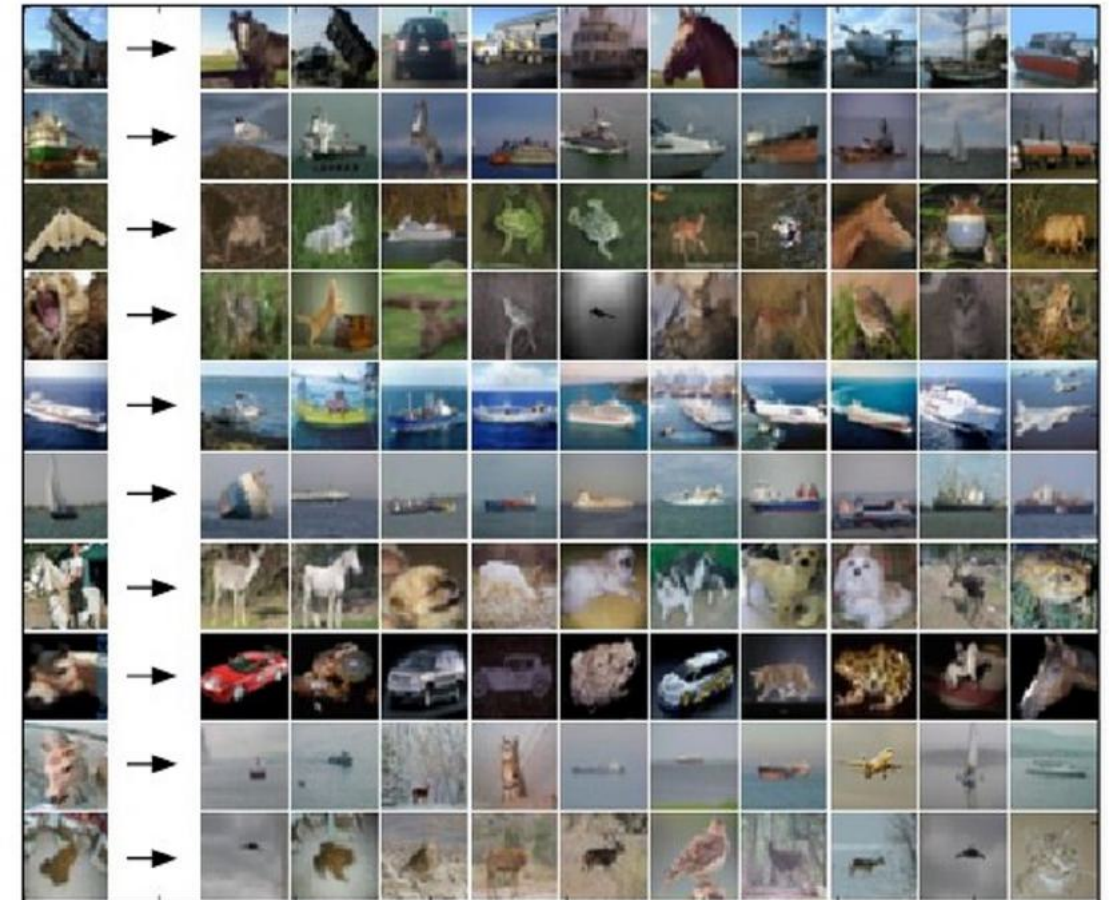
10 labels

50,000 training images

10,000 test images.



For every test image (first column),
examples of nearest neighbors in rows



What is the best **distance** to use?

What is the best value of **k** to use?

i.e. how do we set the **hyperparameters**?

What is the best **distance** to use?

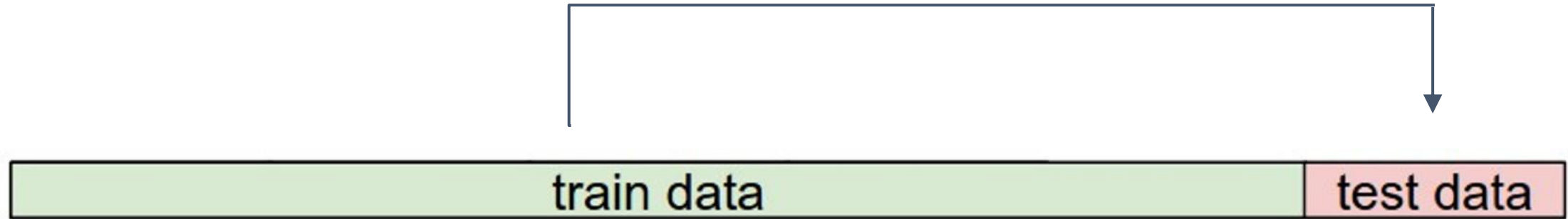
What is the best value of **k** to use?

i.e. how do we set the **hyperparameters**?

Very problem-dependent.

Must try them all out and see what works best.

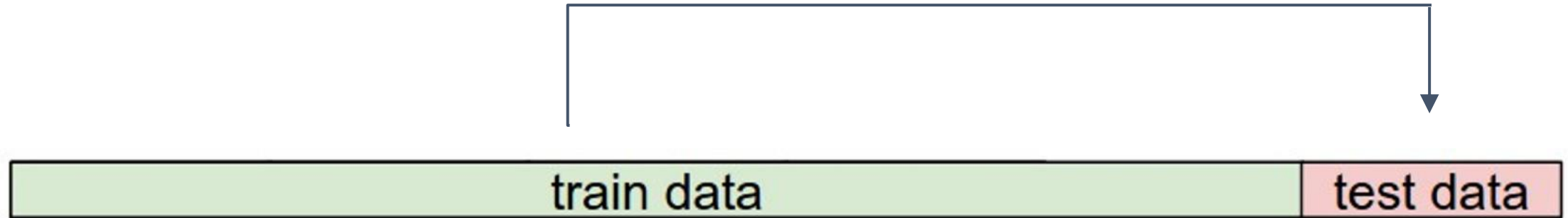
Try out what hyperparameters work best on test set.

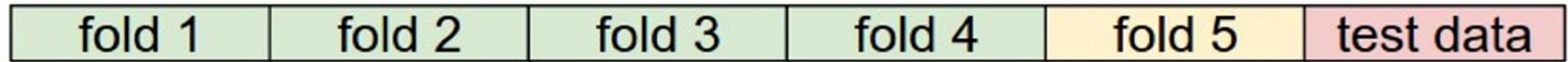
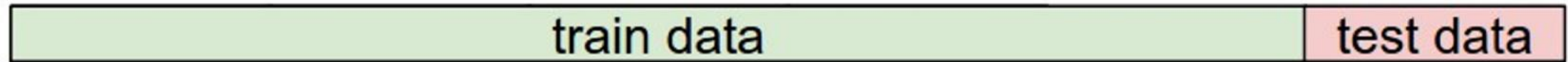


Trying out what hyperparameters work best on test set:

Very bad idea. The test set is a proxy for the generalization performance!

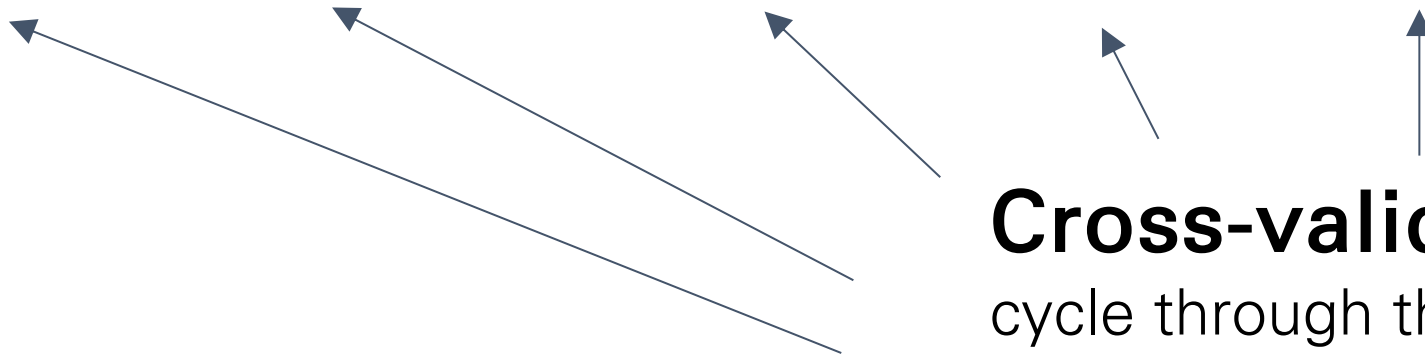
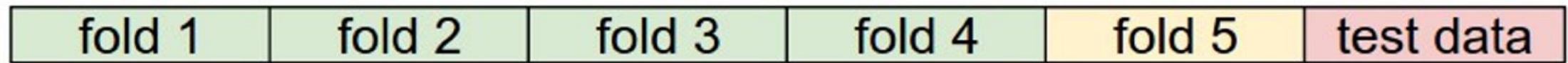
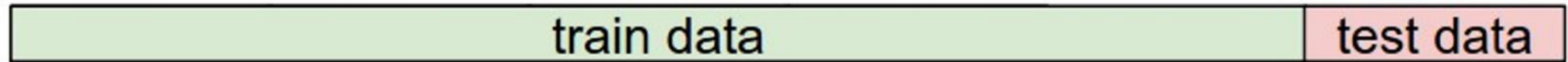
Use only **VERY SPARINGLY**, at the end.





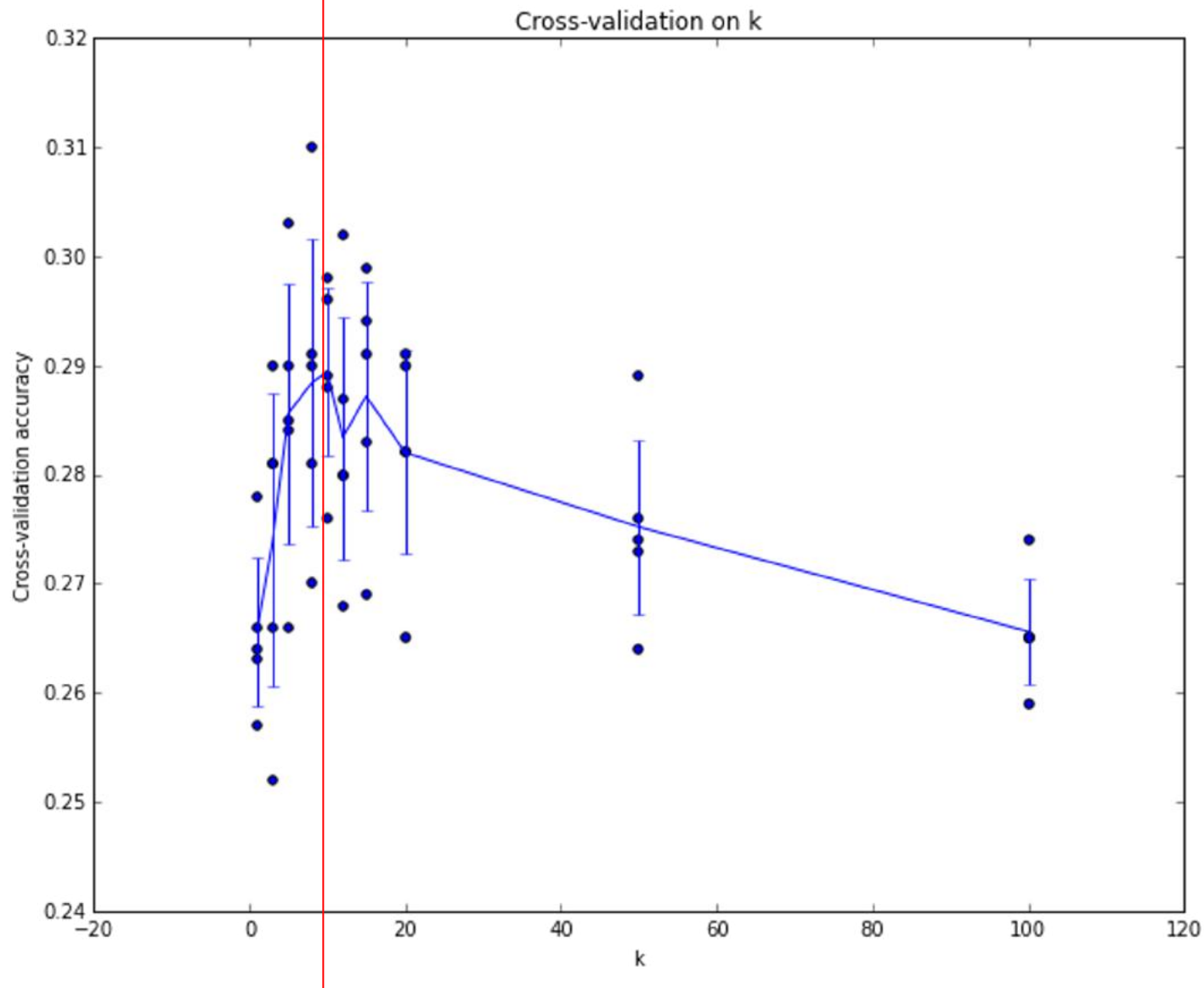
Validation data

use to tune hyperparameters



Cross-validation

cycle through the choice of which fold is the validation fold, average results.



Example of
5-fold cross-validation
for the value of **k**.

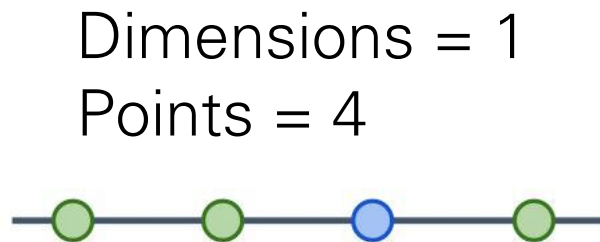
Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

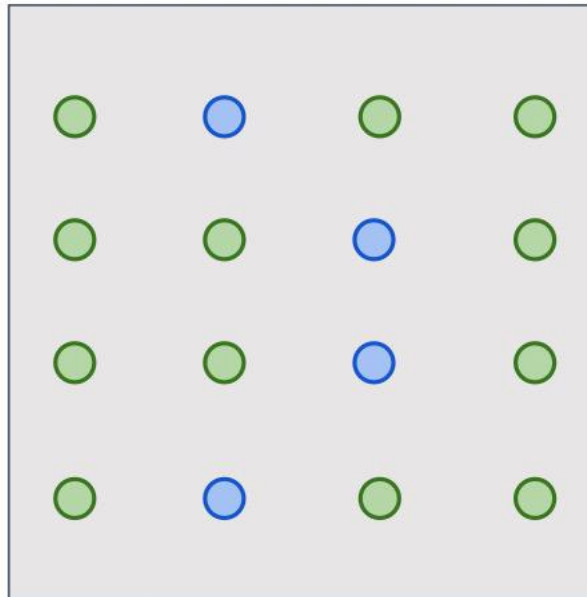
(Seems that $k \approx 7$ works
the best for this data)

Problem: Curse of Dimensionality

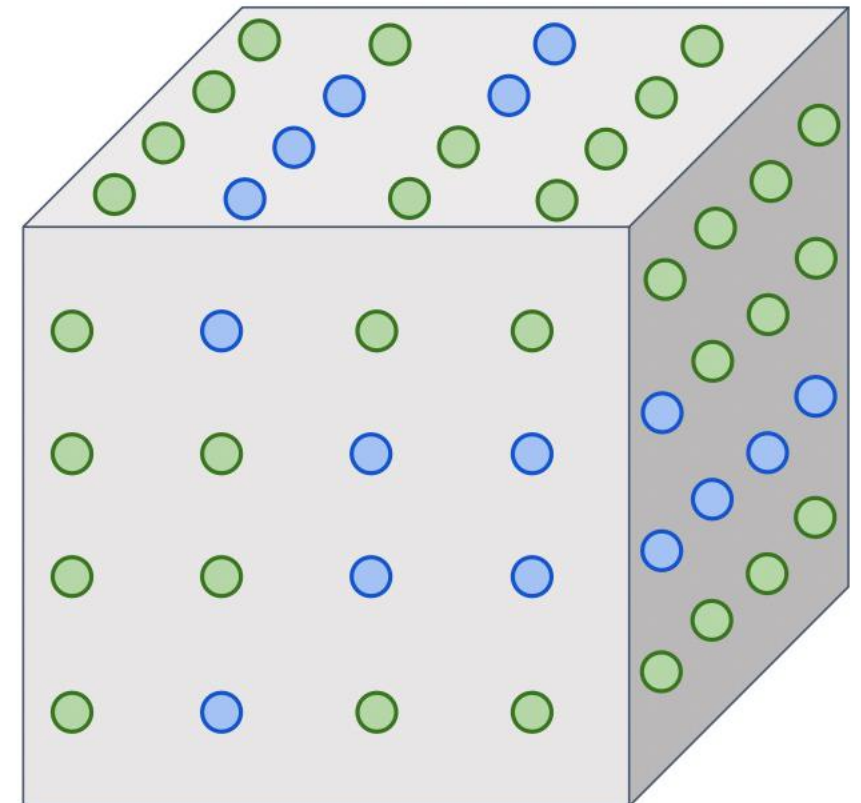
- **Curse of dimensionality:** For uniform coverage of space, number of training points needed grows exponentially with dimension



Dimensions = 3
Points = 4^3



Dimensions = 3
Points = 4^3



Problem: Curse of Dimensionality

- **Curse of dimensionality:** For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible
32x32 binary images:

$$2^{32 \times 32} \approx 10^{308}$$

Problem: Curse of Dimensionality

- **Curse of dimensionality:** For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible
32x32 binary images:

$$2^{32 \times 32} \approx 10^{308}$$

Number of elementary particles
in the visible universe: [\(source\)](#)

$$\approx 10^{97}$$

k-Nearest Neighbor on images **never** used.

- Very slow at test time
- Distance metrics on pixels are not informative

Original



Boxed



Shifted



Tinted



(all 3 images have same L2 distance to the one on the left)

Nearest Neighbor with ConvNet features works well



Devlin et al, "Exploring Nearest Neighbor Approaches for Image Captioning", 2015

Nearest Neighbor with ConvNet features works well

Example: Image Captioning with Nearest Neighbor



A bedroom with a bed and a couch.



A cat sitting in a bathroom sink.



A train is stopped at a train station.



A wooden bench in front of a building.

The learning problem

- linear classification
- hypothesis class, estimation algorithm
- loss and estimation criterion
- sampling, empirical and expected losses

The Learning Problem

Digit Recognition

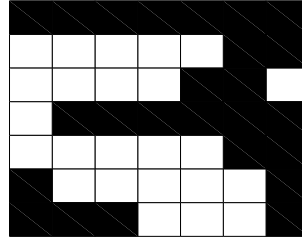


Image Classification

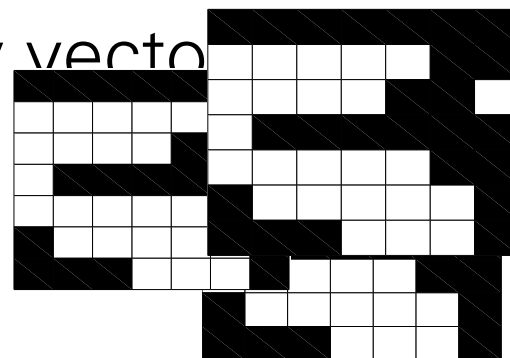


- Steps
 - entertain a (biased) set of possibilities (hypothesis class)
 - adjust predictions based on available examples (estimation)
 - rethink the set of possibilities (model selection)
- Principles of learning are “universal”
 - society (e.g., scientific community)
 - animal (e.g., human)
 - machine

Hypothesis class

- Representation: examples are binary vectors with $d = 64$

$$\mathbf{x} = [111 \dots 0001]^T$$



and labels $y \in \{-1, 1\}$ ("no", "yes")

- The mapping from examples to labels is a "**linear classifier**"

$$\hat{y} = \text{sign}(\theta \cdot \mathbf{x}) = \text{sign}(\theta_1 x_1 + \dots + \theta_d x_d)$$

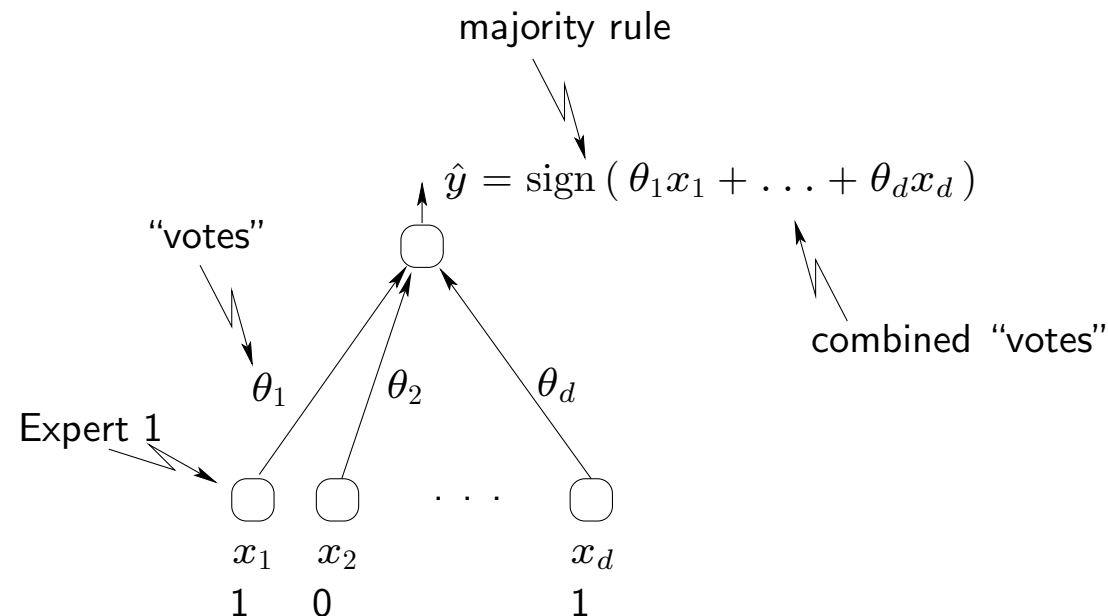
where θ is a vector of **parameters** we have to learn from examples.

Linear classifier/experts

- We can understand the simple linear classifier

$$\hat{y} = \text{sign} (\theta \cdot \mathbf{x}) = \text{sign} (\theta_1 x_1 + \dots + \theta_d x_d)$$

as a way of combining expert opinion (in this case simple binary features)



Estimation

$\mathbf{x}$	y
0111111001110010000000100000001001111110111011111001110111110001	+1
0001111100000011000001110000011001111110111111001111111100000011	+1
1111111000000110000011000111111000000111100000111110001101111111	-1
...	...

- How do we adjust the parameters θ based on the labeled examples?

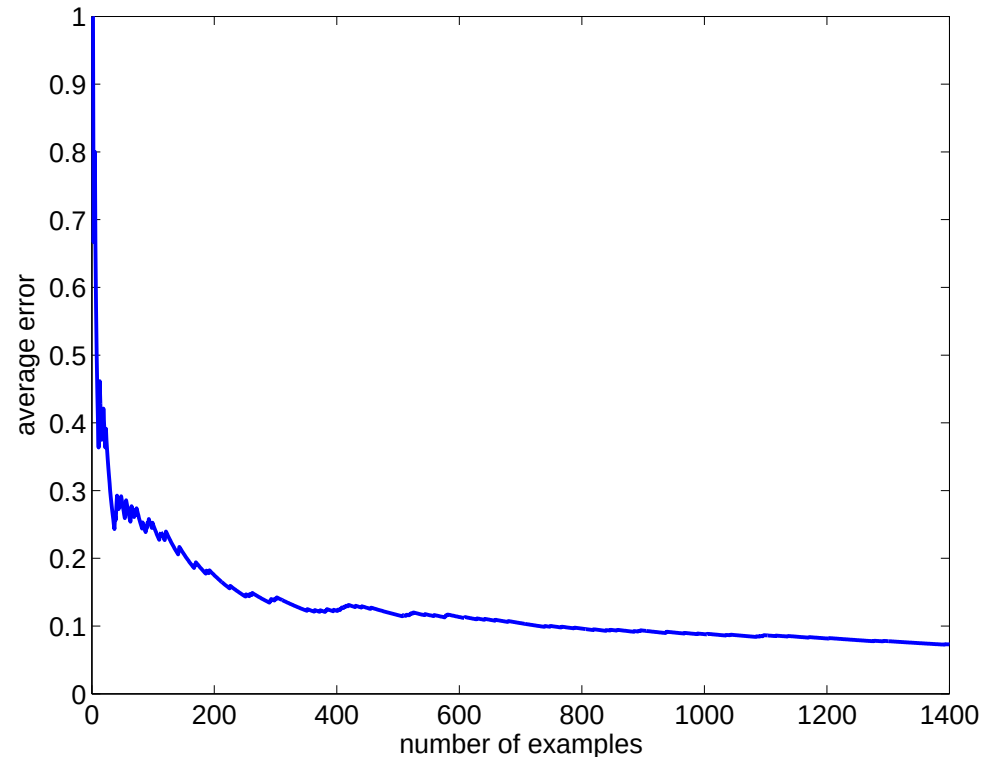
$$\hat{y} = \text{sign}(\theta \cdot \mathbf{x})$$

For example, we can simply refine/update the parameters whenever we make a mistake (**perceptron algorithm**):

$$\theta_i \leftarrow \theta_i + y x_i, \quad i = 1, \dots, d \quad \text{if prediction was wrong}$$

Evaluation

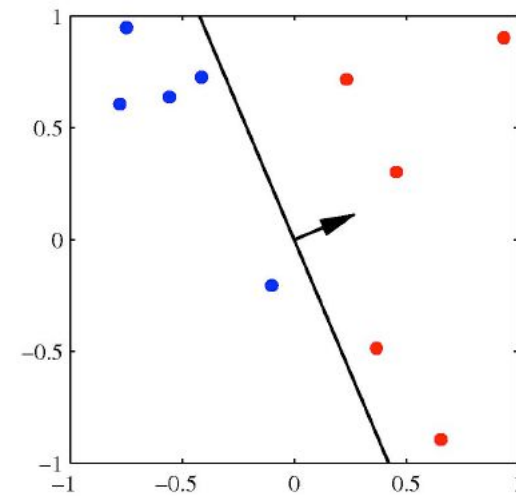
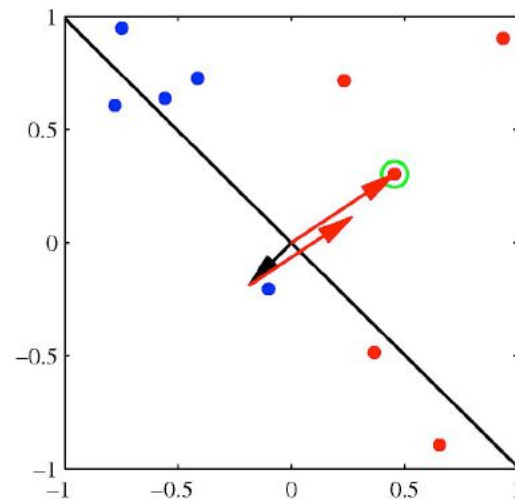
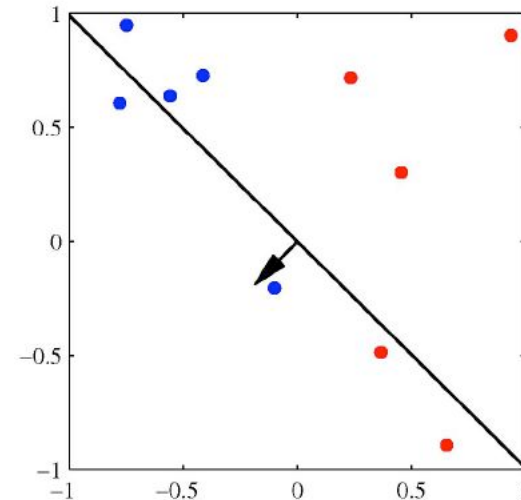
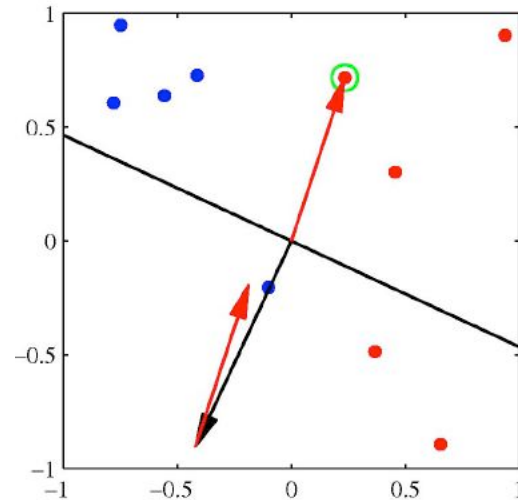
- Does the simple mistake driven algorithm work?



(average classification error as a function of the number of examples and labels seen so far)

Illustration of Convergence

- Convergence of the perceptron learning algorithm

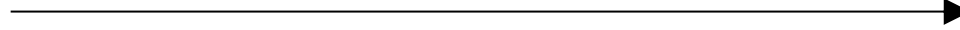


Linear classifier: image classification



image parameters

$$f(x, W)$$



10 numbers,
indicating class
scores

[32x32x3]

array of numbers 0...1
(3072 numbers total)

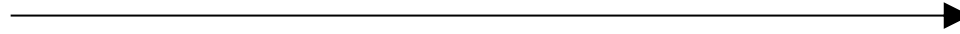
Linear classifier: image classification

$$f(x, W) = Wx$$



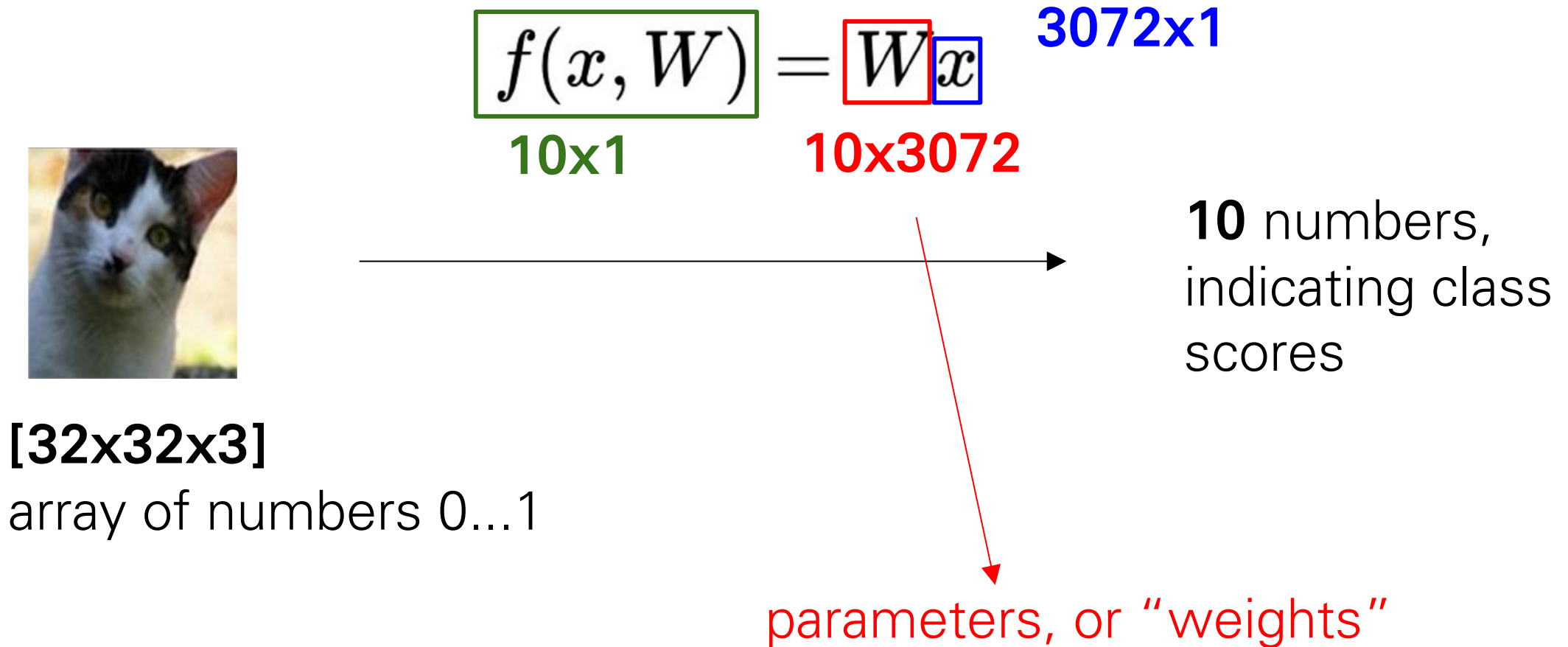
[32x32x3]

array of numbers 0...1

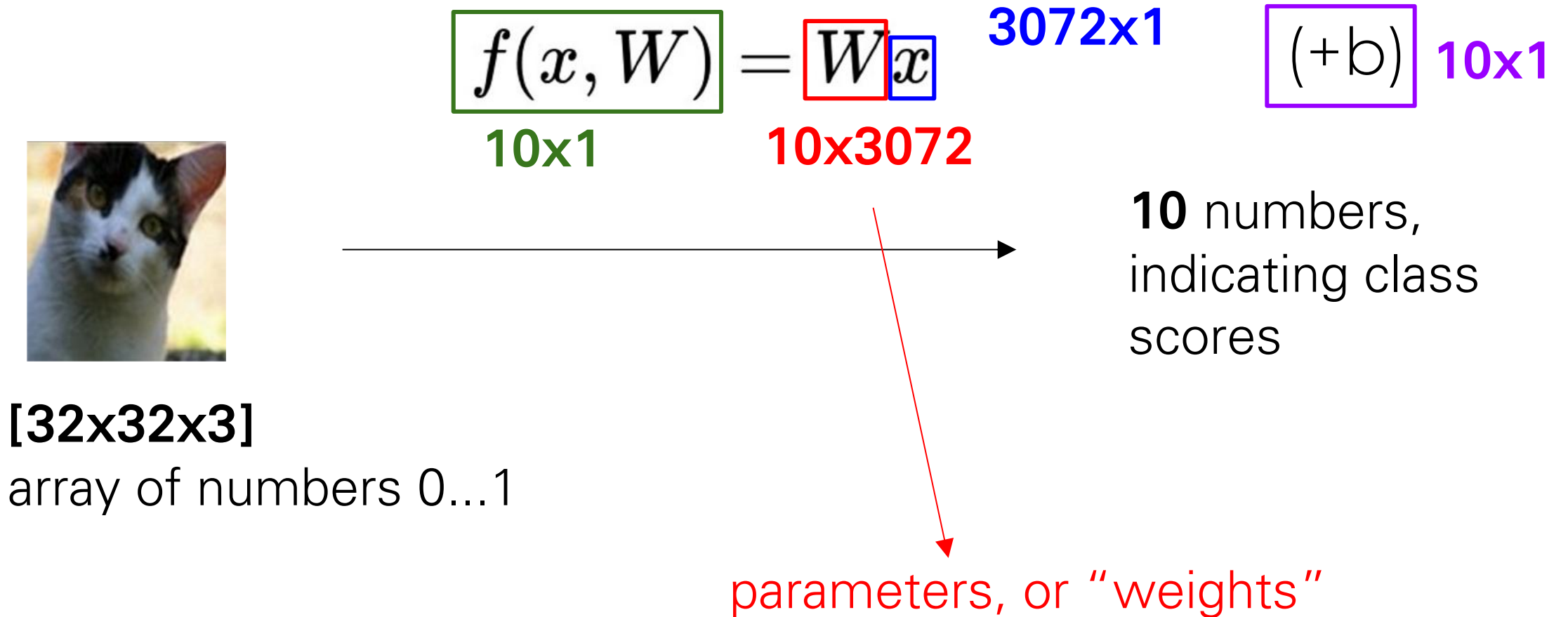


10 numbers,
indicating class
scores

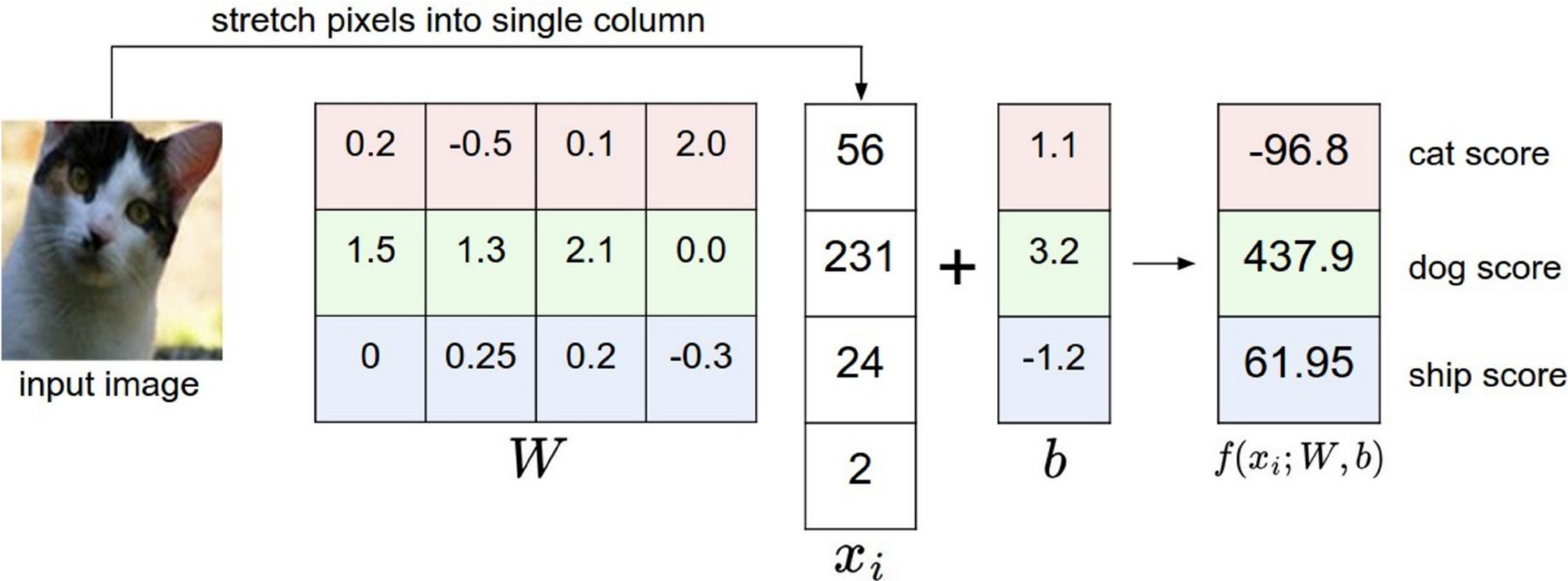
Linear classifier: image classification



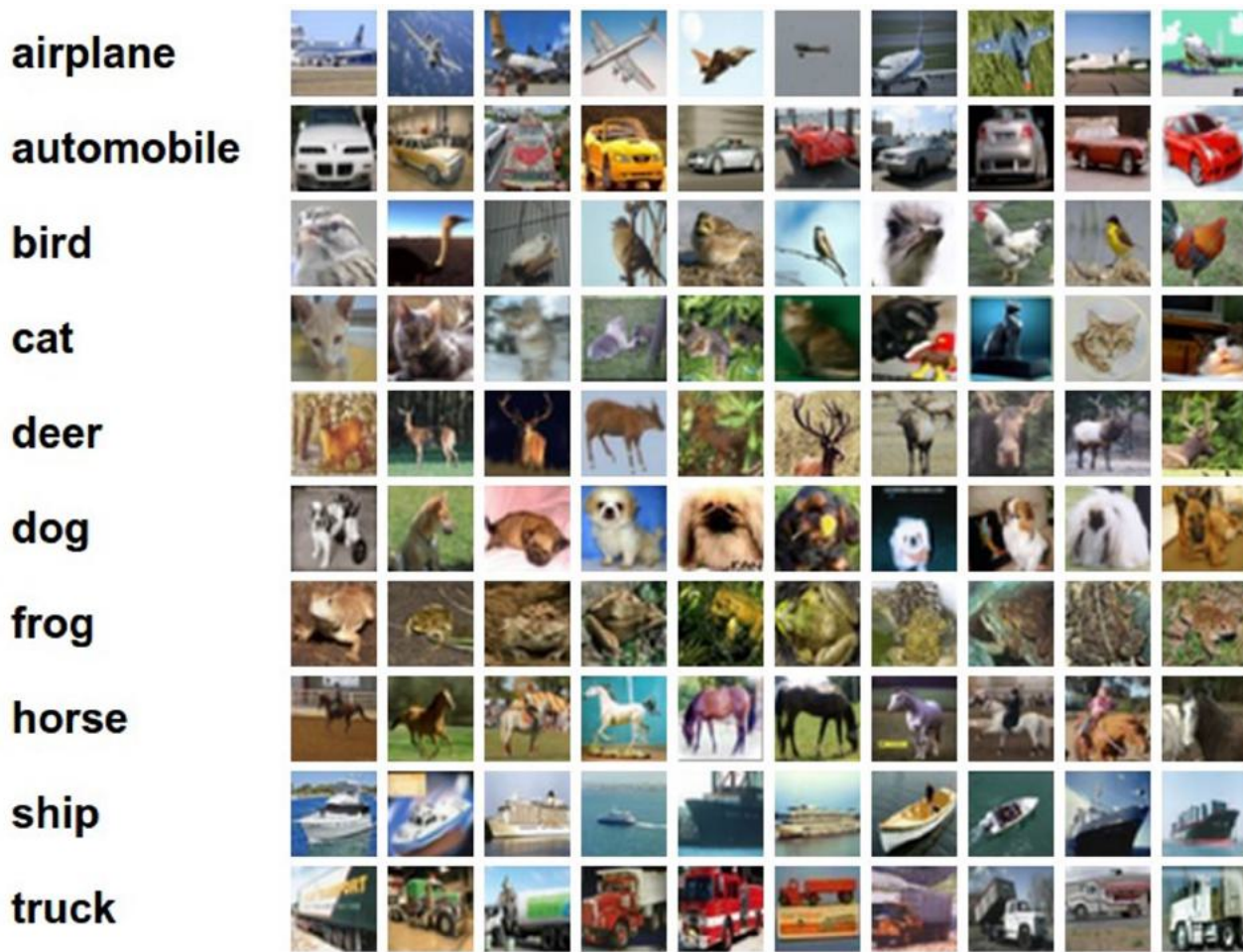
Linear classifier: image classification



Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



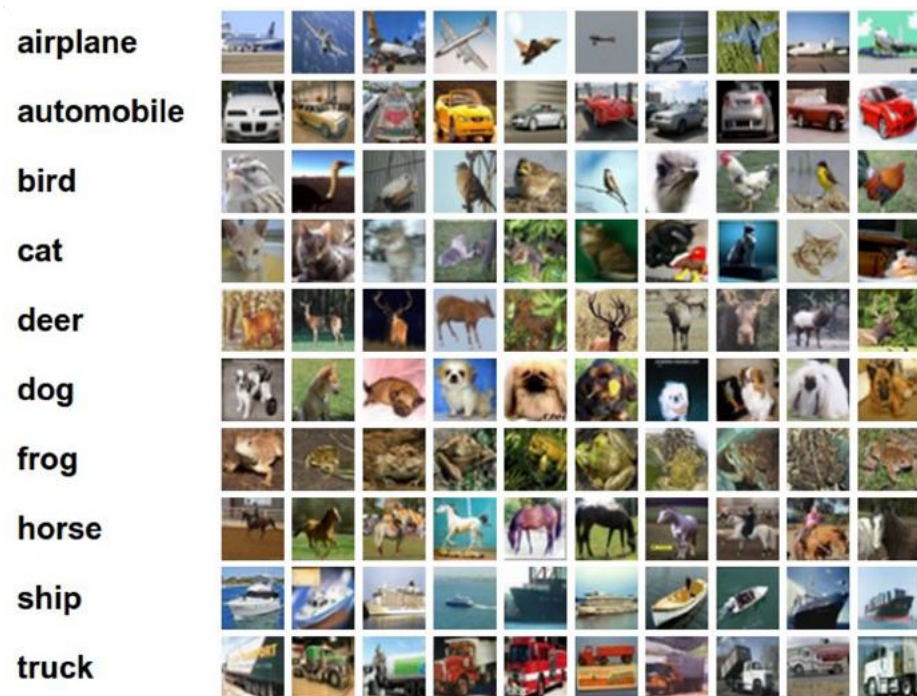
Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

Q: what does the linear classifier do, in plain English?

Interpreting a Linear Classifier

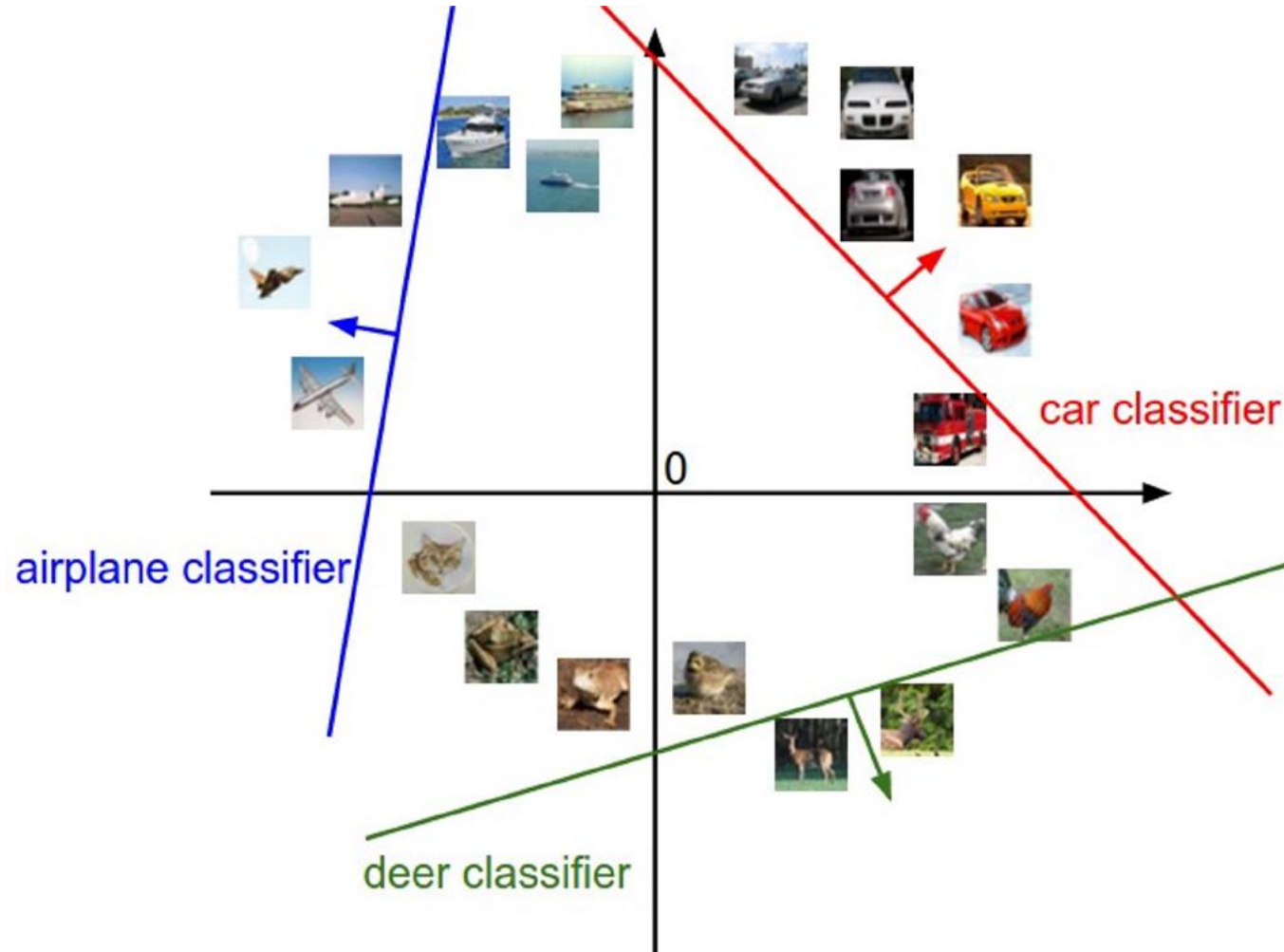


$$f(x_i, W, b) = Wx_i + b$$

Example trained weights of a linear classifier trained on CIFAR-10:



Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

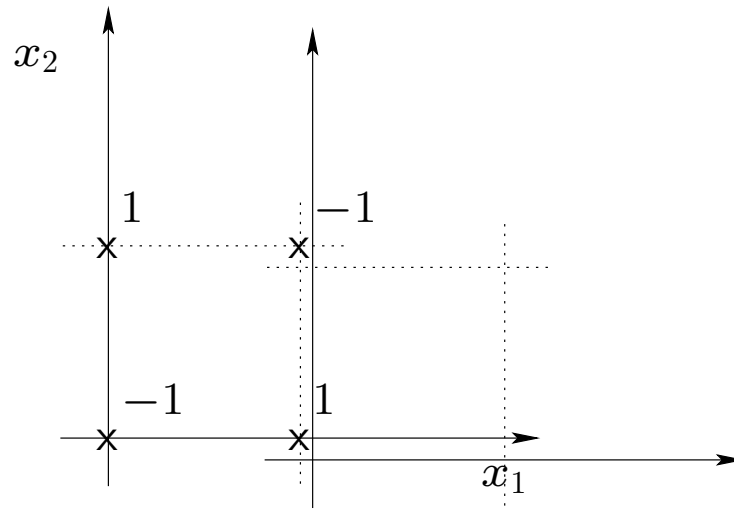


[32x32x3]

array of numbers 0...1
(3072 numbers total)

Model selection

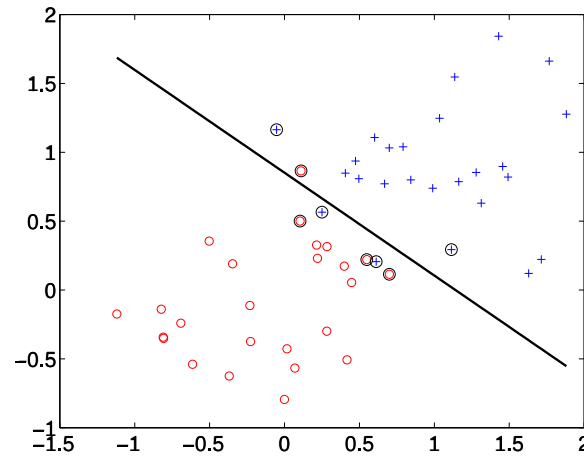
- The simple linear classifier cannot solve all the problems (e.g., XOR)



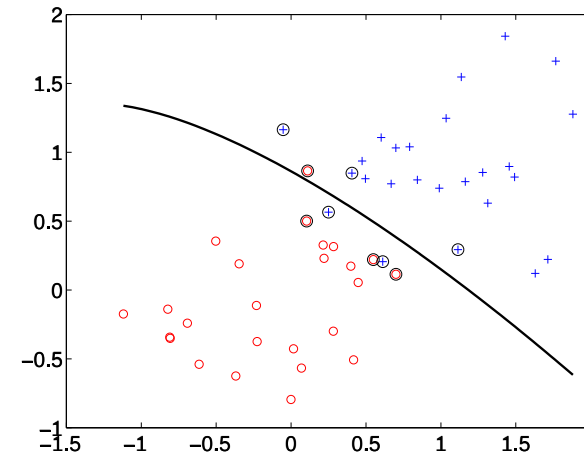
- Can we rethink the approach to do even better?
- We can, for example, add “polynomial experts”

$$\hat{y} = \text{sign} (\theta_1 x_1 + \dots + \theta_d x_d + \theta_{12} x_1 x_2 + \dots)$$

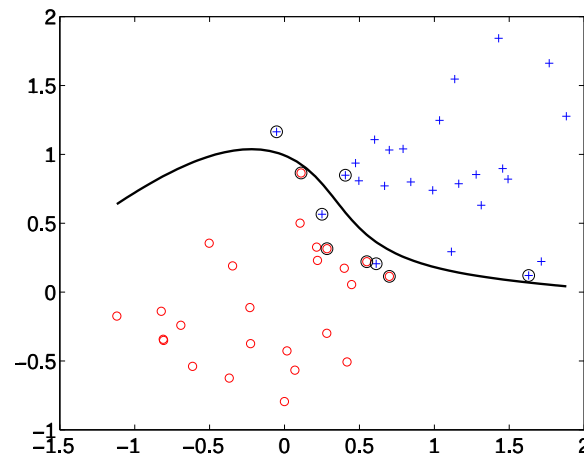
Model selection (cont'd)



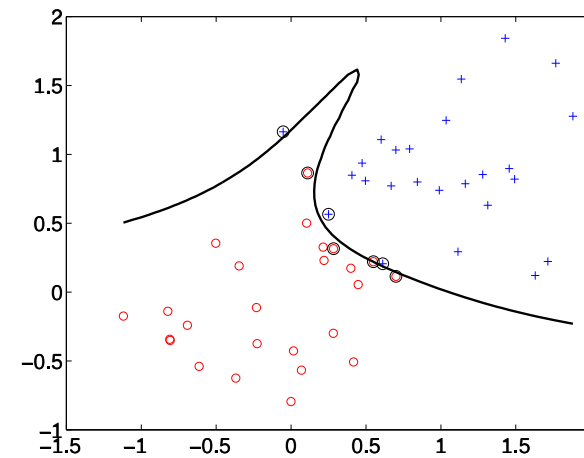
linear



2nd order polynomial



4th order polynomial



8th order polynomial

Review: The learning problem

Image Classification



- **Hypothesis class:** we consider some **restricted** set $\mathcal{F}$ of mappings $f : \mathcal{X} \rightarrow \mathcal{L}$ from images to labels
- **Estimation:** on the basis of a training set of examples and labels, $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, we find an estimate $\hat{f} \in \mathcal{F}$
- **Evaluation:** we measure how well $\hat{f}$ **generalizes** to yet unseen examples, i.e., whether $\hat{f}(\mathbf{x}_{new})$ agrees with y_{new}

Hypothesis and estimation

- We used a simple linear classifier, a parameterized mapping $f(\mathbf{x}; \theta)$ from images $\mathcal{X}$ to labels $\mathcal{L}$, to solve a binary image classification problem (2's vs 3's):

$$\hat{y} = f(\mathbf{x}; \theta) = \text{sign}(\theta \cdot \mathbf{x})$$

where $\mathbf{x}$ is a pixel image and $\hat{y} \in \{-1, 1\}$.

- The parameters θ were adjusted on the basis of the training examples and labels according to a simple mistake driven update rule (written here in a vector form)

$$\theta \leftarrow \theta + y_i \mathbf{x}_i, \quad \text{whenever } y_i \neq \text{sign}(\theta \cdot \mathbf{x}_i)$$

- The update rule attempts to minimize the number of errors that the classifier makes on the training examples

Estimation criterion

- We can formulate the binary classification problem more explicitly by defining a **zero-one loss**:

$$\text{Loss}(y, \hat{y}) = \begin{cases} 0, & y = \hat{y} \\ 1, & y \neq \hat{y} \end{cases}$$

so that

$$\frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, \hat{y}_i) = \frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f(\mathbf{x}_i; \theta))$$

gives the fraction of prediction errors on the training set.

- This is a function of the parameters θ and we can try to minimize it directly.

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the
scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

Losses: **2.9**

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

and using the shorthand for the
 scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$\begin{aligned}
 &= \max(0, 5.1 - 3.2 + 1) \\
 &\quad + \max(0, -1.7 - 3.2 + 1) \\
 &= \max(0, 2.9) + \max(0, -3.9) \\
 &= 2.9 + 0 \\
 &= 2.9
 \end{aligned}$$

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

and using the shorthand for the
 scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$\begin{aligned}
 &= \max(0, 1.3 - 4.9 + 1) \\
 &\quad + \max(0, 2.0 - 4.9 + 1) \\
 &= \max(0, -2.6) + \max(0, -1.9) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

and using the shorthand for the
 scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$\begin{aligned}
 &= \max(0, 2.2 - (-3.1) + 1) \\
 &\quad + \max(0, 2.5 - (-3.1) + 1) \\
 &= \max(0, 5.3) + \max(0, 5.6) \\
 &= 5.3 + 5.6 \\
 &= 10.9
 \end{aligned}$$

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

and using the shorthand for the
 scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

and the full training loss is the
 mean over all examples in the
 training data:

$$L = \frac{1}{N} \sum_{i=1}^N L_i$$

$$\begin{aligned} L &= (2.9 + 0 + 10.9)/3 \\ &= 4.6 \end{aligned}$$

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

and using the shorthand for the
 scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: what is the min/max
 possible loss?

There is something missing in the loss:

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

e.g. suppose that we found a W such that $L = 0$.
Is this W unique?

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Before:

$$\begin{aligned}
 &= \max(0, 1.3 - 4.9 + 1) \\
 &\quad + \max(0, 2.0 - 4.9 + 1) \\
 &= \max(0, -2.6) + \max(0, -1.9) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$

With W twice as large:

$$\begin{aligned}
 &= \max(0, 2.6 - 9.8 + 1) \\
 &\quad + \max(0, 4.0 - 9.8 + 1) \\
 &= \max(0, -6.2) + \max(0, -4.8) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$

Weight Regularization

λ = regularization strength
(hyperparameter)

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

In common use:

L2 regularization

L1 regularization

Elastic net (L1 + L2)

Max norm regularization

Dropout (will see later)

Batch normalization (will see later)

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

L2 regularization: motivation

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Estimation criterion (revisited)

- We have reduced the estimation problem to a minimization problem

$$\text{find } \theta \text{ that minimizes } \overbrace{\frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f(\mathbf{x}_i; \theta))}^{\text{empirical loss}}$$

- valid for any parameterized class of mappings from examples to predictions
 - valid when the predictions are discrete labels, real valued, or other provided that the loss is defined appropriately
 - may be ill-posed (under-constrained) as stated
- But why is it sensible to minimize the empirical loss in the first place since we are only interested in the performance on new examples?

Training and test performance: sampling

- We assume that each training **and** test example-label pair, $(\mathbf{x}, y)$ is drawn **independently at random** from the **same** but unknown population of examples and labels.
- We can represent this population as a joint probability distribution $P(\mathbf{x}, y)$ so that each training/test example is a **sample** from this distribution

$$(\mathbf{x}_i, y_i) \sim P$$

Indyk

Barzilay

Collins

Jaakkola



Indy

Training and test performance: sampling

- We assume that each training **and** test example-label pair, $(\mathbf{x}, y)$ is drawn **independently at random** from the **same** but unknown population of examples and labels.
- We can represent this population as a joint probability distribution $P(\mathbf{x}, y)$ so that each training/test example is a **sample** from this distribution
 $(\mathbf{x}_i, y_i) \sim P$

Indyk

Barzilay

Collins

Jaakkola

$$\text{Empirical (training) loss} = \frac{1}{n} \sum_{i=1}^n \text{Loss}(y_i, f(\mathbf{x}_i; \theta))$$

$$\text{Expected (test) loss} = E_{(\mathbf{x}, y) \sim P} \{ \text{Loss}(y, f(\mathbf{x}; \theta)) \}$$

- The training loss based on a few sampled examples and labels serves as a proxy for the test performance measured over the whole population

Indy

Regression, example

Linear regression

- Estimation, errors, analysis

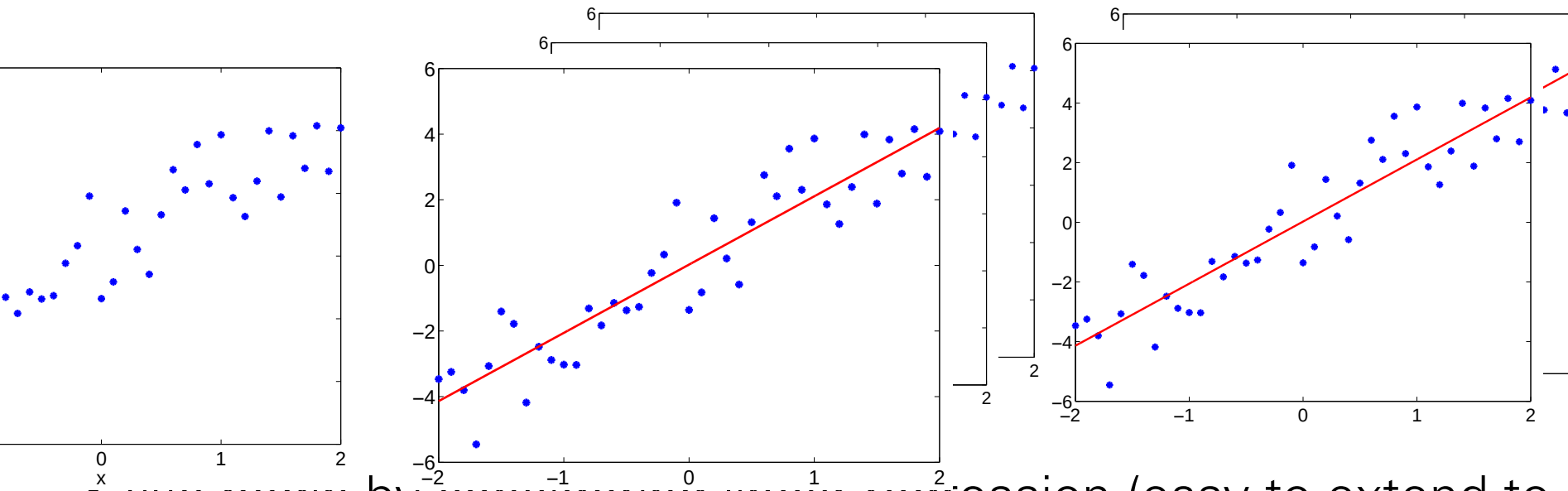
Regression

- The goal is to make quantitative (real valued) predictions on the basis of a (vector of) features or attributes
- Example: predicting vehicle fuel efficiency (mpg) from 8 attributes

y	x				
	cyls	disp	hp	weight	...
18.0	8	307.0	130.00	3504	...
26.0	4	97.00	46.00	1835	...
33.5	4	98.00	83.00	2075	...
...					

- We need to
 - specify the class of functions (e.g., linear)
 - select how to measure prediction loss
 - solve the resulting minimization problem

Linear regression



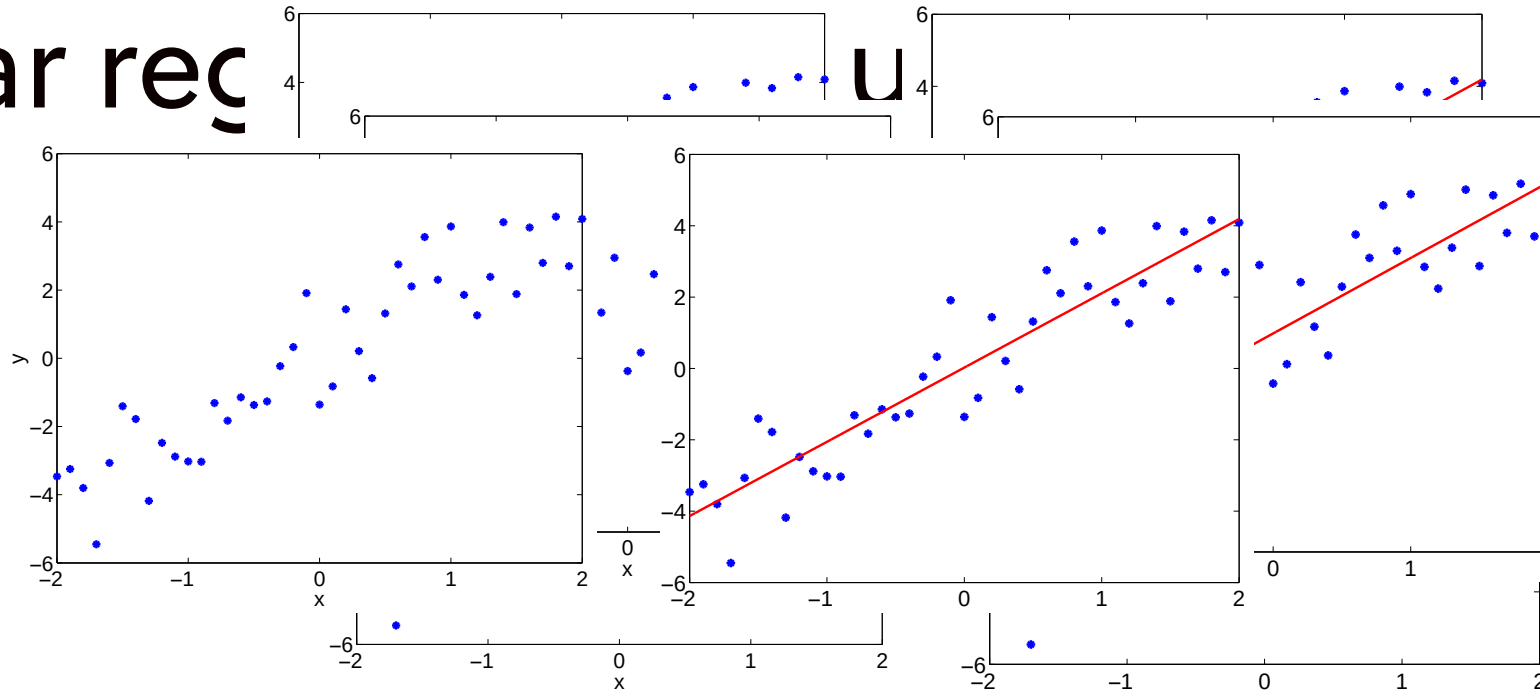
- we begin by considering linear regression (easy to extend to more complex predictions later on)

$$f : \mathcal{R} \rightarrow \mathcal{R} \quad f(x; \mathbf{w}) = w_0 + w_1 x$$

$$f : \mathcal{R}^d \rightarrow \mathcal{R} \quad f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + \dots w_d x_d$$

where $\mathbf{w}$ are **parameters** we need to set.

Linear reg



$$f : \mathcal{R} \rightarrow \mathcal{R} \quad f(x; \mathbf{w}) = w_0 + w_1 x$$

$$f : \mathcal{R}^d \rightarrow \mathcal{R} \quad f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + \dots w_d x_d$$

- We can measure the prediction loss in terms of squared error, $\text{Loss}(y, \hat{y}) = (y - \hat{y})^2$, so that the empirical loss on n training samples becomes mean squared error

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i; \mathbf{w}))^2$$

Linear regression: estimation

- We have to minimize the **empirical** squared loss

$$\begin{aligned} J_n(\mathbf{w}) &= \frac{1}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i; \mathbf{w}))^2 \\ &= \frac{1}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)^2 \quad (1\text{-dim}) \end{aligned}$$

- By setting the derivatives with respect to w_1 and w_0 to zero, we get necessary conditions for the “optimal” parameter values

$$\frac{\partial}{\partial w_1} J_n(\mathbf{w}) = 0$$

$$\frac{\partial}{\partial w_0} J_n(\mathbf{w}) = 0$$

Optimality conditions: derivation

$$\frac{\partial}{\partial w_1} J_n(\mathbf{w}) = \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)^2$$

Optimality conditions: derivation

$$\begin{aligned}\frac{\partial}{\partial w_1} J_n(\mathbf{w}) &= \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)^2\end{aligned}$$

Optimality conditions: derivation

$$\begin{aligned}\frac{\partial}{\partial w_1} J_n(\mathbf{w}) &= \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)^2 \\ &= \frac{2}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i) \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)\end{aligned}$$

Optimality conditions: derivation

$$\begin{aligned}\frac{\partial}{\partial w_1} J_n(\mathbf{w}) &= \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)^2 \\&= \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)^2 \\&= \frac{2}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i) \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i) \\&= \frac{2}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)(-x_i) = 0\end{aligned}$$

Optimality conditions: derivation

$$\begin{aligned}\frac{\partial}{\partial w_1} J_n(\mathbf{w}) &= \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)^2 \\&= \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)^2 \\&= \frac{2}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i) \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i) \\&= \frac{2}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i) (-x_i) = 0 \\ \frac{\partial}{\partial w_0} J_n(\mathbf{w}) &= \frac{2}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i) (-1) = 0\end{aligned}$$

Linear regression: matrix notation

- We can express the solution a bit more generally by resorting to a matrix notation

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_1 \\ \cdots & \cdots \\ 1 & x_n \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

so that

$$\begin{aligned} \frac{1}{n} \sum_{t=1}^n (y_t - w_0 - w_1 x_t)^2 &= \frac{1}{n} \left\| \begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix} - \begin{bmatrix} 1 & x_1 \\ \cdots & \cdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \right\|^2 \\ &= \frac{1}{n} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 \end{aligned}$$

Linear regression: solution

- By setting the derivatives of $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2/n$ to zero, we get the same optimality conditions as before, now expressed in a matrix form

$$\frac{\partial}{\partial \mathbf{w}} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = \frac{\partial}{\partial \mathbf{w}} \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Linear regression: solution

- By setting the derivatives of $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2/n$ to zero, we get the same optimality conditions as before, now expressed in a matrix form

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 &= \frac{\partial}{\partial \mathbf{w}} \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= \frac{2}{n} \mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w})\end{aligned}$$

Linear regression: solution

- By setting the derivatives of $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2/n$ to zero, we get the same optimality conditions as before, now expressed in a matrix form

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 &= \frac{\partial}{\partial \mathbf{w}} \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= \frac{2}{n} \mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= \frac{2}{n} (\mathbf{X}^T \mathbf{y} - \mathbf{X}^T \mathbf{X} \mathbf{w}) = \mathbf{0}\end{aligned}$$

which gives

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

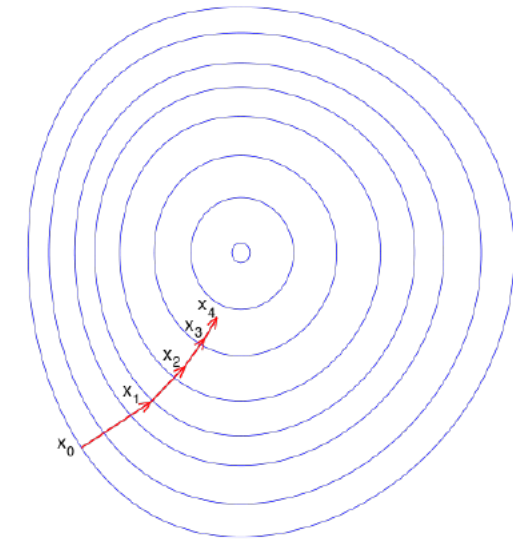
- The solution is a linear function of the outputs $\mathbf{y}$

Alternative: Gradient Descent Algorithm

- $\Delta = -\frac{1}{T} \sum_t \nabla_{\theta} l(f(\mathbf{x}^{(t)}; \theta), y^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta)$
- One straightforward method: **gradient descent**
 - • Initialize θ (e.g. randomly)
 - repeatedly update θ based on the gradient

$$\Delta = -\frac{1}{T} \sum_t \nabla_{\theta} l(f(\mathbf{x}^{(t)}; \theta), y^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta)$$
$$\theta \leftarrow \theta + \alpha \Delta$$

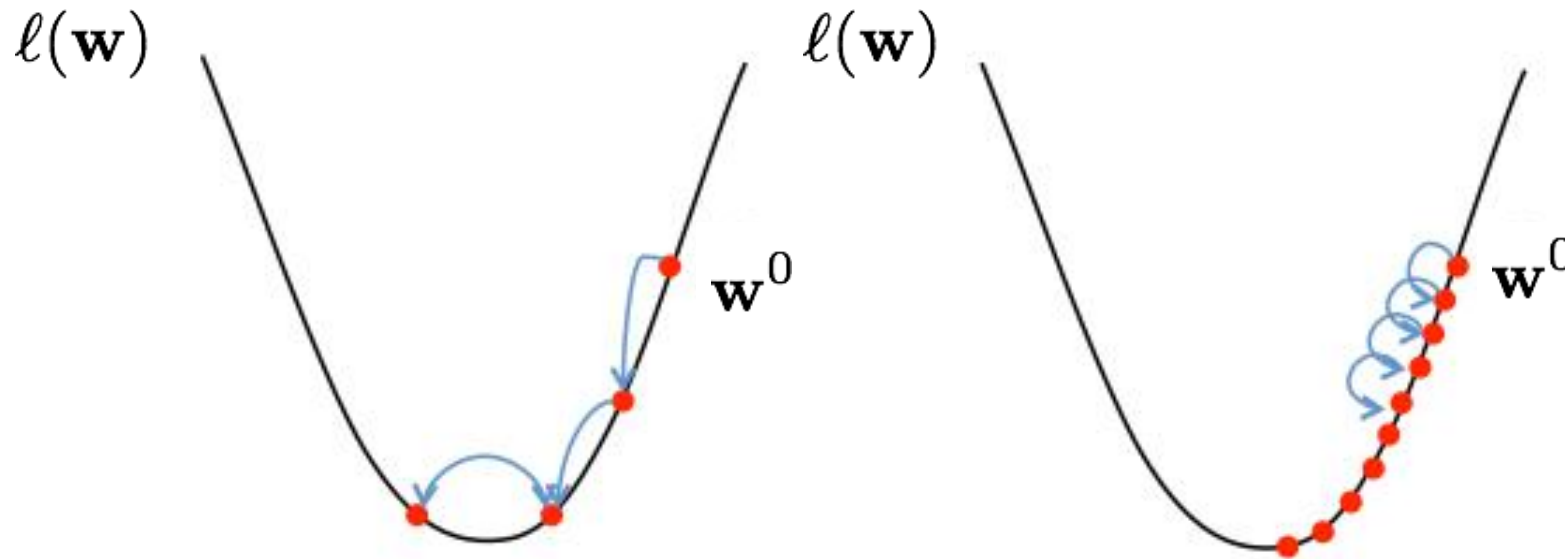
- α is the **learning rate**





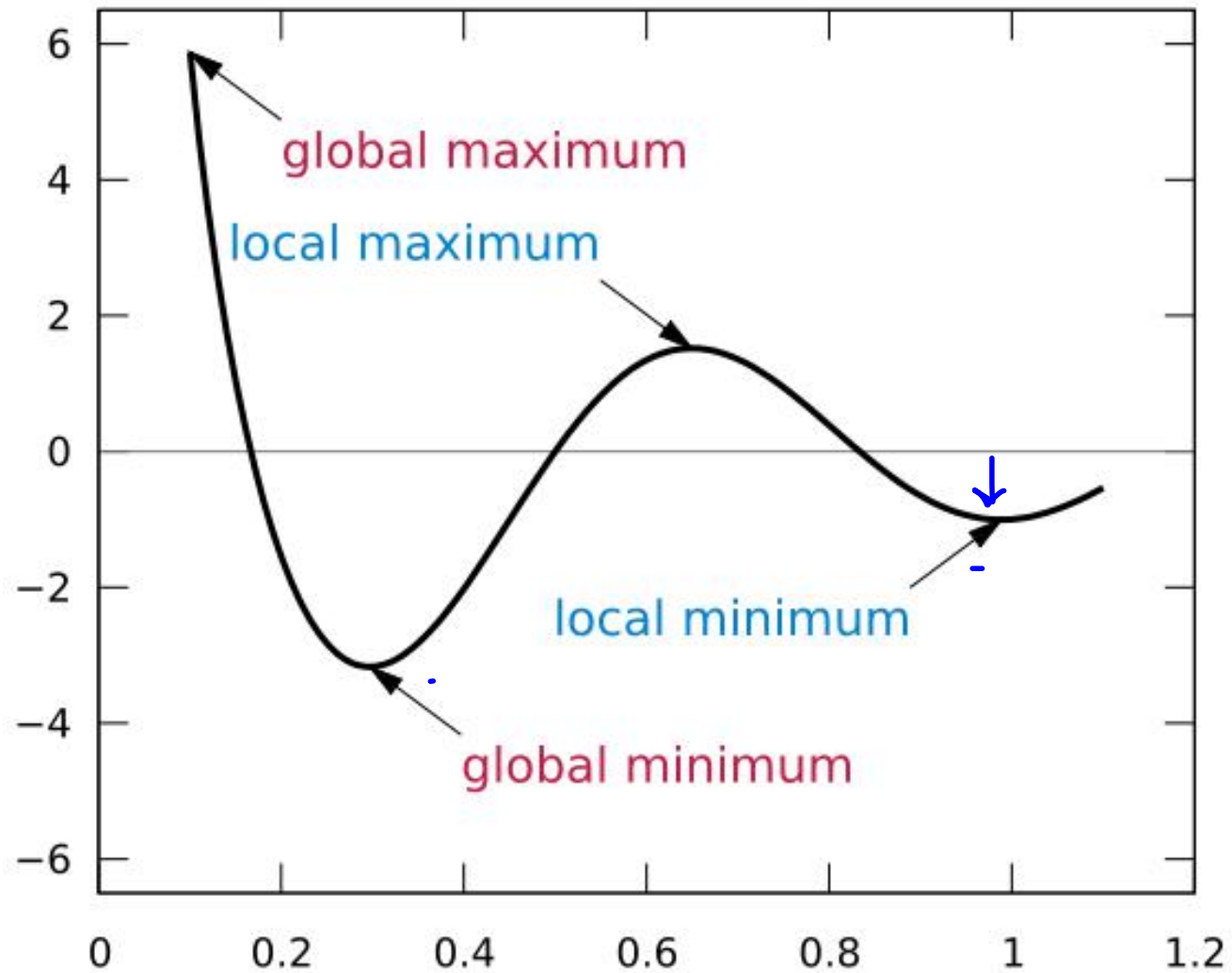


Effect of learning rate λ



- Large $\lambda \Rightarrow$ Fast convergence but larger residual error
Also possible oscillations
- Small $\lambda \Rightarrow$ Slow convergence but small residual error

Local and Global Optima



Stochastic Gradient Descent

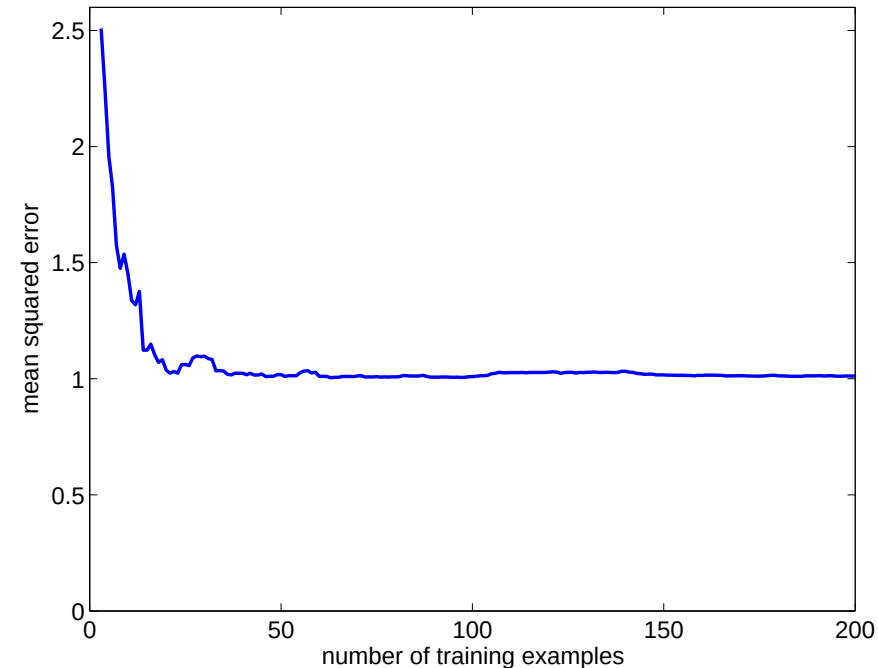
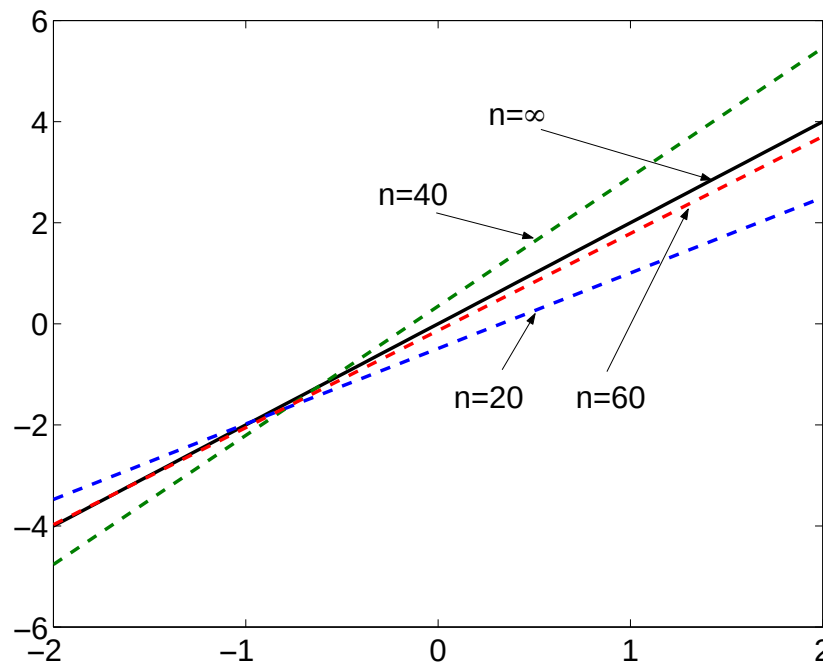
- $l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- $\Omega(\boldsymbol{\theta})$
- Two ways to generalize this for all examples in training set:
 - $\Delta = -\frac{1}{T} \sum_t \nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
 - 1. **Batch updates:** sum or average updates across every example n , then change the parameter values
 - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta$
 - 2. **Stochastic/online updates:** update the parameters for each training case in turn, according to its own gradients

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$$

Linear regression: generalization

- As the number of training examples increases our solution gets “better”



We'd like to understand the error a bit better

Linear regression: types of errors

- **Structural error** measures the error introduced by the limited function class (infinite training data):

related to the
capacity of
the model

$$\min_{w_1, w_0} E_{(x,y) \sim P} (y - w_0 - w_1 x)^2 = E_{(x,y) \sim P} (y - w_0^* - w_1^* x)^2$$

where (w_0^*, w_1^*) are the optimal linear regression parameters.

- **Approximation error** measures how close we can get to the optimal linear predictions with limited training data:

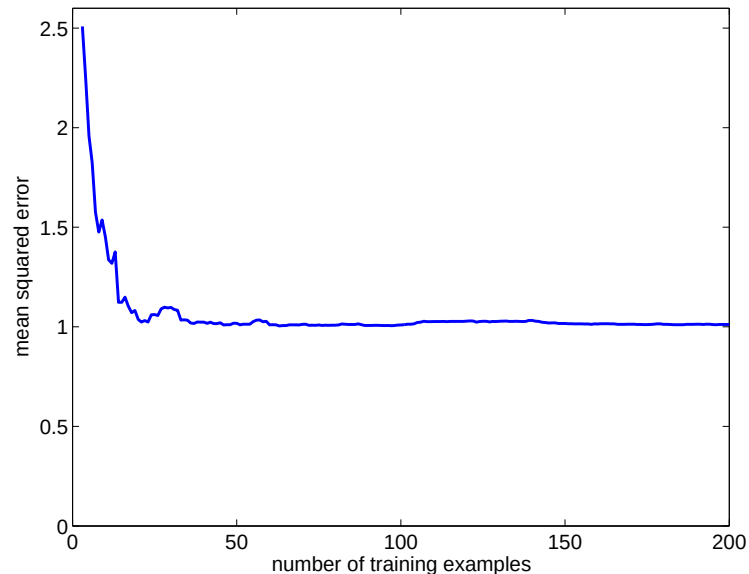
$$E_{(x,y) \sim P} (w_0^* + w_1^* x - \hat{w}_0 - \hat{w}_1 x)^2$$

where $(\hat{w}_0, \hat{w}_1)$ are the parameter estimates based on a small training set (therefore themselves random variables).

Linear regression: error decomposition

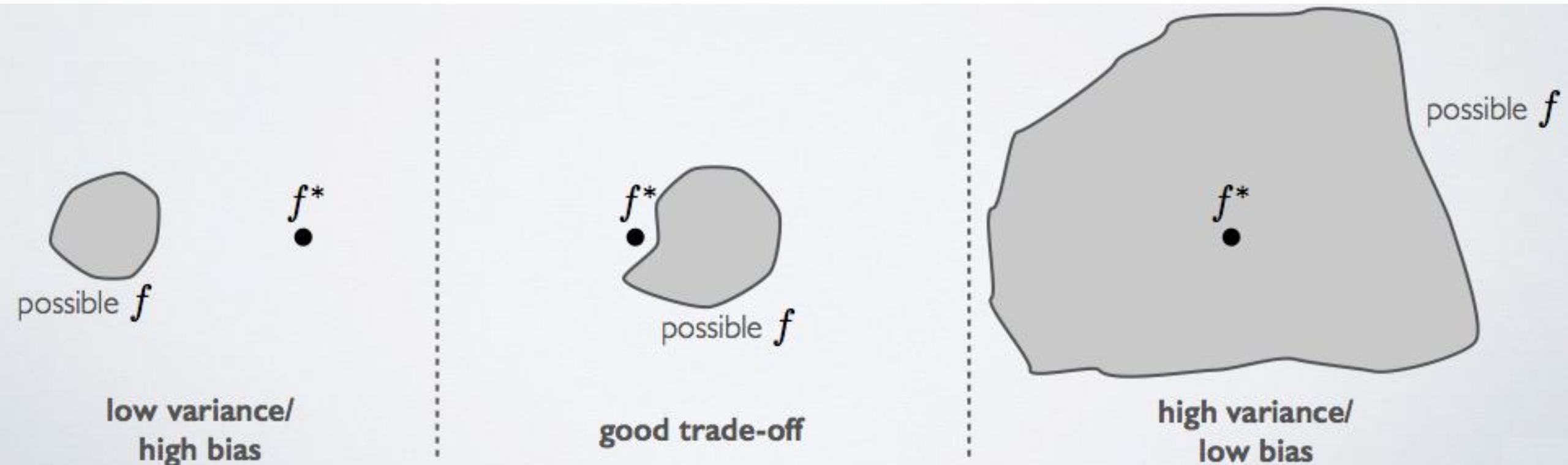
- The expected error of our linear regression function decomposes into the sum of structural and approximation errors

$$\begin{aligned} E_{(x,y) \sim P} (y - \hat{w}_0 - \hat{w}_1 x)^2 = \\ E_{(x,y) \sim P} (y - w_0^* - w_1^* x)^2 + \\ E_{(x,y) \sim P} (w_0^* + w_1^* x - \hat{w}_0 - \hat{w}_1 x)^2 \end{aligned}$$



Bias-Variance Tradeoff

- **Variance** of trained model: does it vary a lot if the training set changes
- **Bias** of trained model: is the average model close to the true solution?
- Generalization error can be seen as the sum of bias and the variance



Parametric vs. non-parametric models

- **Parametric model:** its capacity is fixed and does not increase with the amount of training data
 - examples: linear classifier, neural network with fixed number of hidden units, etc.
- **Non-parametric model:** the capacity increases with the amount of training data
 - examples: k nearest neighbors classifier, neural network with adaptable hidden layer size, etc.

Beyond linear regression models

- additive regression models, examples
- generalization and cross-validation
- population minimizer

Linear regression

- Linear regression functions,

$$f : \mathcal{R} \rightarrow \mathcal{R} \quad f(x; \mathbf{w}) = w_0 + w_1x, \quad \text{or}$$

$$f : \mathcal{R}^d \rightarrow \mathcal{R} \quad f(\mathbf{x}; \mathbf{w}) = w_0 + w_1x_1 + \dots + w_dx_d$$

combined with the squared loss, are convenient because they are **linear in the parameters**.

—we get closed form estimates of the parameters

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

where, for example, $\mathbf{y} = [y_1, \dots, y_n]^T$.

—the resulting prediction errors $\epsilon_i = y_i - f(\mathbf{x}_i; \hat{\mathbf{w}})$ are uncorrelated with any linear function of the inputs $\mathbf{x}$.

—we can easily extend these to non-linear functions of the inputs while still keeping them linear in the parameters

Beyond linear regression

- Example extension: m^{th} order polynomial regression where $f : \mathcal{R} \rightarrow \mathcal{R}$ is given by

$$f(x; \mathbf{w}) = w_0 + w_1x + \dots + w_{m-1}x^{m-1} + w_mx^m$$

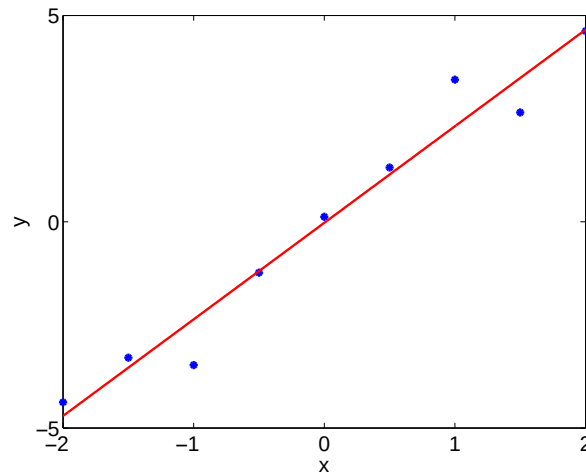
- linear in the parameters, non-linear in the inputs
- solution as before

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

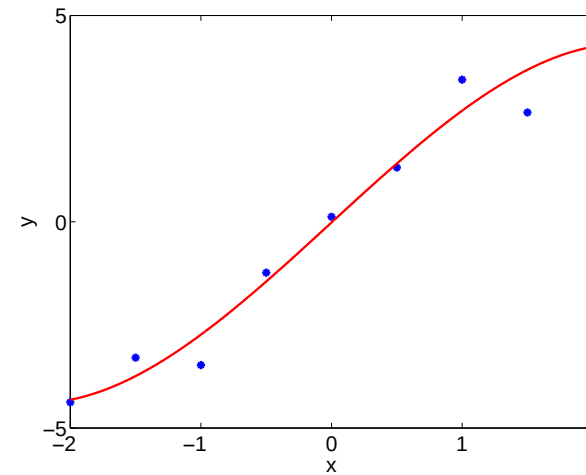
where

$$\hat{\mathbf{w}} = \begin{bmatrix} \hat{w}_0 \\ \hat{w}_1 \\ \dots \\ \hat{w}_m \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix}$$

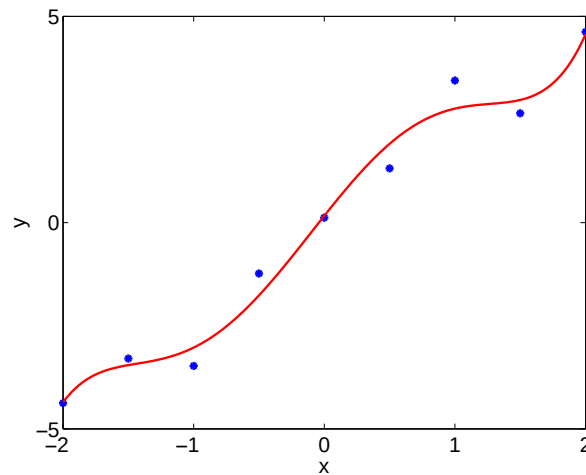
Polynomial regression



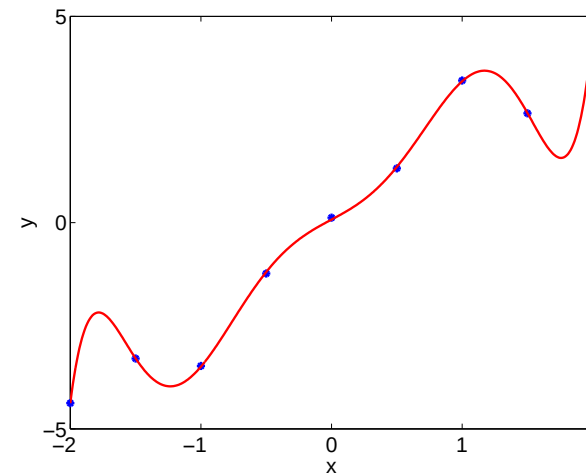
degree = 1



degree = 3



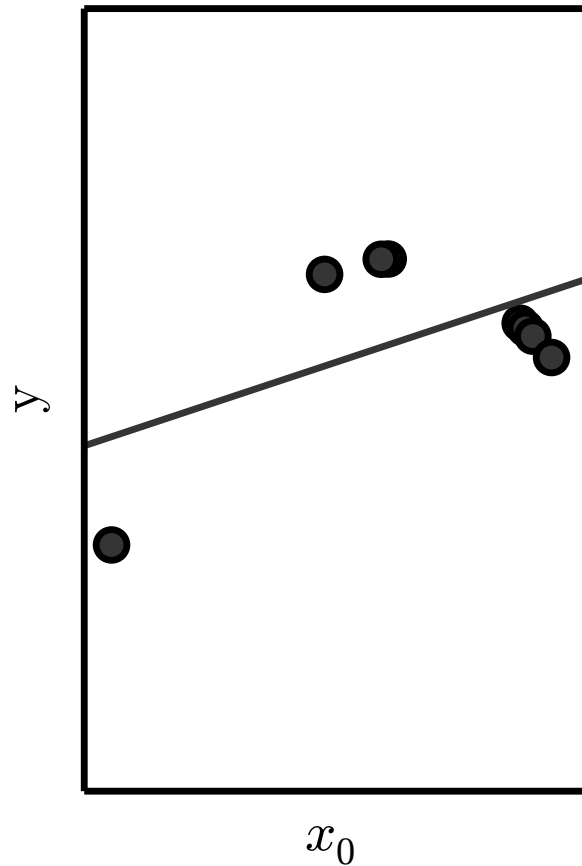
degree = 5



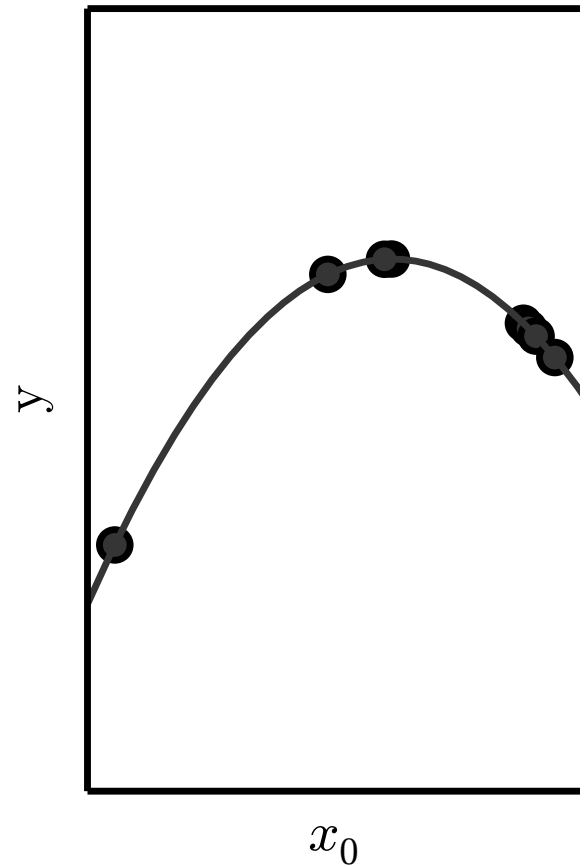
degree = 7

Underfitting and Overfitting

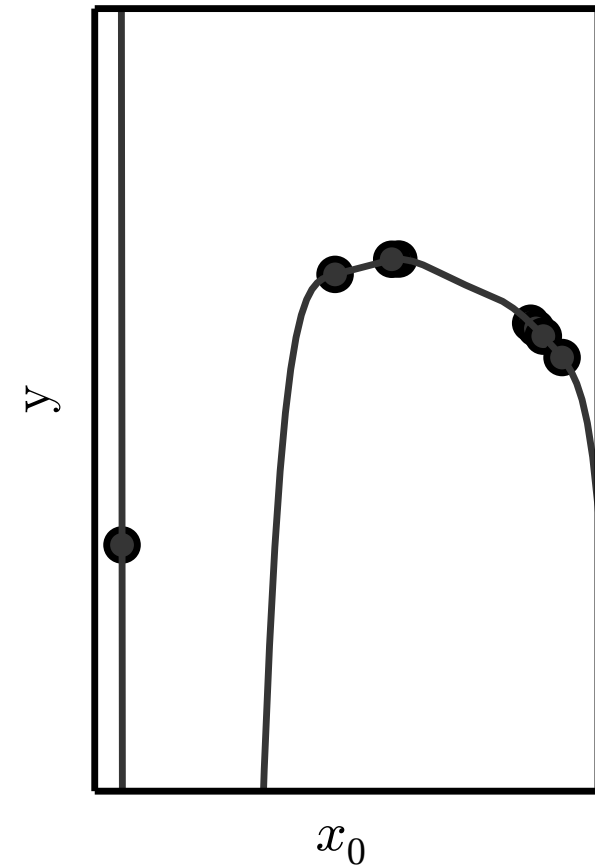
Underfitting



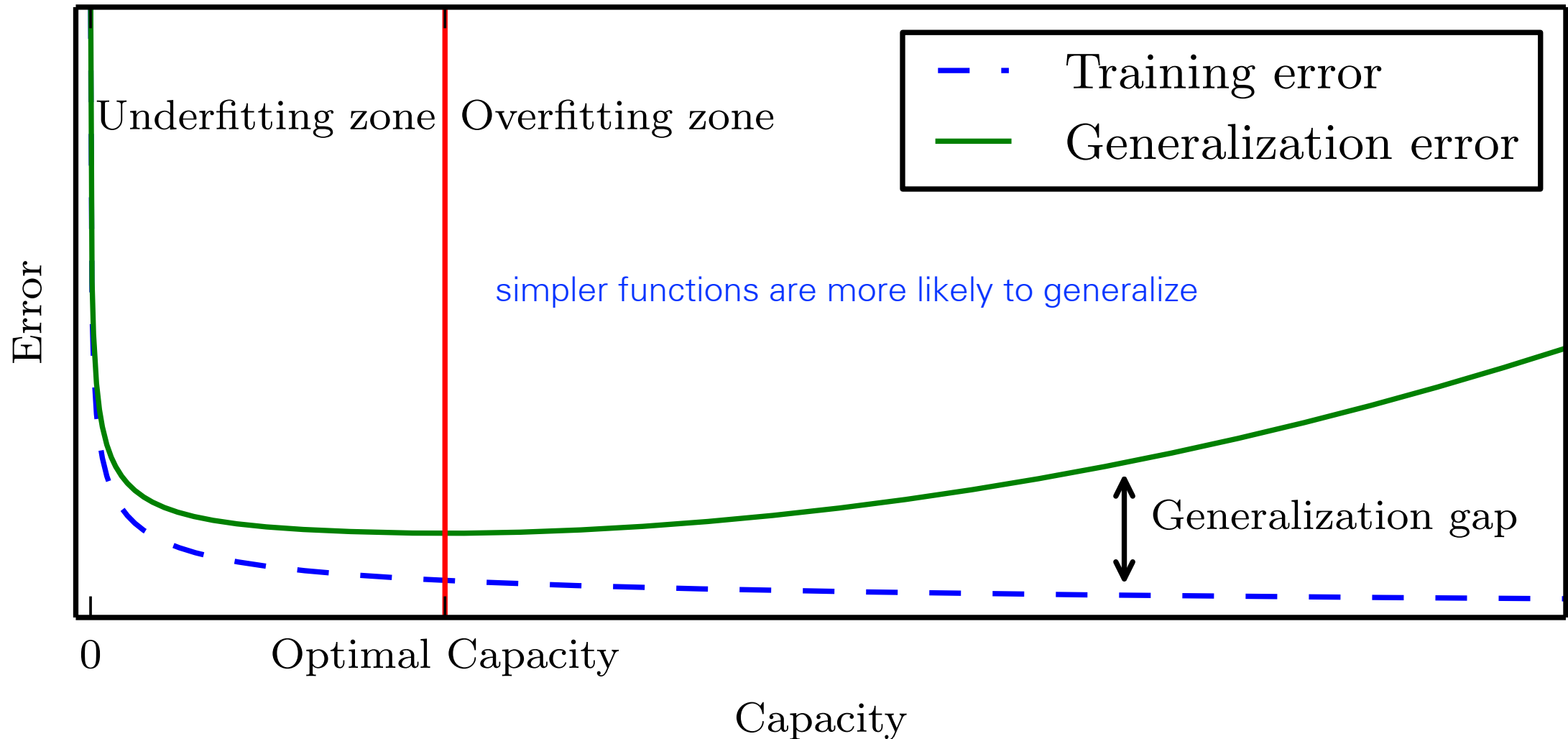
Appropriate capacity



Overfitting

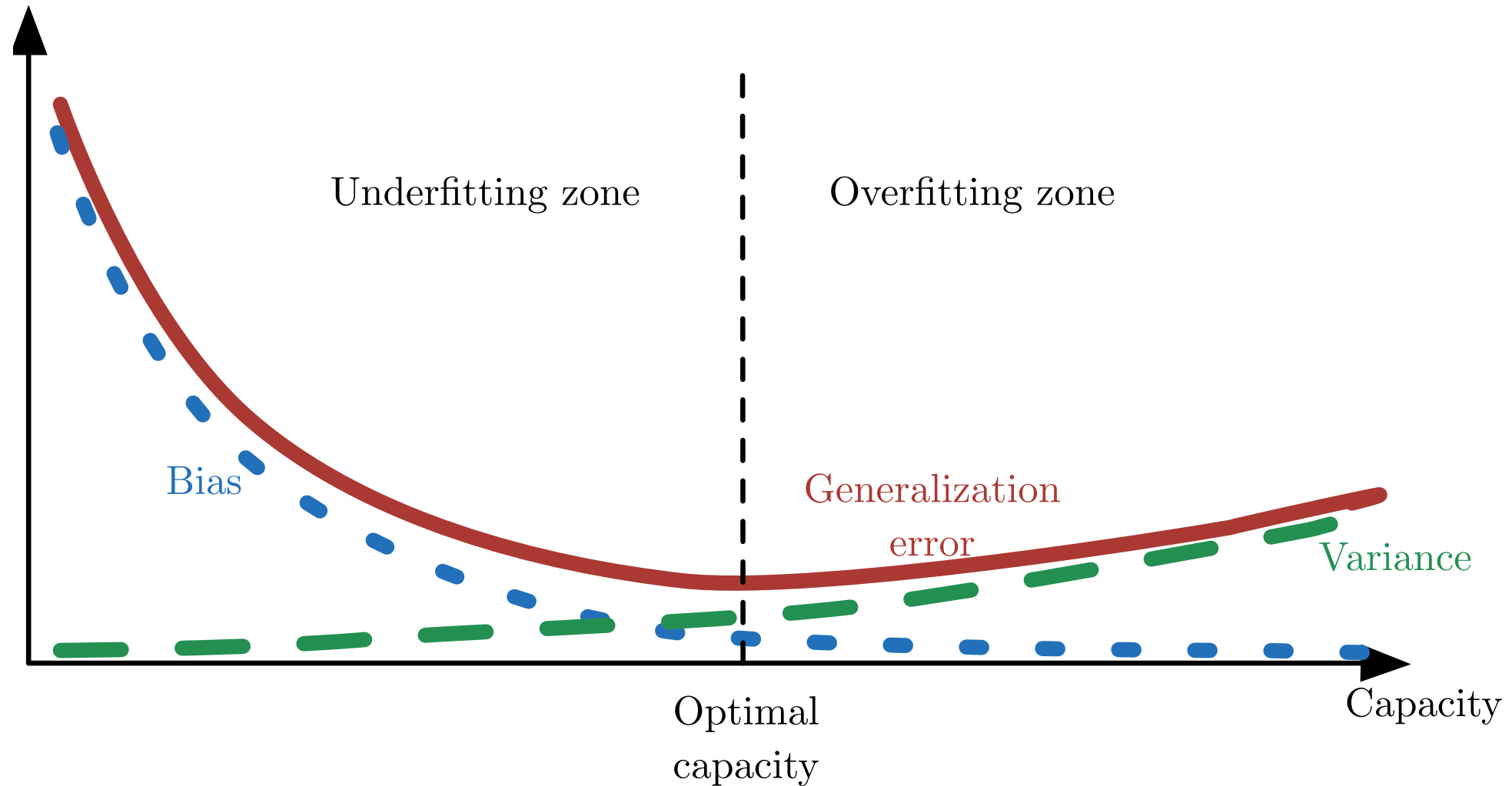


Generalization and Capacity



Bias and Variance

sufficiently simpler models are more likely to generalize

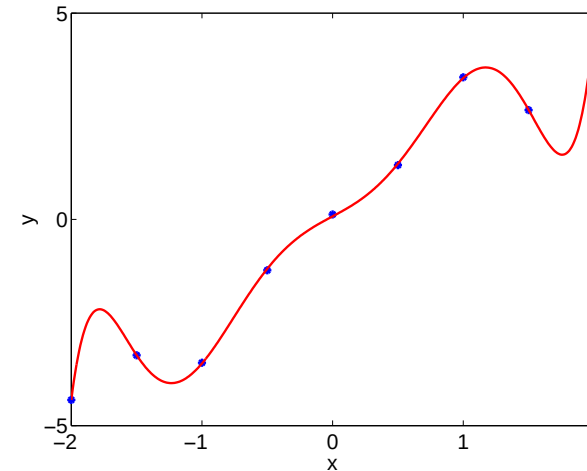


Complexity and overfitting

- With limited training examples our polynomial regression model may achieve zero training error but nevertheless has a large test (generalization) error

$$\text{train} \quad \frac{1}{n} \sum_{t=1}^n (y_t - f(x_t; \hat{\mathbf{w}}))^2 \approx 0$$

$$\text{test} \quad E_{(x,y) \sim P} (y - f(x; \hat{\mathbf{w}}))^2 \gg 0$$



- We suffer from **overfitting** when the training error no longer bears any relation to the generalization error

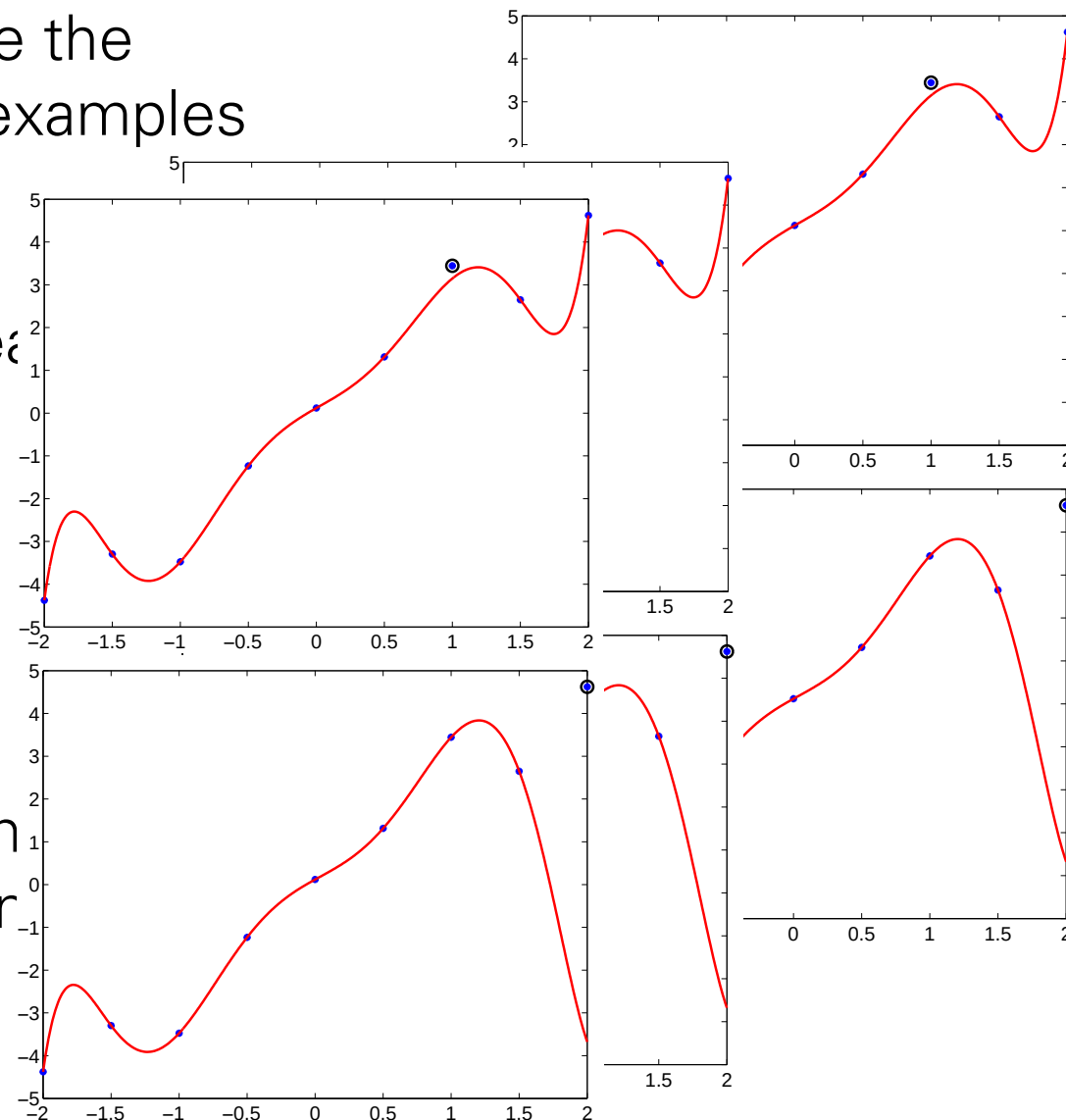
Avoiding overfitting: cross-validation

- **Cross-validation** allows us to estimate the generalization error based on training examples alone

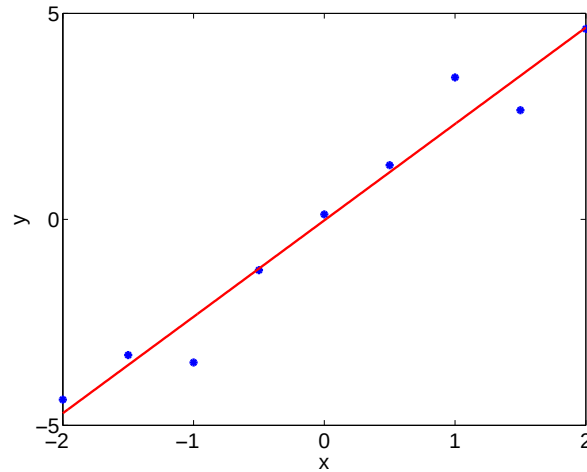
Leave-one-out cross-validation treats each example in turn as a test example:

$$CV = \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i; \hat{\mathbf{w}}^{-i}))^2$$

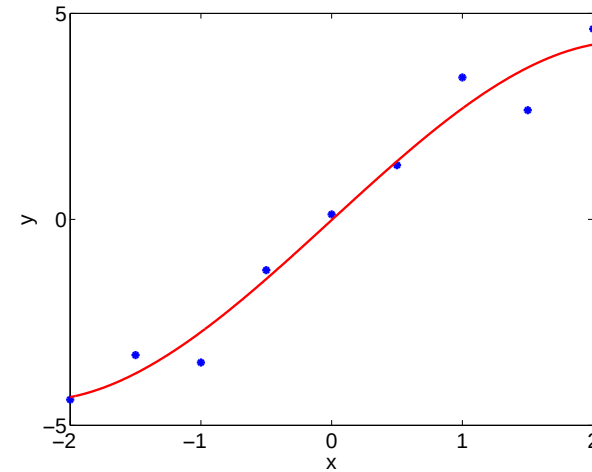
where $\hat{\mathbf{w}}^{-i}$ are the least squares estimation parameters without the i^{th} training example



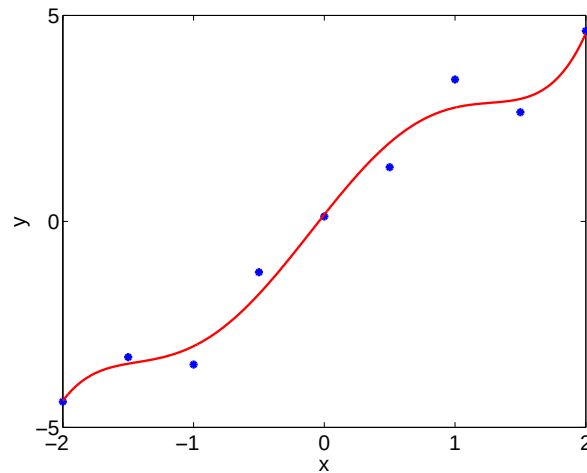
Polynomial regression: example (cont'd)



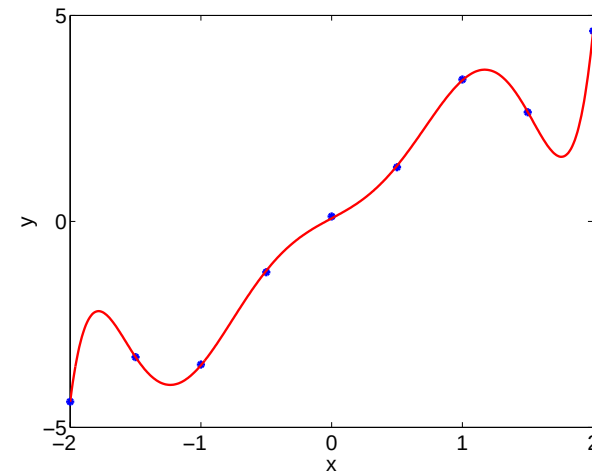
degree = 1, CV = 0.6



degree = 3, CV = 1.5



degree = 5, CV = 6.0



degree = 7, CV = 15.6

Additive models

- More generally, predictions can be based on a linear combination of a set of basis functions (or features) $\{\phi_1(\mathbf{x}), \dots, \phi_m(\mathbf{x})\}$, where each $\phi_i(\mathbf{x}) : \mathcal{R}^d \rightarrow \mathcal{R}$, and

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1\phi_1(\mathbf{x}) + \dots + w_m\phi_m(\mathbf{x})$$

- Examples

If $\phi_i(x) = x^i$, $i = 1, \dots, m$, then

$$f(x; \mathbf{w}) = w_0 + w_1x + \dots + w_{m-1}x^{m-1} + w_mx^m$$

Additive models

- More generally, predictions can be based on a linear combination of a set of basis functions (or features) $\{\phi_1(\mathbf{x}), \dots, \phi_m(\mathbf{x})\}$, where each $\phi_i(\mathbf{x}) : \mathcal{R}^d \rightarrow \mathcal{R}$, and

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1\phi_1(\mathbf{x}) + \dots + w_m\phi_m(\mathbf{x})$$

- Examples

If $\phi_i(x) = x^i$, $i = 1, \dots, m$, then

$$f(x; \mathbf{w}) = w_0 + w_1x + \dots + w_{m-1}x^{m-1} + w_mx^m$$

If $m = d$, $\phi_i(\mathbf{x}) = x_i$, $i = 1, \dots, d$, then

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1x_1 + \dots + w_dx_d$$

Additive models (cont'd)

- The basis functions can capture various (e.g., qualitative) properties of the inputs.
- For example: we can try to rate companies based on text descriptions

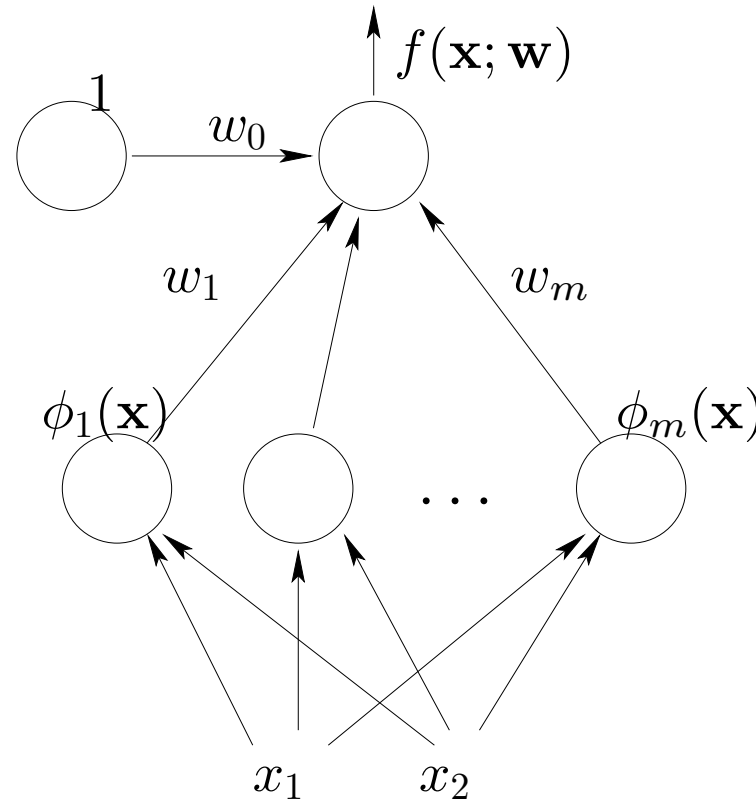
$\mathbf{x}$ = text document (collection of words)

$$\phi_i(\mathbf{x}) = \begin{cases} 1 & \text{if word } i \text{ appears in the document} \\ 0 & \text{otherwise} \end{cases}$$

$$f(\mathbf{x}; \mathbf{w}) = w_0 + \sum_{i \in \text{words}} w_i \phi_i(\mathbf{x})$$

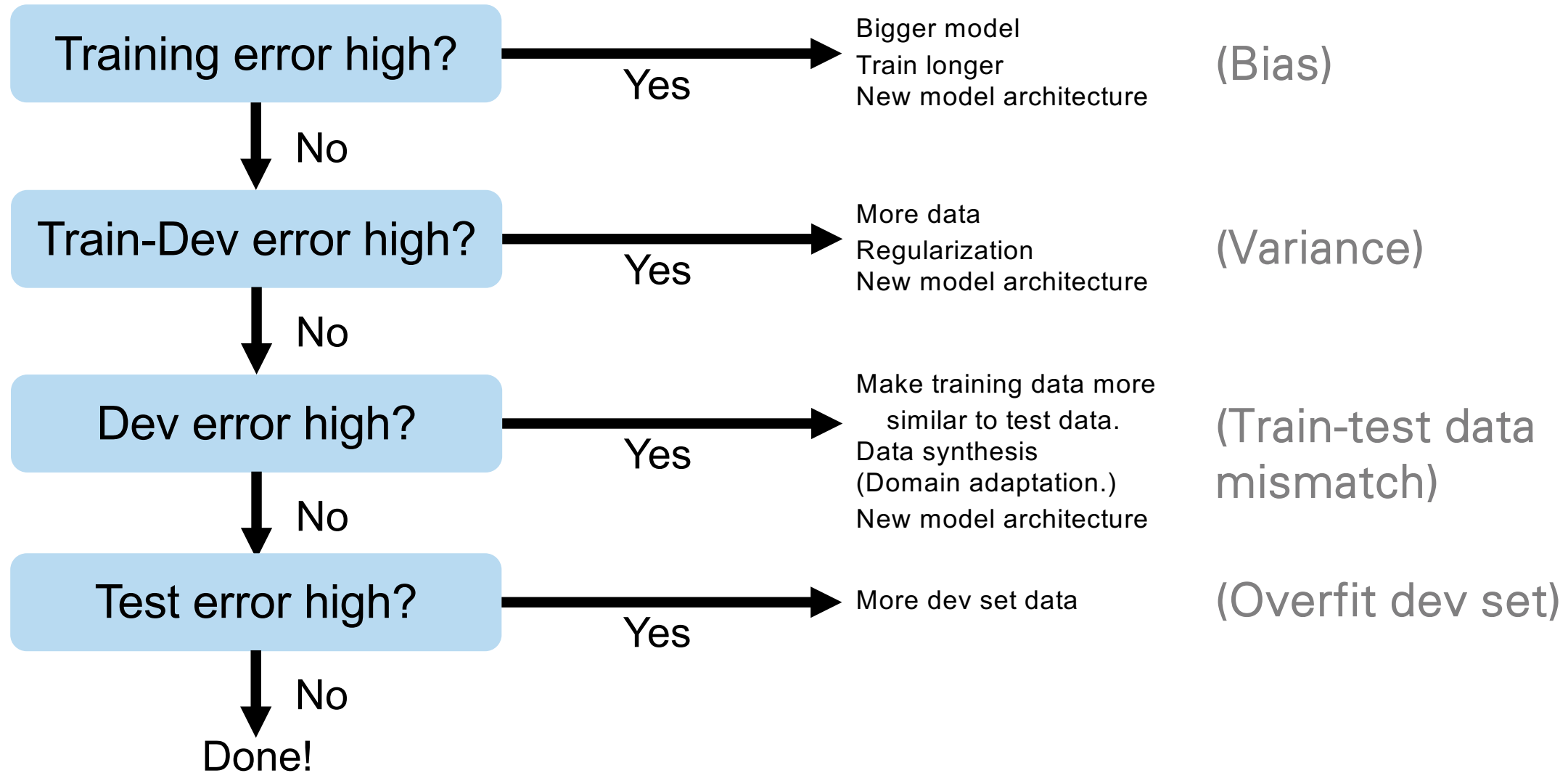
Additive models (cont'd)

- We can view the additive models graphically in terms of simple “units” and “weights”



- In **neural networks**, the basis functions themselves have adjustable parameters (cf. prototypes)

Take-home messages



Statistical regression models

- model formulation, motivation
- maximum likelihood estimation

Statistical view of linear regression

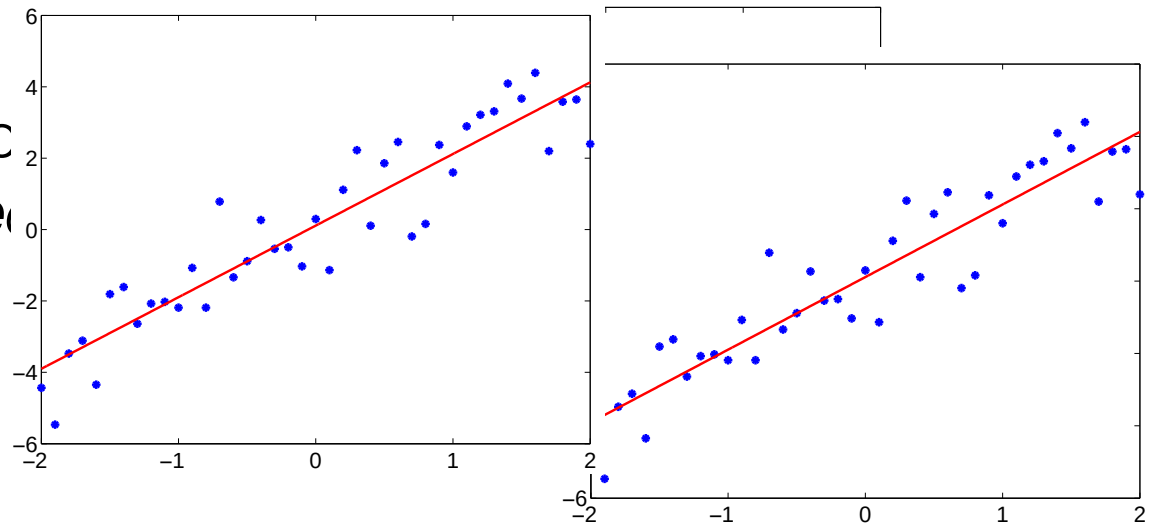
- In a statistical regression model we model both the function and noise

Observed output = function + noise

$$y = f(\mathbf{x}; \mathbf{w}) + \epsilon$$

where, e.g., $\epsilon \sim N(0, \sigma^2)$.

- Whatever we cannot capture with a family of functions will be *interpreted*

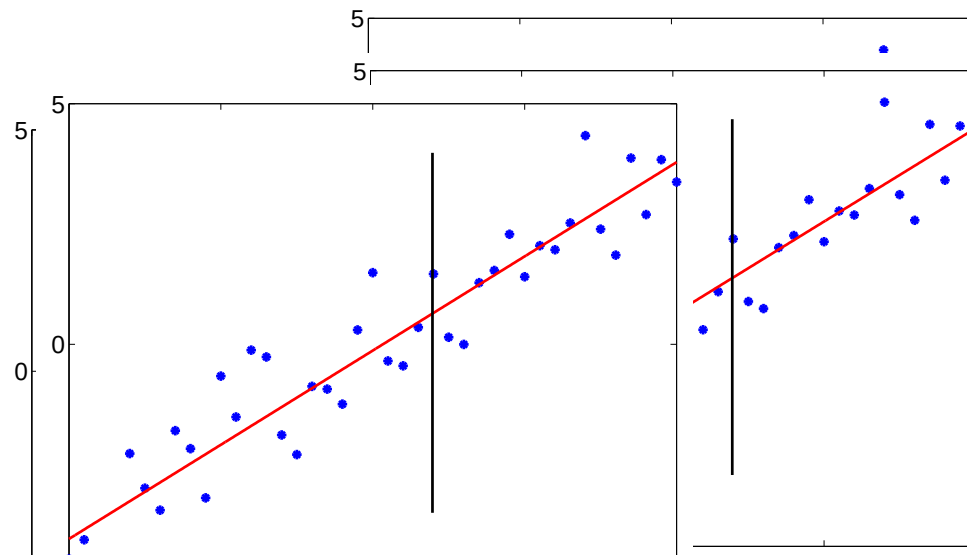


Statistical view of linear regression

- $f(\mathbf{x}; \mathbf{w})$ is trying to capture the mean of the observations y given the input $\mathbf{x}$:

$$\begin{aligned} E\{y \mid \mathbf{x}\} &= E\{f(\mathbf{x}; \mathbf{w}) + \epsilon \mid \mathbf{x}\} \\ &= f(\mathbf{x}; \mathbf{w}) \end{aligned}$$

where $E\{y \mid \mathbf{x}\}$ is the conditional expectation of y given $\mathbf{x}$, evaluated according to the model (not according to the underlying distribution P)



Statistical view of linear regression

- According to our statistical model

$$y = f(\mathbf{x}; \mathbf{w}) + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

the outputs y given $\mathbf{x}$ are normally distributed with mean $f(\mathbf{x}; \mathbf{w})$ and variance σ^2 :

$$p(y|\mathbf{x}, \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2\sigma^2}(y - f(\mathbf{x}; \mathbf{w}))^2 \right\}$$

(we model the uncertainty in the predictions, not just the mean)

- Loss function? Estimation?

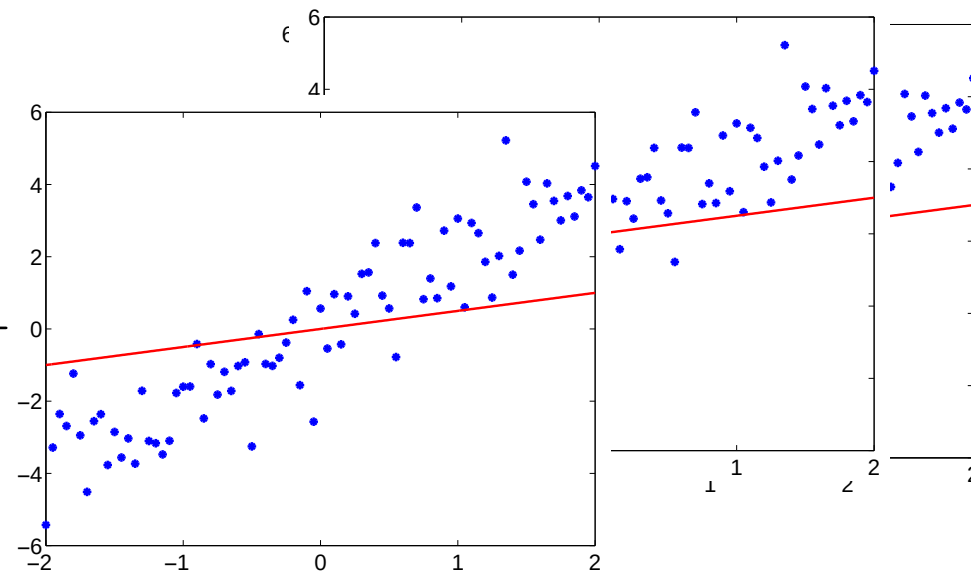
Maximum likelihood estimation

- Given observations $D_n = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ we find the parameters $\mathbf{w}$ that maximize the (conditional) likelihood of the outputs

$$L(D_n; \mathbf{w}, \sigma^2) = \prod_{i=1}^n p(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2)$$

- Example: linear function

$$p(y | \mathbf{x}, \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(y - w_0 - \right.$$



Maximum likelihood estimation (cont'd)

Likelihood of the observed outputs:

$$L(D; \mathbf{w}, \sigma^2) = \prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2)$$

- It is often easier (but equivalent) to try to maximize the log-likelihood:

$$\begin{aligned} l(D; \mathbf{w}, \sigma^2) &= \log L(D; \mathbf{w}, \sigma^2) = \sum_{i=1}^n \log P(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2) \\ &= \sum_{i=1}^n \left(-\frac{1}{2\sigma^2} (y_i - f(\mathbf{x}_i; \mathbf{w}))^2 - \log \sqrt{2\pi\sigma^2} \right) \\ &= \left(-\frac{1}{2\sigma^2} \right) \sum_{i=1}^n (y_i - f(\mathbf{x}_i; \mathbf{w}))^2 + \dots \end{aligned}$$

Maximum likelihood estimation (cont'd)

- Maximizing log-likelihood is equivalent to minimizing empirical loss when the loss is defined according to

$$\text{Loss}(y_i, f(\mathbf{x}_i; \mathbf{w})) = -\log P(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2)$$

Loss defined as the negative log-probability is known as the log-loss.

Maximum likelihood estimation (cont'd)

- The log-likelihood of observations

$$\log L(D; \mathbf{w}, \sigma^2) = \sum_{i=1}^n \log P(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2)$$

is a generic fitting criterion and can be used to estimate the noise variance σ^2 as well.

- Let $\hat{\mathbf{w}}$ be the maximum likelihood (here least squares) setting of the parameters. What is the maximum likelihood estimate of σ^2 , obtained by solving

$$\frac{\partial}{\partial \sigma^2} \log L(D; \mathbf{w}, \sigma^2) = 0 \quad ?$$

Maximum likelihood estimation (cont'd)

- The log-likelihood of observations

$$\log L(D; \mathbf{w}, \sigma^2) = \sum_{i=1}^n \log P(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2)$$

is a generic fitting criterion and can be used to estimate the noise variance σ^2 as well.

- Let $\hat{\mathbf{w}}$ be the maximum likelihood (here least squares) setting of the parameters. The maximum likelihood estimate of the noise variance σ^2 is

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

i.e., the mean squared prediction error.

Polynomial regression

- Consider again a simple m^{th} degree polynomial regression model

$$y = w_0 + w_1x + \dots + w_mx^m + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

where σ^2 is assumed fixed (known).

- In this model the outputs $\{y_1, \dots, y_n\}$ corresponding to any inputs $\{x_1, \dots, x_n\}$ are generated according to

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \mathbf{e}, \quad \text{where}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \dots \\ y_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_1 & \dots & x_1^m \\ \dots & \dots & \dots & \dots \\ 1 & x_n & \dots & x_n^m \end{bmatrix}, \quad \mathbf{e} = \begin{bmatrix} \epsilon_1 \\ \dots \\ \epsilon_n \end{bmatrix}$$

and $\epsilon_i \sim N(0, \sigma^2)$, $i = 1, \dots, n$.

ML estimator, uncertainty

- We are interested in studying how the choice of inputs $\{x_1, \dots, x_n\}$ or, equivalently, $\mathbf{X}$, affects the accuracy of our regression model
- Our model for the outputs $\{y_1, \dots, y_n\}$ given $\mathbf{X}$ is

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \mathbf{e}, \quad \mathbf{e} \sim N(\mathbf{0}, \sigma^2 \mathbf{I})$$

- We assume also that the training outputs are actually generated by a model in this class with some fixed but unknown parameters $\mathbf{w}^*$ (same σ^2)

$$\mathbf{y} = \mathbf{X}\mathbf{w}^* + \mathbf{e}, \quad \mathbf{e} \sim N(\mathbf{0}, \sigma^2 \mathbf{I})$$

- We can now ask, for a given $\mathbf{X}$, how accurately we are able to recover the "true" parameters $\mathbf{w}^*$

ML estimator, uncertainty

- The ML estimator $\hat{\mathbf{w}}$ viewed here as a function of the outputs $\mathbf{y}$ for a fixed $\mathbf{X}$, is given by

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- We need to understand how $\hat{\mathbf{w}}$ varies in relation to $\mathbf{w}^*$ when the outputs are generated according to

$$\mathbf{y} = \mathbf{X}\mathbf{w}^* + \mathbf{e}, \quad \mathbf{e} \sim N(\mathbf{0}, \sigma^2 \mathbf{I})$$

- In the absence of noise $\mathbf{e}$, the ML estimator would recover $\mathbf{w}^*$ exactly (with only minor constraints on $\mathbf{X}$)

$$\begin{aligned} \hat{\mathbf{w}} &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{X}\mathbf{w}^*) \\ &= (\mathbf{X}^T \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{X}) \mathbf{w}^* \\ &= \mathbf{w}^* \end{aligned}$$

ML estimator, uncertainty

- In the presence of noise we can still use the fact that $\mathbf{y} = \mathbf{X}\mathbf{w}^* + \mathbf{e}$ to simplify the parameter estimates

$$\begin{aligned}\hat{\mathbf{w}} &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \\ &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{X} \mathbf{w}^* + \mathbf{e}) \\ &= (\mathbf{X}^T \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{X}) \mathbf{w}^* + (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{e} \\ &= \mathbf{w}^* + (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{e}\end{aligned}$$

- So the ML estimate is the correct parameter vector plus an estimate based purely on noise

Recap: Lecture overview

- what is learning?
- types of machine learning problems
- image classification
- linear regression
- generalization
- cross-validation
- maximum likelihood estimation

Next Lecture:
Multi-layer Perceptrons