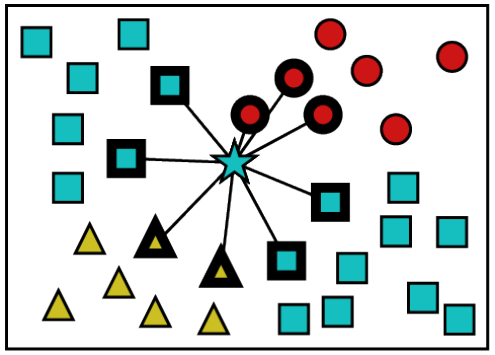
Machine Learning

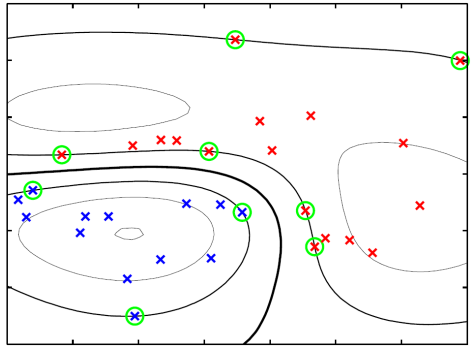
Computer Vision (UW EE/CSE 576)



Richard Szeliski

Facebook & UW

Lecture 6 – Apr 16, 2020



Class calendar

Date	Lecture	Reading	Homework
March 31	Introduction	Szeliski, Chapter 1	
April 2	Human Vision, Color Spaces and Transforms	Brown, M. S. (2019). ICCV 2019 tutorial on understanding color and the in-camera image processing pipeline for computer vision.	HW1 assigned
April 7	Image coordinates, resizing	Szeliski, Chapter 2.1, 3.6	
April 9	Filters and convolutions	Szeliski, Chapter 3	HW1 due, HW2 assigned
April 14	Interpolation and Optimization	Szeliski, Chapter 4	
April 16	Machine Learning	Szeliski, Chapter 5.1-5.2	

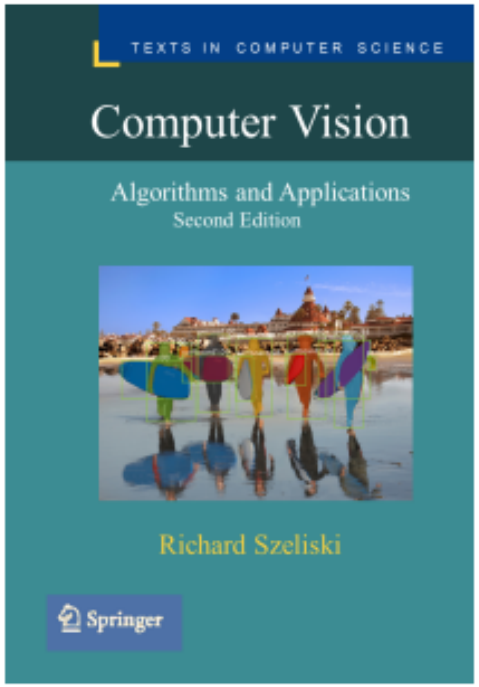
Readings

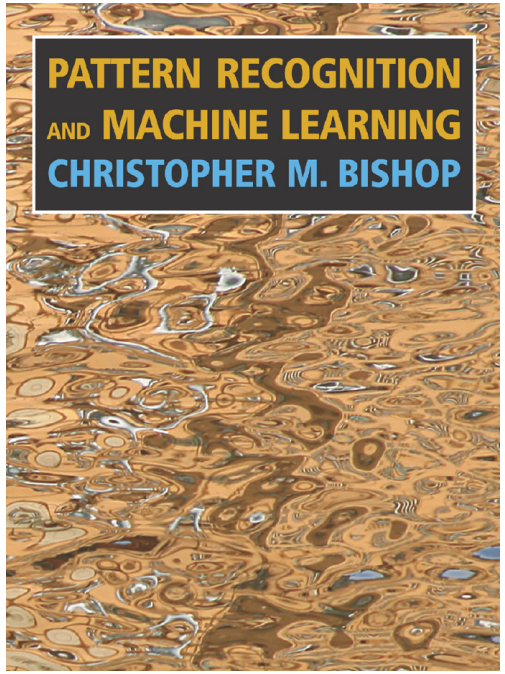
Chapter 5

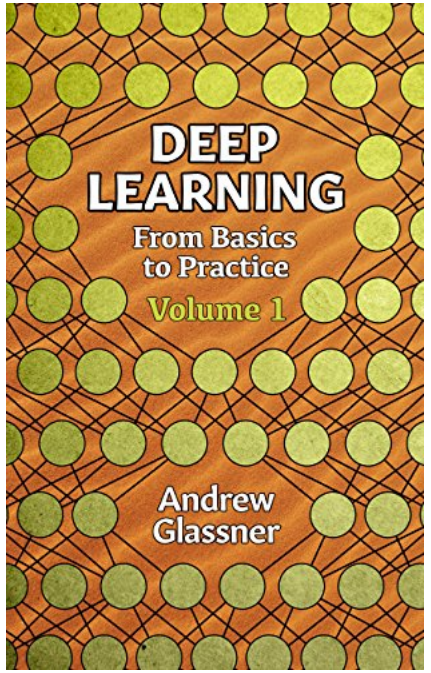
Deep Learning

5.1 Supervised learning 239
5.1.1 Nearest neighbors 241
5.1.2 Bayesian classification 243
5.1.3 Logistic regression 247
5.1.4 Support vector machines 249
5.1.5 Decision trees and forests 251
5.2 Unsupervised learning 254
5.2.1 Clustering 254
5.2.2 K-means and mixtures of Gaussians 256
5.2.3 Principal component analysis 258
5.2.4 Semi-supervised learning 268
5.3 Deep neural networks 268

References

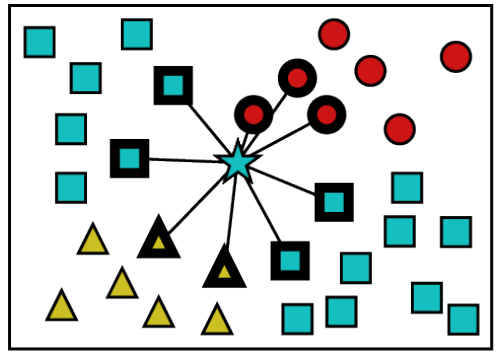

TEXTS IN COMPUTER SCIENCE
Computer Vision
Algorithms and Applications
Second Edition
Richard Szeliski
Springer

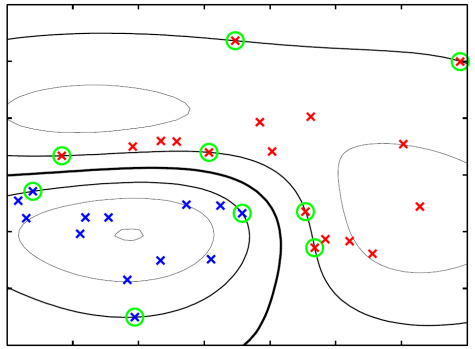

PATTERN RECOGNITION
AND MACHINE LEARNING
CHRISTOPHER M. BISHOP


DEEP
LEARNING
From Basics
to Practice
Volume 1
Andrew
Glassner

Traditional machine learning

- Why learning?
- Nearest neighbors
- Bayesian classification
- Logistic regression
- Support vector machines
- Clustering
- Principal component analysis





… finish off previous lecture …

Why Bayesian (probabilistic) modeling?

- Merge uncertain measurement in an optimal way
- Estimate the uncertainty in the final answer
- Build in (quantitative) prior assumptions

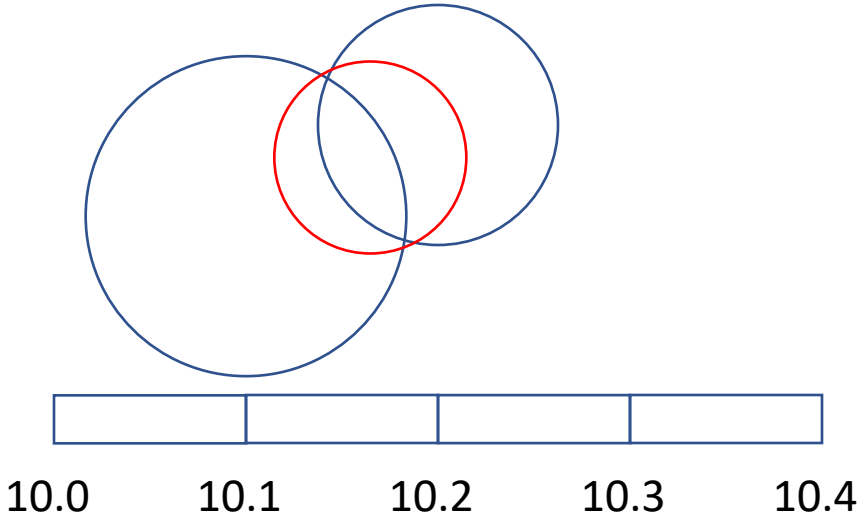
Appendix B

Bayesian modeling and inference

B.1 Estimation theory . . . 877
B.1.1 Likelihood for multivariate Gaussian noise . . . 877
B.2 Maximum likelihood estimation and least squares . . . 879
B.3 Robust statistics . . . 880
B.4 Prior models and Bayesian inference . . . 883
B.5 Markov random fields . . . 884
B.5.1 Gradient descent and simulated annealing . . . 886
B.5.2 Dynamic programming . . . 888
B.5.3 Belief propagation . . . 890
B.5.4 Graph cuts . . . 893
B.5.5 Linear programming . . . 896
B.6 Uncertainty estimation (error analysis) . . . 897

Example: merge GPS readings

- *Measurement 1*: 10.1 ± 0.1667 (1/6) Gaussian noise
- *Measurement 2*: 10.2 ± 0.1250 (1/8) Gaussian noise
- What is the optimal combined measurement?



Exercise for next lecture

(send me your answer on Slack)

Answer: use inverse variances (Appendix B)

$$\mathbf{y}_i = \mathbf{f}_i(\mathbf{x}) + \mathbf{n}_i$$

$$L = p(\mathbf{y}|\mathbf{x}) = \prod_i p(\mathbf{y}_i|\mathbf{x}) = \prod_i p(\mathbf{y}_i|\mathbf{f}_i(\mathbf{x}))$$

$$= \prod_i |2\pi\mathbf{\Sigma}_i|^{-1/2} \exp\left(-\frac{1}{2}\|\mathbf{y}_i - \mathbf{f}_i(\mathbf{x})\|^2_{\mathbf{\Sigma}_i^{-1}}\right)$$

$$E = -\log L = \frac{1}{2}\sum_i (\mathbf{y}_i - \mathbf{f}_i(\mathbf{x}))^T \mathbf{\Sigma}_i^{-1} (\mathbf{y}_i - \mathbf{f}_i(\mathbf{x})) + k$$

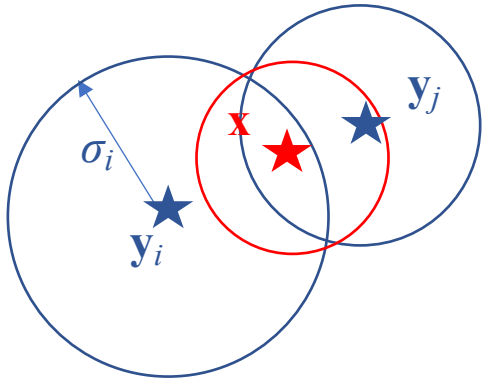
$$\mathbf{f}_i(\mathbf{x}) = \mathbf{H}_i\mathbf{x}$$

$$\mathbf{C}_i = \mathbf{\Sigma}_i^{-1}$$

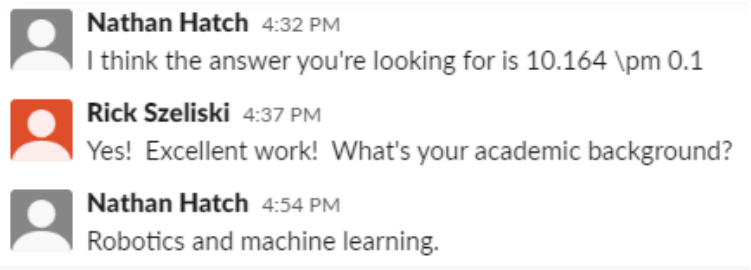
$$E = \sum_i \|\tilde{\mathbf{y}}_i - \mathbf{H}_i\mathbf{x}\|_{\mathbf{\Sigma}_i^{-1}} = \sum_i (\tilde{\mathbf{y}}_i - \mathbf{H}_i\mathbf{x})^T \mathbf{C}_i (\tilde{\mathbf{y}}_i - \mathbf{H}_i\mathbf{x}), \qquad \text{(B.16)}$$

$$\mathbf{H}_i = \mathbf{I}$$

$$\mathbf{x} = \left(\sum_i \mathbf{C}_i\right)^{-1} \left(\sum_i \mathbf{C}_i\tilde{\mathbf{y}}_i\right)$$

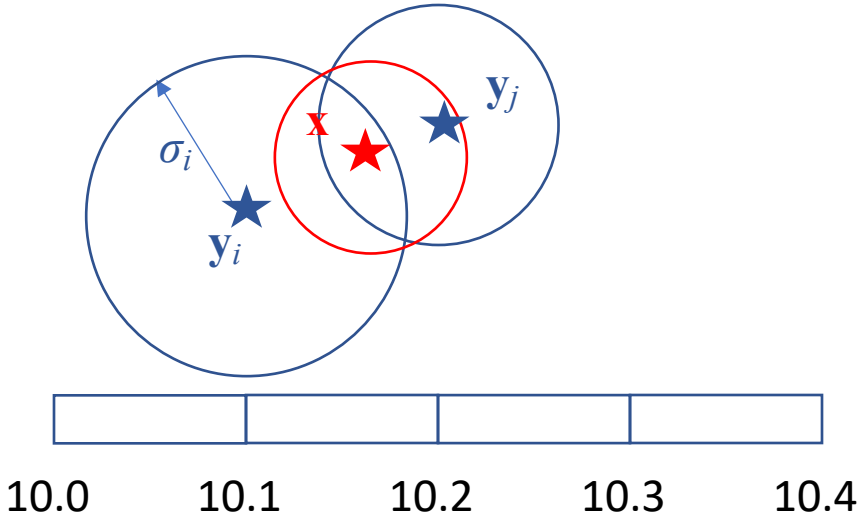


Answer: merge GPS readings



- *Measurement 1*: 10.1 ± 0.1667 (1/6) Gaussian noise
- *Measurement 2*: 10.2 ± 0.1250 (1/8) Gaussian noise
- What is the optimal combined measurement?

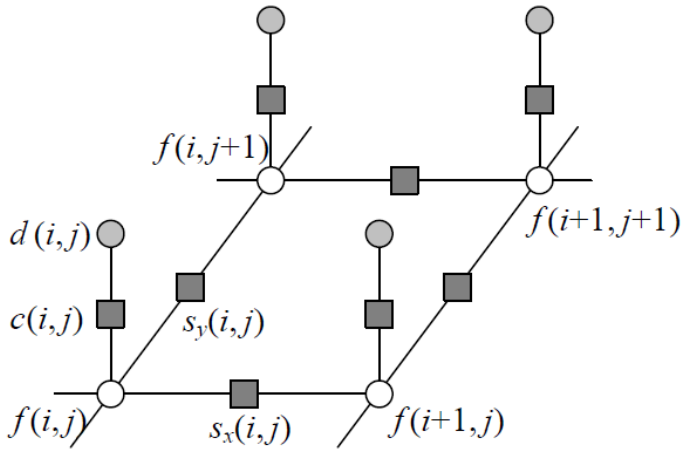
$$\mathbf{x} = \left(\sum_i \mathbf{C}_i \right)^{-1} \left(\sum_i \mathbf{C}_i \tilde{\mathbf{y}}_i \right)$$



Measurement	value	std.dev.	variance	inv.var.	wght. contr.
1	10.1	0.166667	0.02778	36	363.6
2	10.2	0.125	0.01563	64	652.8
summed	10.16	0.1	0.01	100	1016

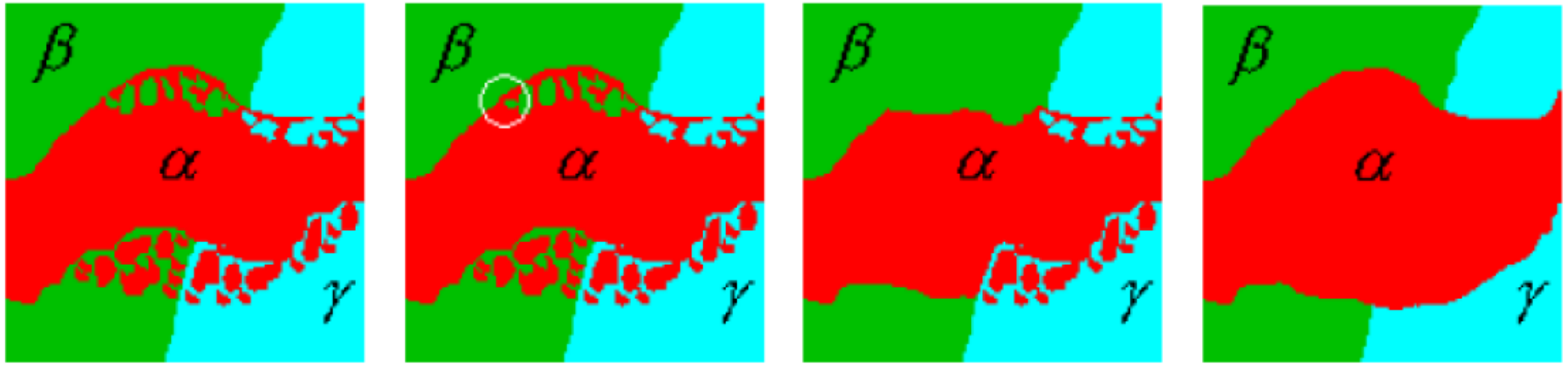
Why a Bayesian formulation?

- Wider range of probability distributions
- Learn distributions from data
- Wider range of inference algorithms [Kappes, Andres, *et al.*, IJCV 2015]





Energy minimization



(a) initial labeling (b) standard move (c) α-β-swap (d) α-expansion

Figure 4.15 *Multi-level graph optimization from (Boykov, Veksler, and Zabih 2001) © 2001 IEEE: (a) initial problem configuration; (b) the standard move only changes one pixel; (c) the α-β-swap optimally exchanges all α and β-labeled pixels; (d) the α-expansion move optimally selects among current pixel values and the α label.*

Digital Photomontage



Figure 4.17 *An unordered label MRF (Agarwala, Dontcheva, Agrawala* et al. *2004)* © *2004 ACM: Strokes in each of the source images on the left are used as constraints on an MRF optimization, which is solved using graph cuts. The resulting multi-valued label field is shown as a color overlay in the middle image, and the final composite is shown on the right.*

Markov Random Fields

According to Bayes' Rule (Appendix B.4), the *posterior* distribution for a given set of measurements $\mathbf{y}$, $p(\mathbf{y}|\mathbf{x})$, combined with a prior $p(\mathbf{x})$ over the unknowns $\mathbf{x}$, is given by

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}, \tag{4.33}$$

where $p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$ is a normalizing constant used to make the $p(\mathbf{x}|\mathbf{y})$ distribution *proper* (integrate to 1). Taking the negative logarithm of both sides of (4.33), we get

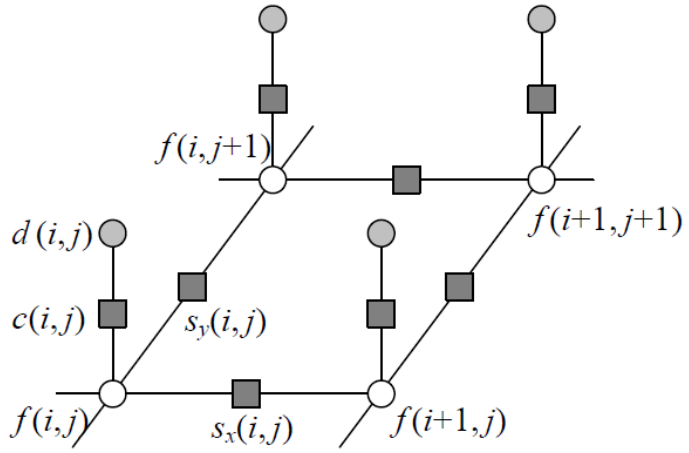
$$-\log p(\mathbf{x}|\mathbf{y}) = -\log p(\mathbf{y}|\mathbf{x}) - \log p(\mathbf{x}) + C, \tag{4.34}$$

which is the *negative posterior log likelihood*.

To find the most likely (*maximum a posteriori* or MAP) solution $\mathbf{x}$ given some measurements $\mathbf{y}$, we simply minimize this negative log likelihood, which can also be thought of as an *energy*,

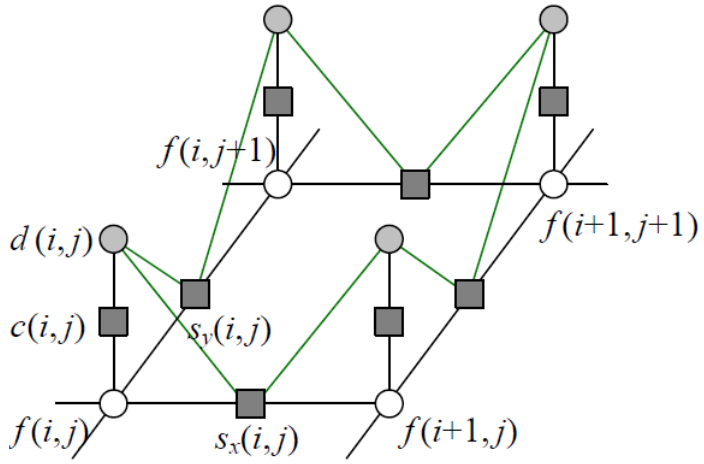
$$E(\mathbf{x}, \mathbf{y}) = E_{\mathrm{D}}(\mathbf{x}, \mathbf{y}) + E_{\mathrm{P}}(\mathbf{x}). \tag{4.35}$$

(We drop the constant C because its value does not matter during energy minimization.) The first term $E_{\mathrm{D}}(\mathbf{x}, \mathbf{y})$ is the *data energy* or *data penalty*; it measures the negative log likelihood that the data were observed given the unknown state $\mathbf{x}$. The second term $E_{\mathrm{P}}(\mathbf{x})$ is the *prior energy*; it plays a role analogous to the smoothness energy in regularization. Note



Conditional Random Fields

- Interaction potentials depend on guide image (sound familiar?)



Since the smoothness term now depends on the data, Bayes' Rule (4.44) no longer applies. Instead, we use a direct model for the posterior distribution $p(\mathbf{x}|\mathbf{y})$, whose negative log likelihood can be written as

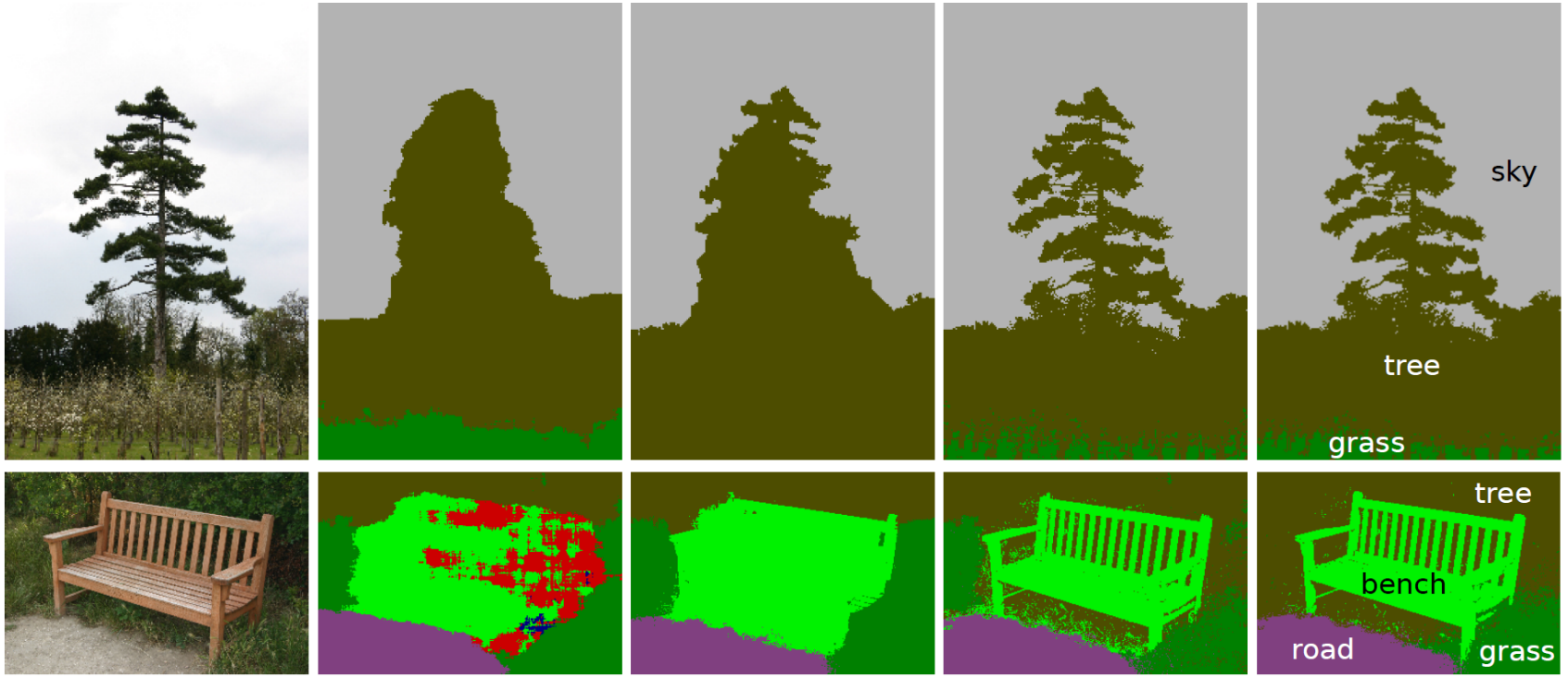
$$E(\mathbf{x}|\mathbf{y}) = E_{\mathrm{D}}(\mathbf{x},\mathbf{y}) + E_{\mathrm{S}}(\mathbf{x},\mathbf{y})$$
$$= \sum_p V_p(x_p,\mathbf{y}) + \sum_{(p,q)\in\mathcal{N}} V_{p,q}(x_p,x_q,\mathbf{y}), \qquad (4.45)$$

using the notation introduced in (4.43). The resulting probability distribution is called a *conditional random field* (CRF) and was first introduced to the computer vision field by Kumar and Hebert (2003), based on earlier work in text modeling by Lafferty, McCallum, and Pereira (2001).

Figure 4.18 shows a graphical model where the smoothness terms depend on the data values. In this particular model, each smoothness term depends only on its adjacent pair of data values, i.e., terms are of the form $V_{p,q}(x_p,x_q,y_p,y_q)$ in (4.45).

Dense CRF [Krähenbühl and Koltun 2011]

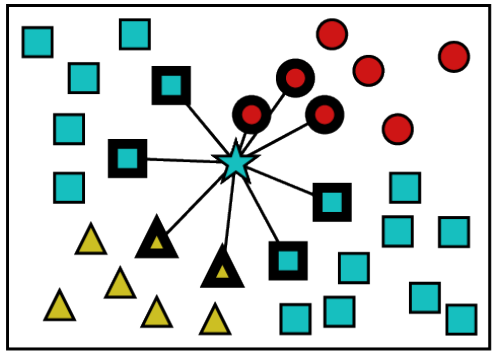
- Use a wide color-based support region (like bilateral solver)

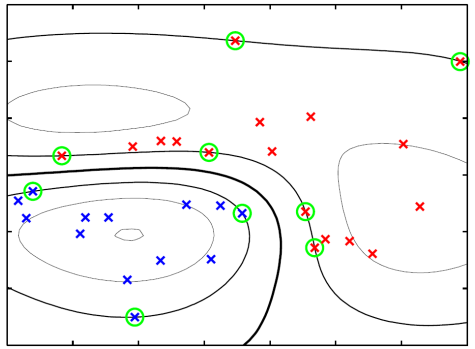


(a) Image (b) Unary classifiers (c) Robust P^n CRF (d) Fully connected CRF, MCMC inference, 36 hrs (e) Fully connected CRF, our approach, 0.2 seconds

Traditional machine learning (today's lecture)

- Why learning?
- Nearest neighbors
- Bayesian classification
- Logistic regression
- Support vector machines
- Clustering
- Principal component analysis





I'll be borrowing slides from



UNIVERSITY OF MICHIGAN

EECS 498-007 / 598-005
Deep Learning for Computer Vision
Fall 2019

Course Description

Computer Vision has become ubiquitous in our society, with applications in search, image understanding, apps, mapping, medicine, drones, and self-driving cars. Core to many of these applications are visual recognition tasks such as image classification and object detection. Recent developments in neural network approaches have greatly advanced the performance of these state-of-the-art visual recognition systems. This course is a deep dive into details of neural-network based deep learning methods for computer vision. During this course, students will learn to implement, train and debug their own neural networks and gain a detailed understanding of cutting-edge research in computer vision. We will cover learning algorithms, neural network architectures, and practical engineering tricks for training and fine-tuning networks for visual recognition tasks.

Instructor

Justin Johnson

Graduate Student Instructors

Yunseok Jang

Kibok Lee

Luowei Zhou

Lecture 2:
Image Classification

Image Classification

Input: image

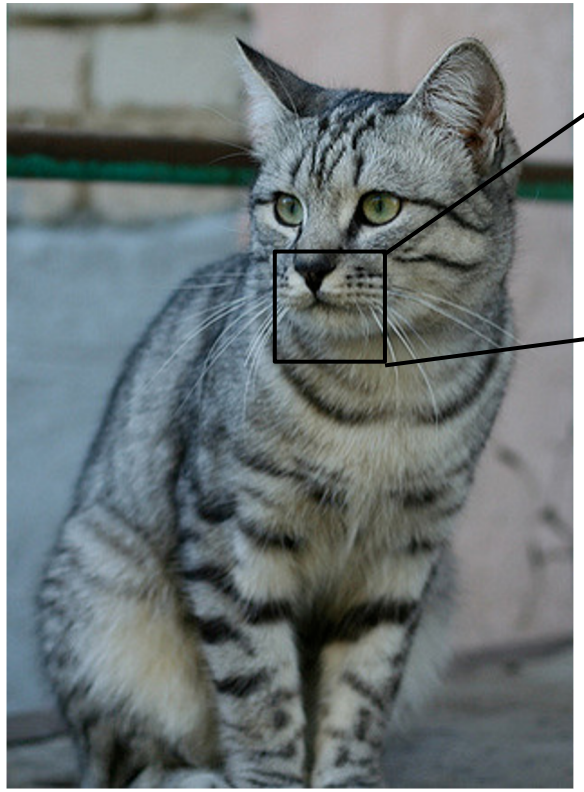


This image by Nikita is licensed under CC-BY 2.0

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

cat
bird
deer
dog
truck

Problem: Semantic Gap



This image by Nikita is licensed under CC-BY 2.0

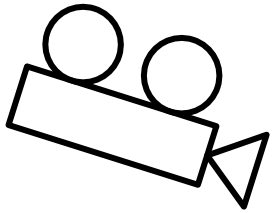
```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  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]
 [125 133 148 137 119 121 117  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]
 [130 128 134 161 139 100 109 118 121 134 114  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]]
```

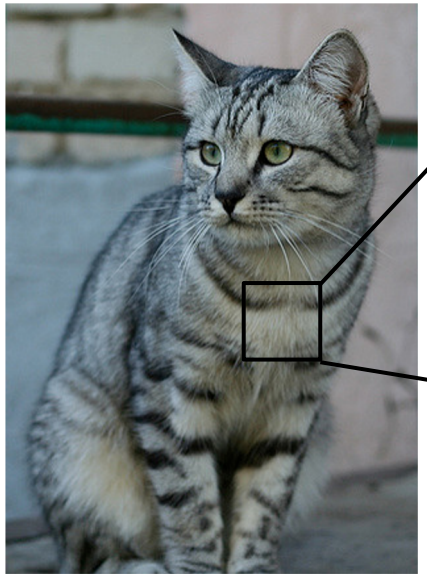
What the computer sees

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  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]
 [125 133 148 137 119 121 117  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]
 [130 128 134 161 139 100 109 118 121 134 114  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!

This image by Nikita is licensed under CC-BY 2.0

Challenges: Intraclass Variation



This image is CC0 1.0 public domain

Challenges: Fine-Grained Categories

Maine Coon



This image is free for for use under the Pixabay License

Ragdoll



This image is CC0 public domain

American Shorthair



This image is CC0 public domain

Challenges: Background Clutter



This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

Challenges: Illumination Changes



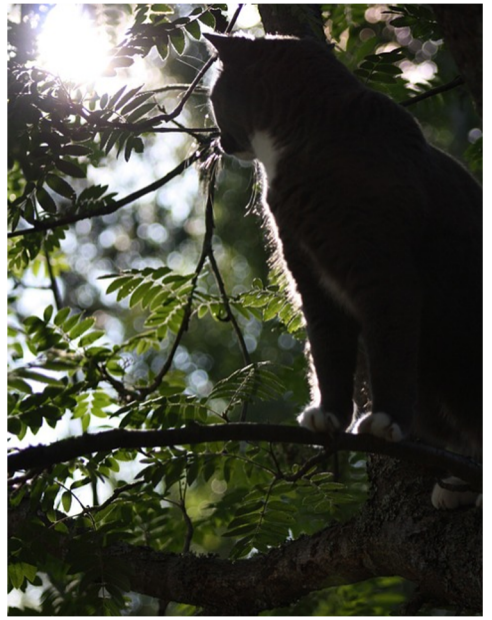
This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

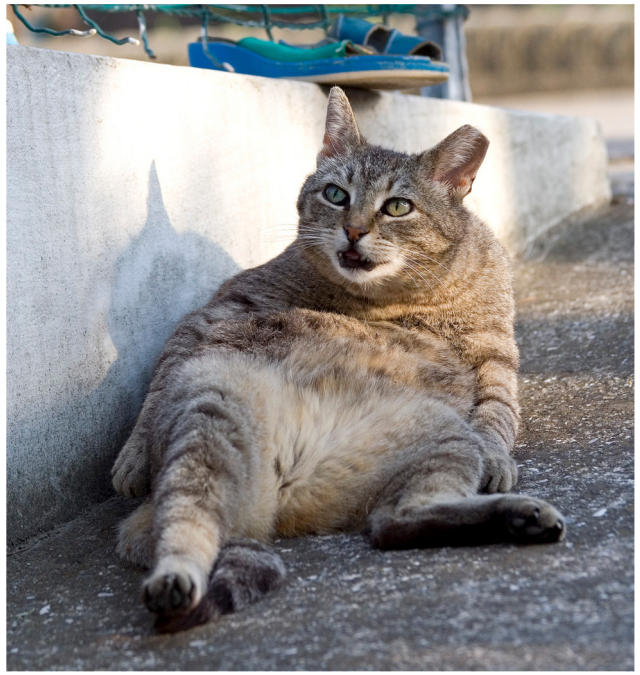


This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

Challenges: Deformation



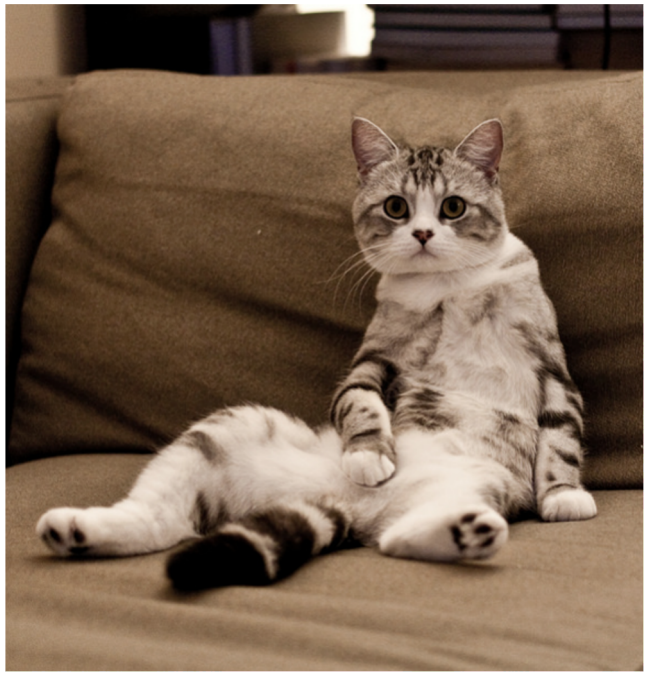
This image by Umberto Salvagnin is licensed under CC-BY 2.0



This image by Umberto Salvagnin is licensed under CC-BY 2.0



This image by sare bear is licensed under CC-BY 2.0



This image by Tom Thai is licensed under CC-BY 2.0

Challenges: Occlusion



This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

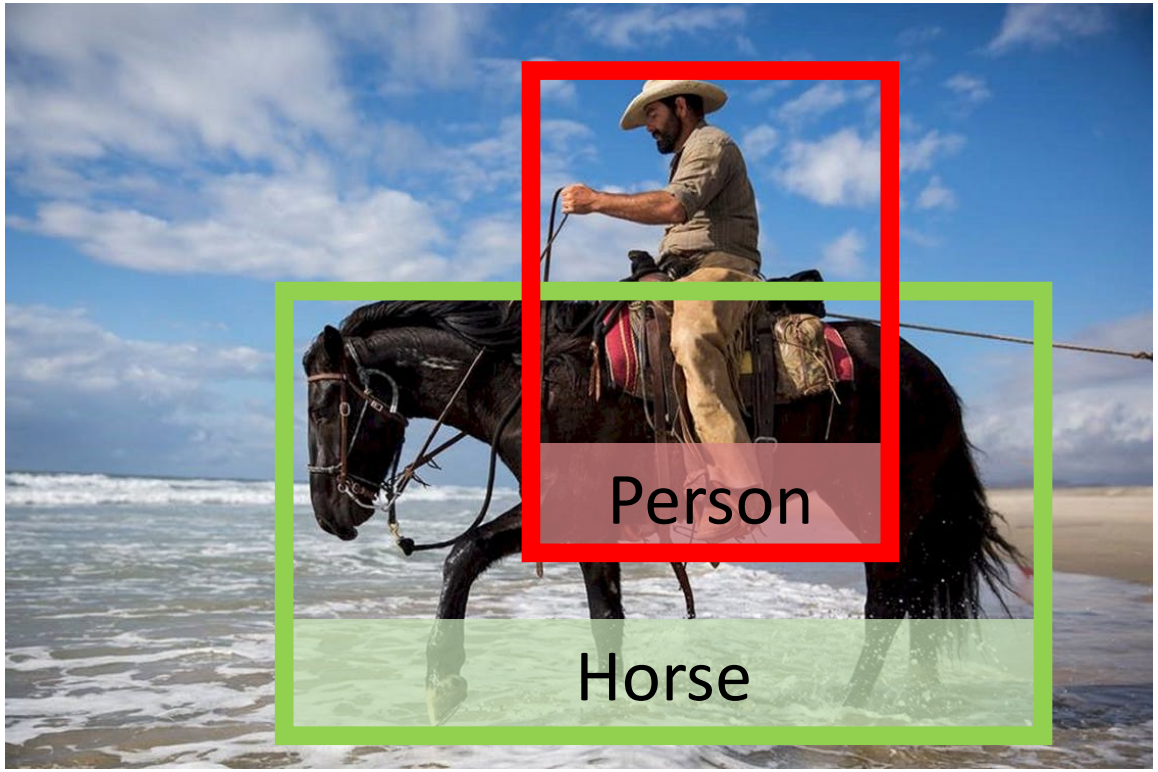


This image by jonsson is licensed under CC-BY 2.0

Image Classification: Building Block for other tasks!

Example: Object Detection





This image is free to use under the Pexels license

Image Classification: Building Block for other tasks!

Example: Object Detection



This image is free to use under the Pexels license

Background

Horse

Person

Car

Truck

Image Classification: Building Block for other tasks!

Example: Object Detection



This image is free to use under the Pexels license

→

Background
Horse
Person
Car
Truck

Image Classification: Building Block for other tasks!

Example: Image Captioning



This image is free to use under the Pexels license

riding
cat
horse
man
when
…
<STOP>

What word to say next?

Caption:
Man

Image Classification: Building Block for other tasks!

Example: Image Captioning



This image is free to use under the Pexels license



riding
cat
horse
man
when
…
<STOP>

What word to say next?

Caption:
Man riding

Image Classification: Building Block for other tasks!

Example: Image Captioning



This image is free to use under the Pexels license



riding
cat
horse
man
when
…
<STOP>

What word
to say next?

Caption:
Man riding horse

Image Classification: Building Block for other tasks!

Example: Image Captioning



This image is free to use under the Pexels license



riding
cat
horse
man
when
…
<STOP>

What word
to say next?

Caption:
Man riding horse

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.

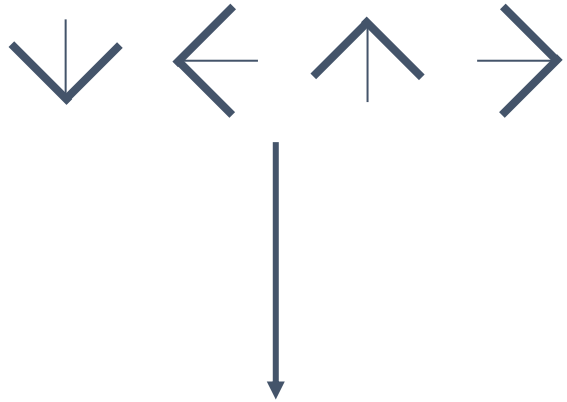
You could try …



Find edges



Find corners



?

John Canny, “A Computational Approach to Edge Detection”, IEEE TPAMI 1986

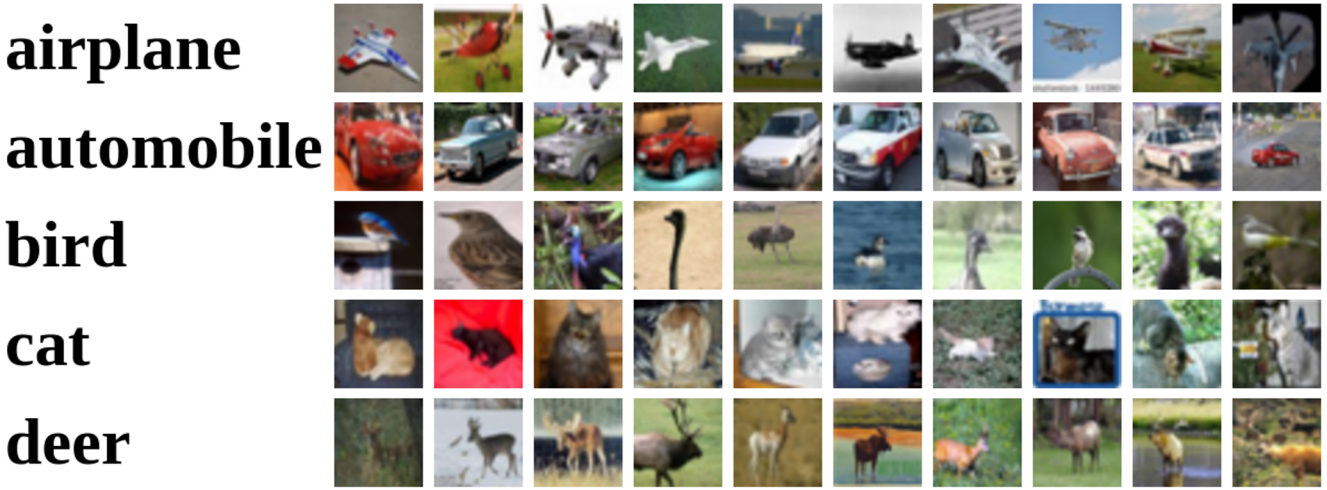
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

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

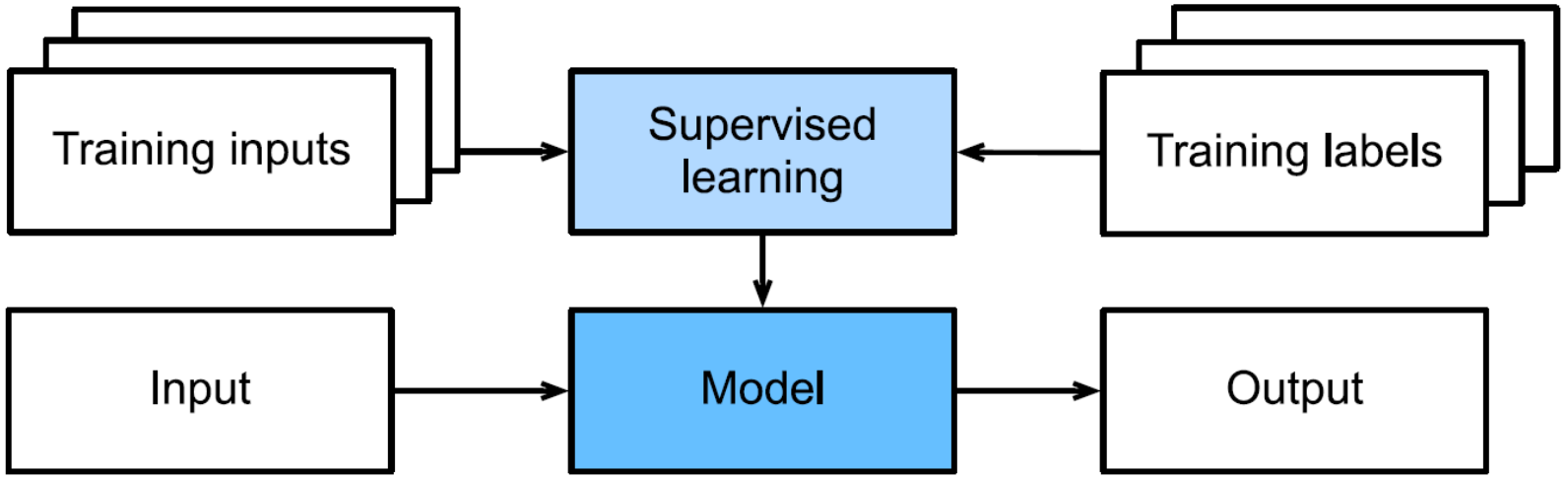
```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Example training set

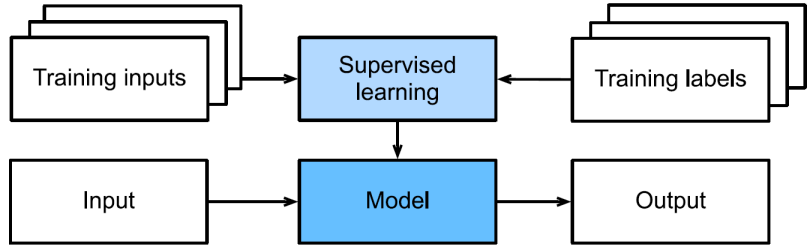


Supervised learning

- Goal: provide best output predictions for novel inputs



Supervised learning



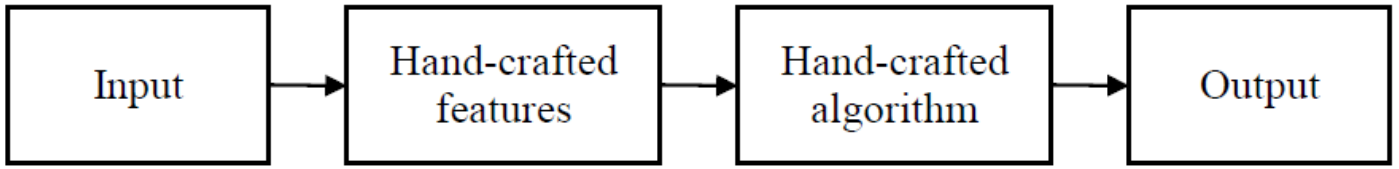
- Goal: provide best output predictions for novel inputs
- How can we predict future performance?
- Placeholder answer: minimize *empirical risk*

$$E_{\text{Risk}}(\mathbf{w}) = \frac{1}{N} \sum L(\mathbf{y}_i, \mathbf{f}(\mathbf{x}_i; \mathbf{w})). \tag{5.1}$$

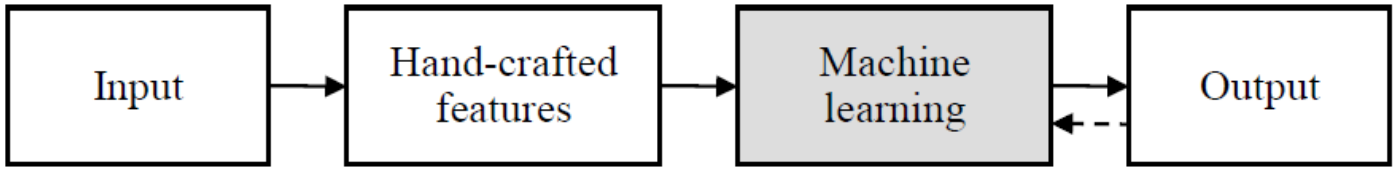
The loss function L measures the "cost" of predicting an output $\mathbf{f}(\mathbf{x}_i; \mathbf{w})$ for input $\mathbf{x}_i$ and model parameters $\mathbf{w}$ when the corresponding target is $\mathbf{y}_i$.

- Where have we seen this before?
- What are potential *models* and *loss functions*?

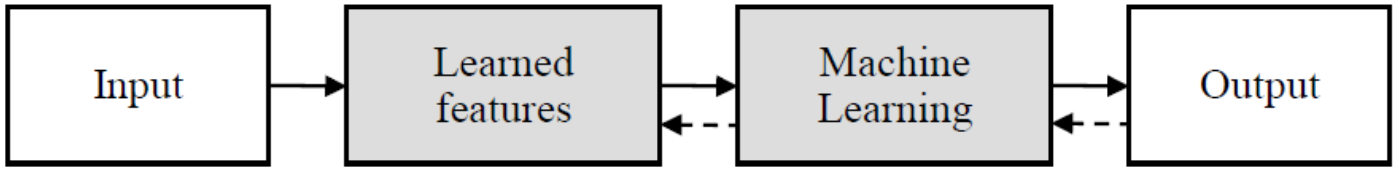
Traditional and deep learning



(a) Traditional vision pipeline



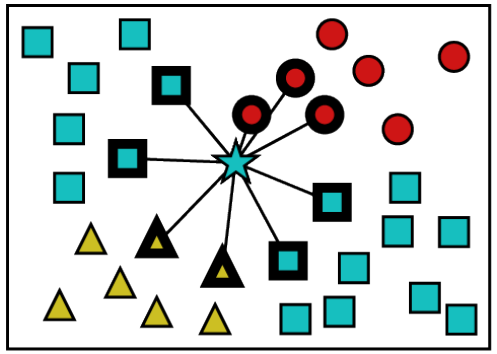
(b) Classic machine learning pipeline

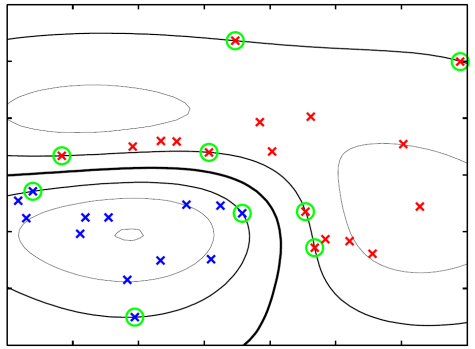


(c) Deep learning pipeline

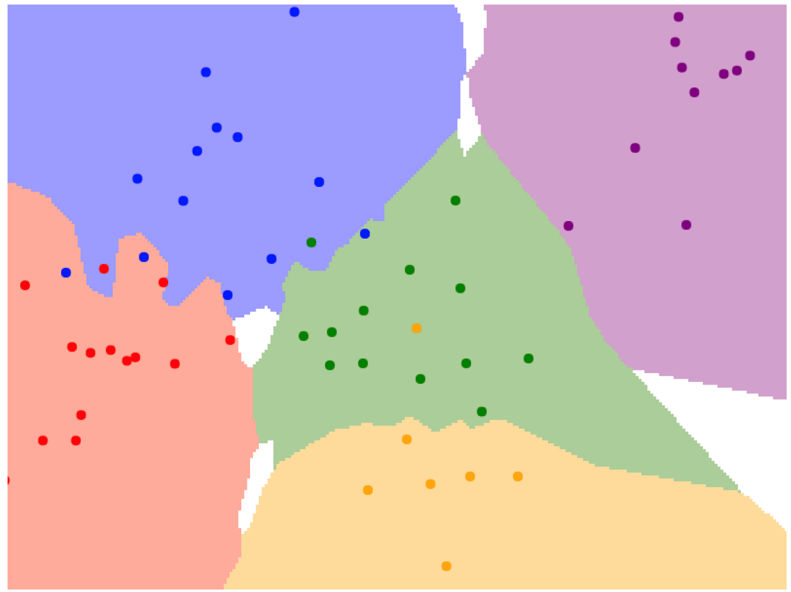
Traditional machine learning

- Why learning?
- Nearest neighbors
- Bayesian classification
- Logistic regression
- Support vector machines
- Clustering
- Principal component analysis





Nearest neighbors



First classifier: **Nearest Neighbor**

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


Memorize all data and labels

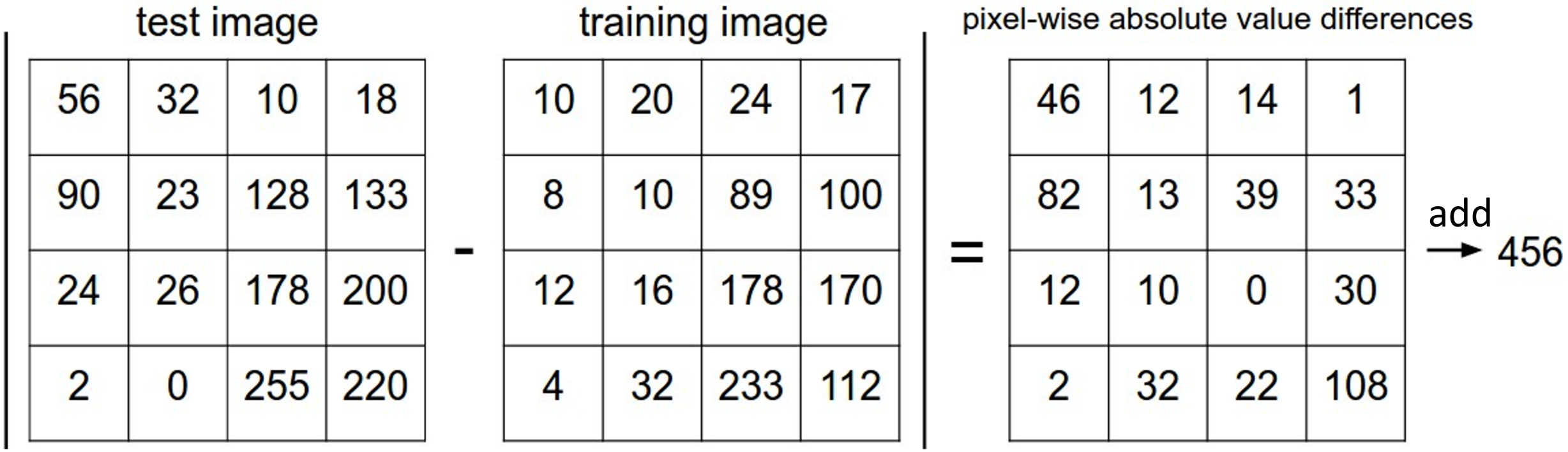
```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```


Predict the label of the most similar training image

Distance Metric to compare images

L1 distance:

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



Nearest Neighbor Classifier

```python
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

```python
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

```python
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 each test image:

Find nearest training image

Return label of nearest image

```python
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: With N examples, how fast is training?

```python
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: With N examples, how fast is training?

A: O(1)

```python
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: With N examples, how fast is training?

A: O(1)

Q: With N examples, how fast is testing?

```python
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: With N examples, how fast is training?

A: O(1)

Q: With N examples, how fast is testing?

A: O(N)

```python
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: With N examples, how fast is training?

A: O(1)

Q: With N examples, how fast is testing?

A: O(N)

This is **bad**: We can afford slow training, but we need fast testing!

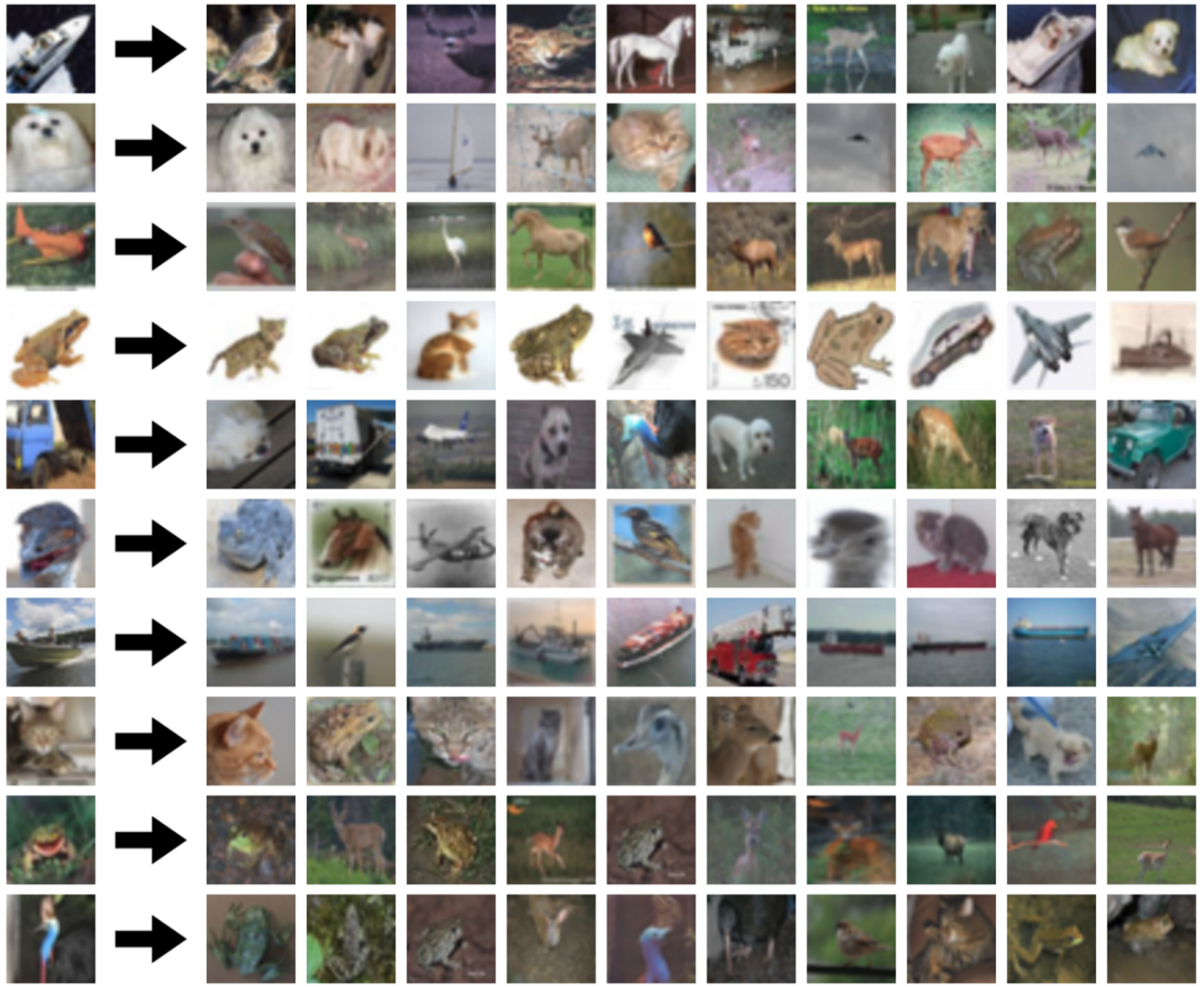
Nearest Neighbor Classifier

```python
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
```

There are many methods for fast / approximate nearest neighbors; e.g. see

https://github.com/facebookresearch/faiss

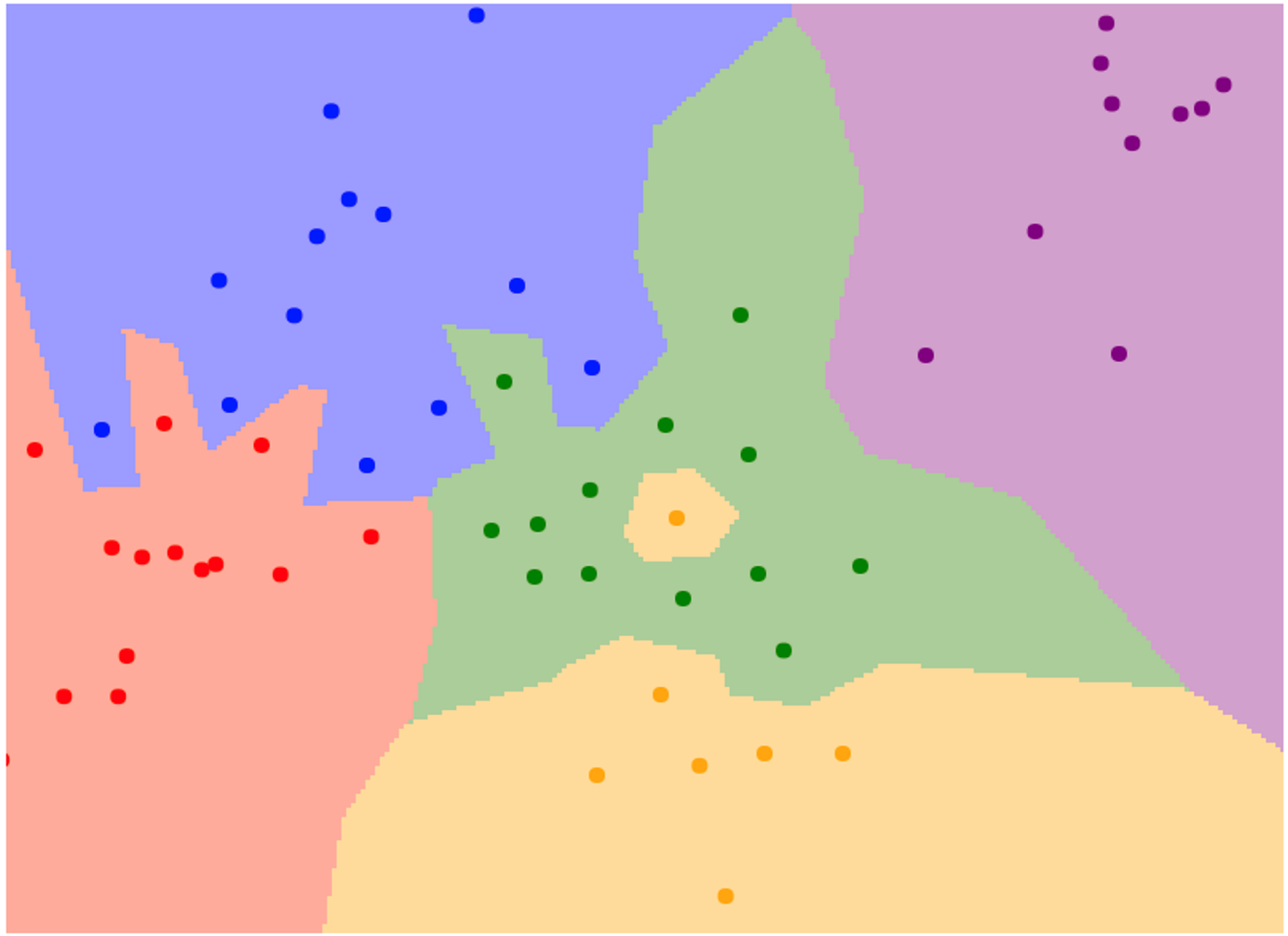
What does this look like?



What does this look like?

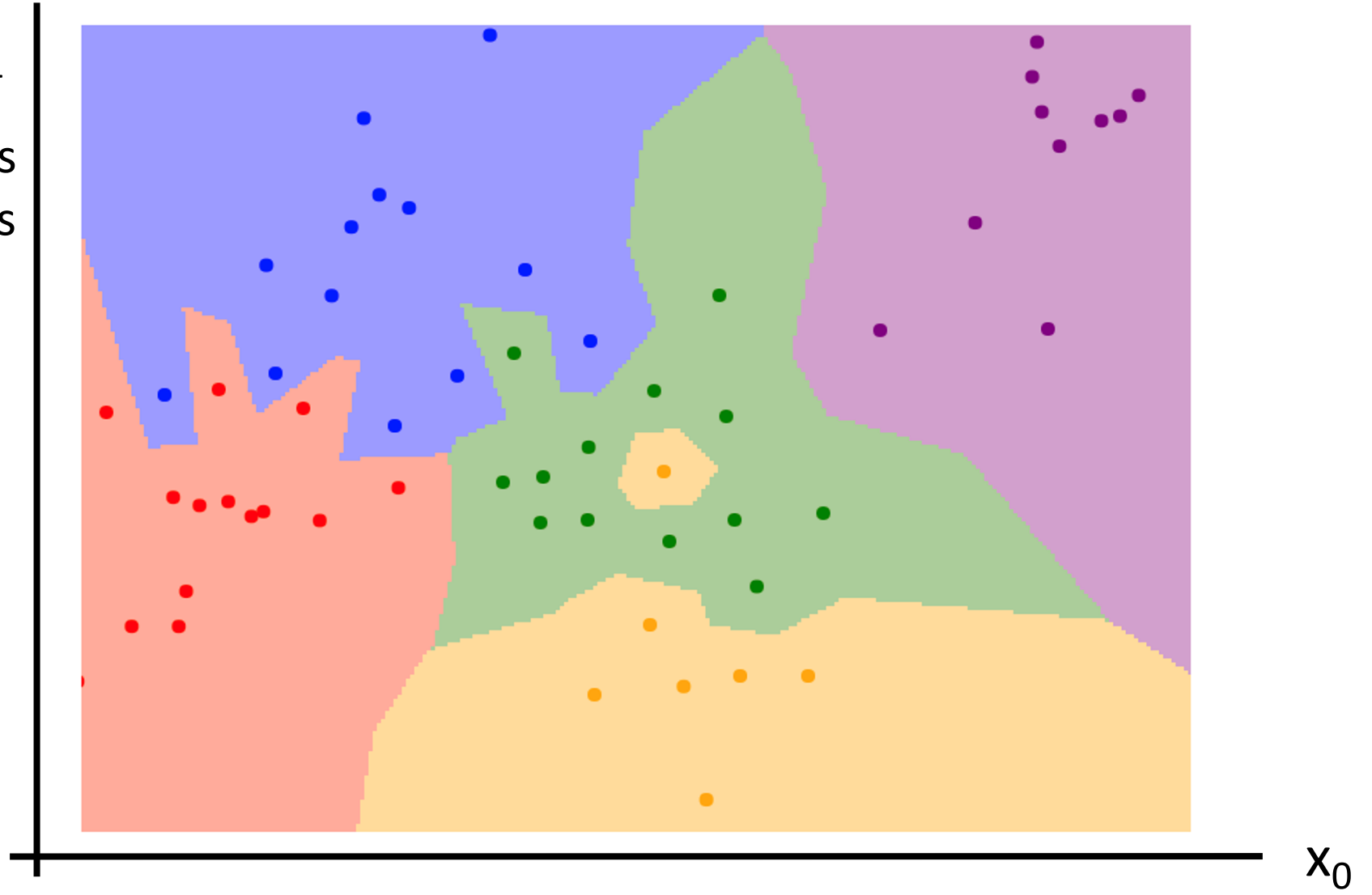


Nearest Neighbor **Decision Boundaries**



Nearest Neighbor **Decision Boundaries**

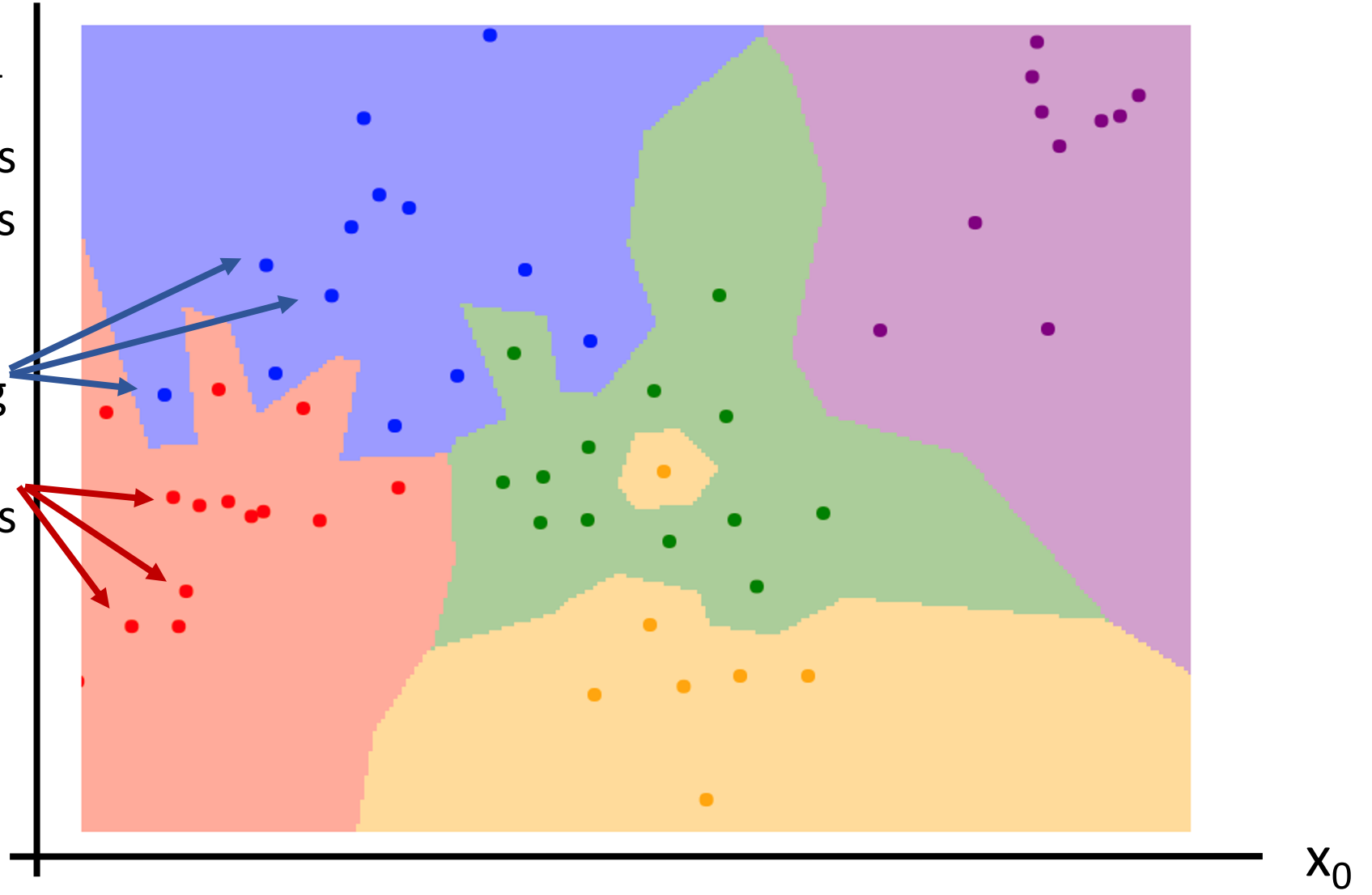
Nearest neighbors in two dimensions



Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

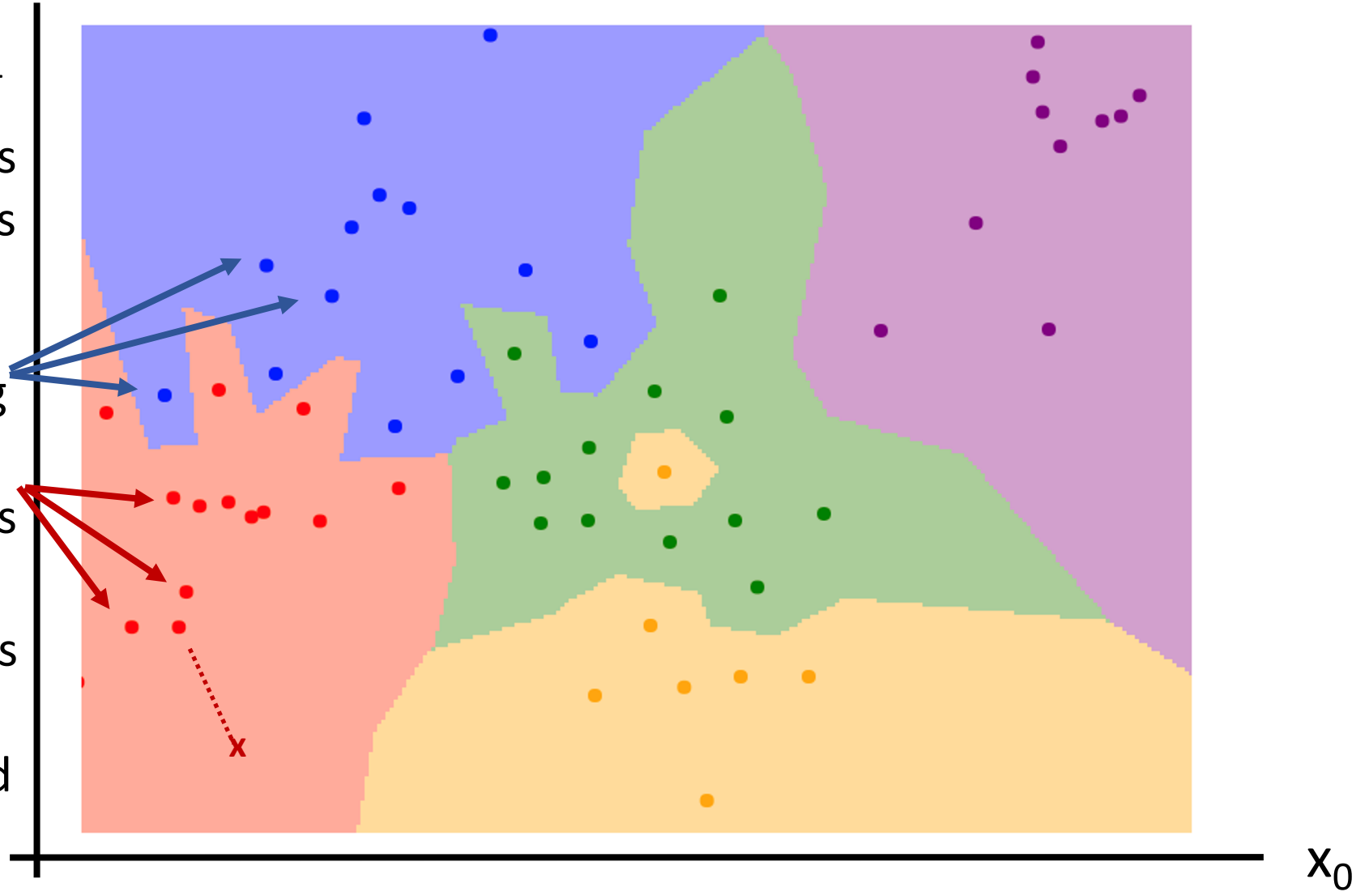


Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned

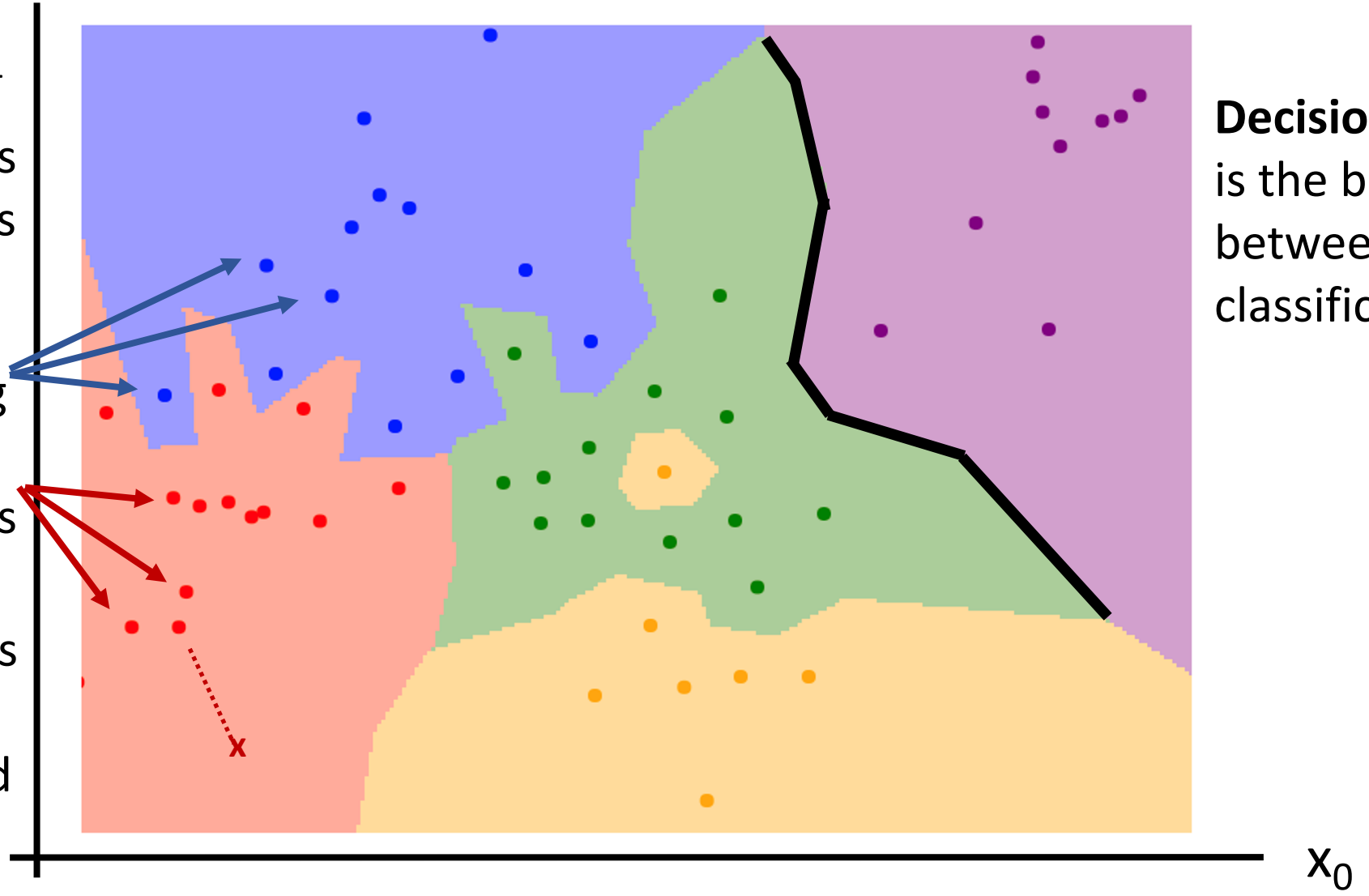


Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



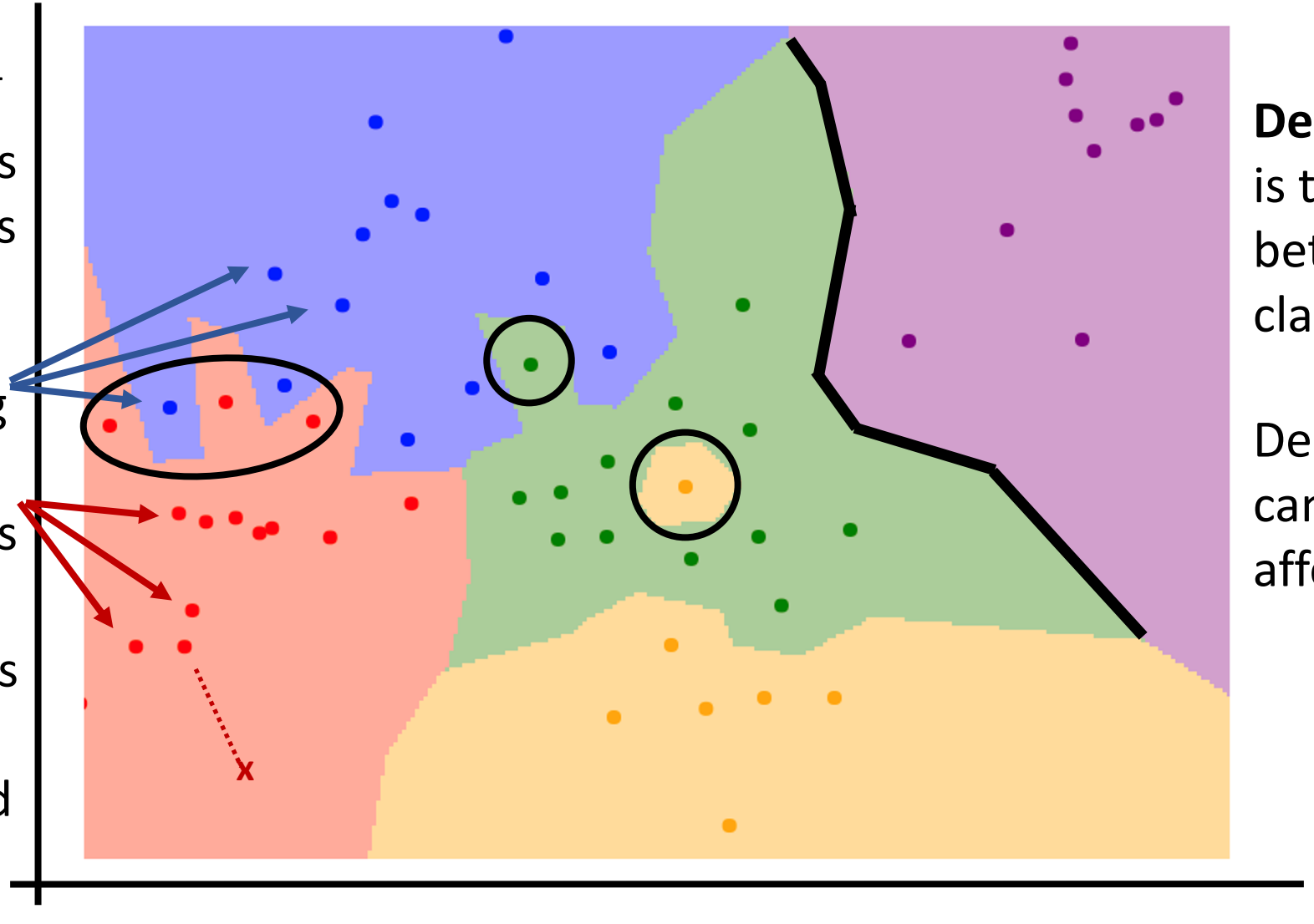
Decision boundary is the boundary between two classification regions

Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



Decision boundary is the boundary between two classification regions

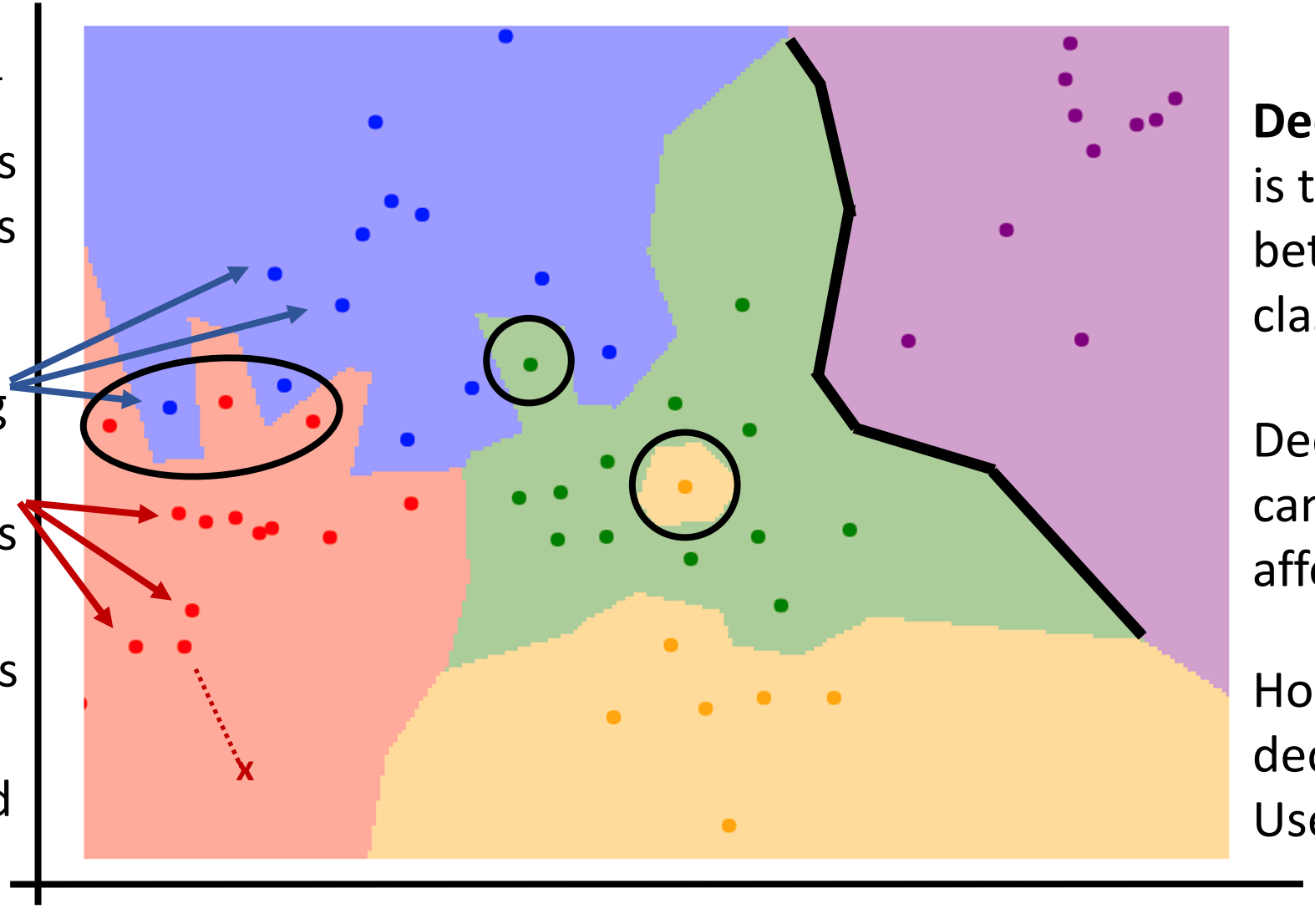
Decision boundaries can be noisy; affected by outliers

Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



Decision boundary is the boundary between two classification regions

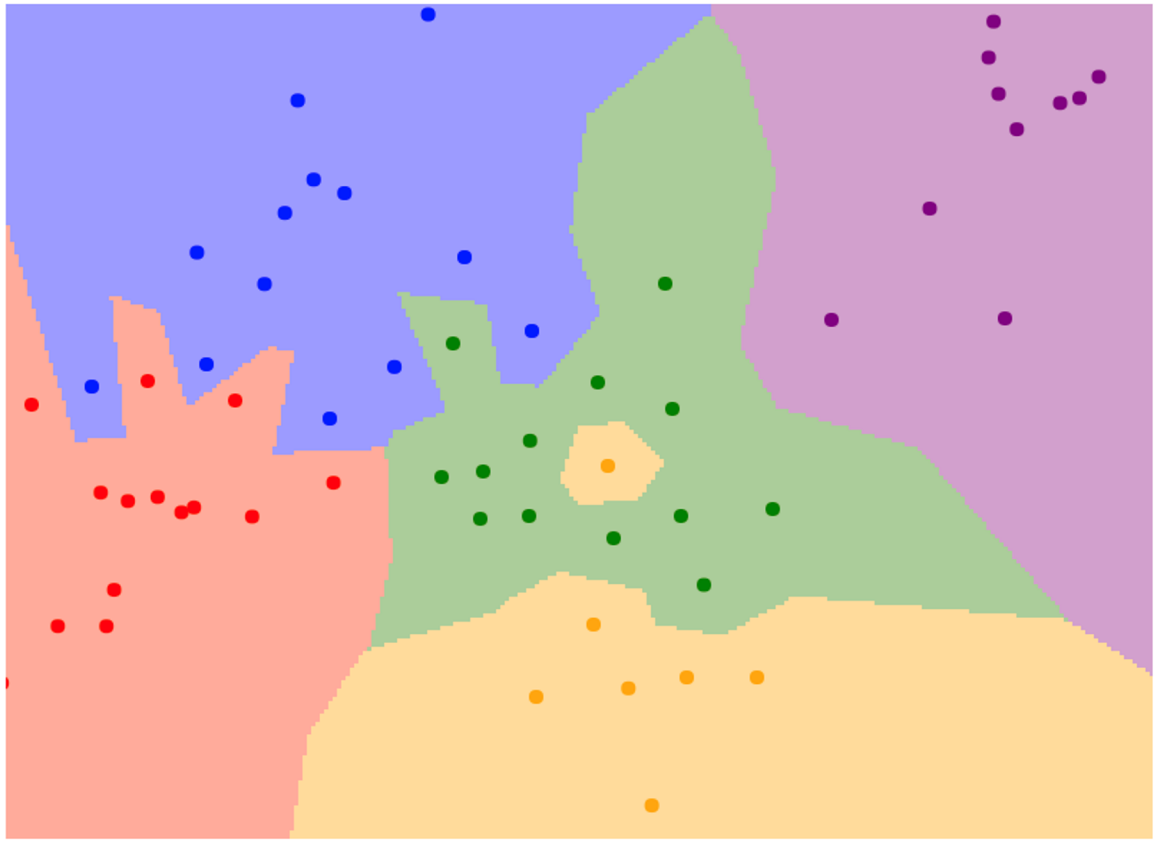
Decision boundaries can be noisy; affected by outliers

How to smooth out decision boundaries? Use more neighbors!

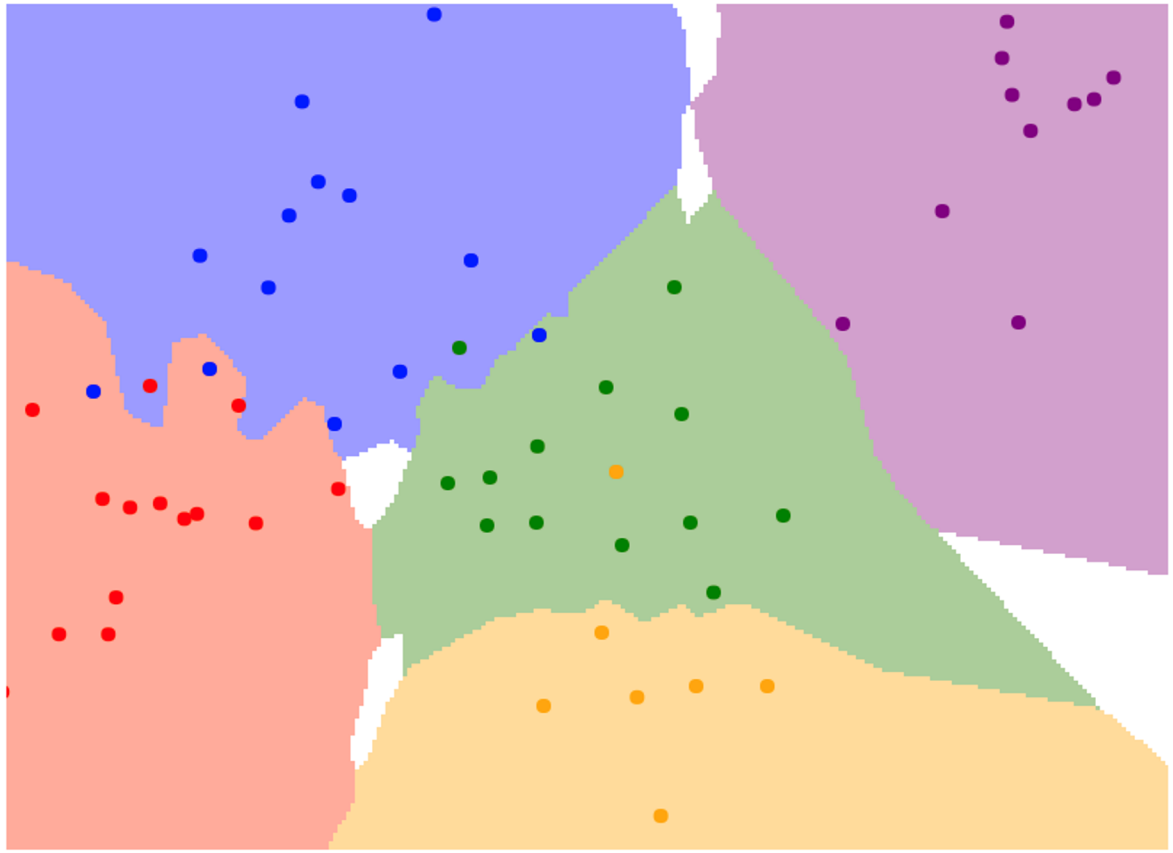
K-Nearest Neighbors

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

K = 1



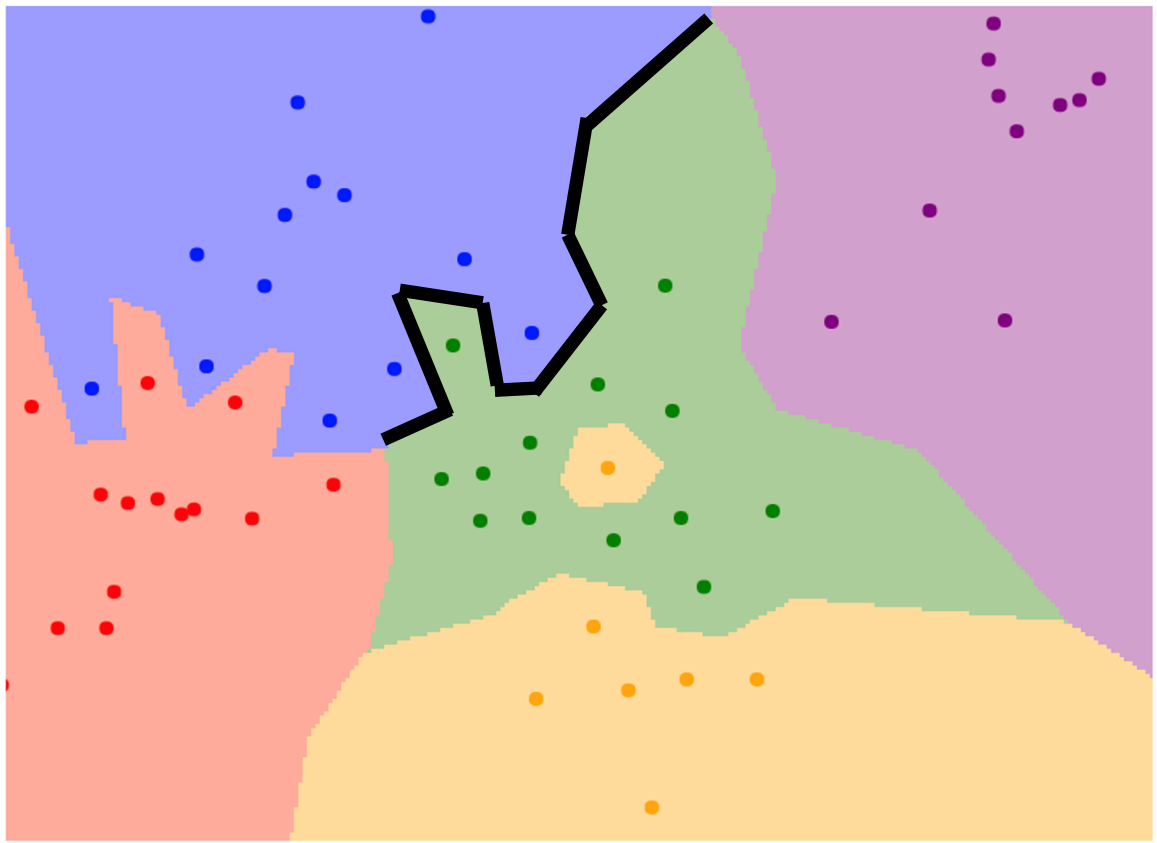
K = 3



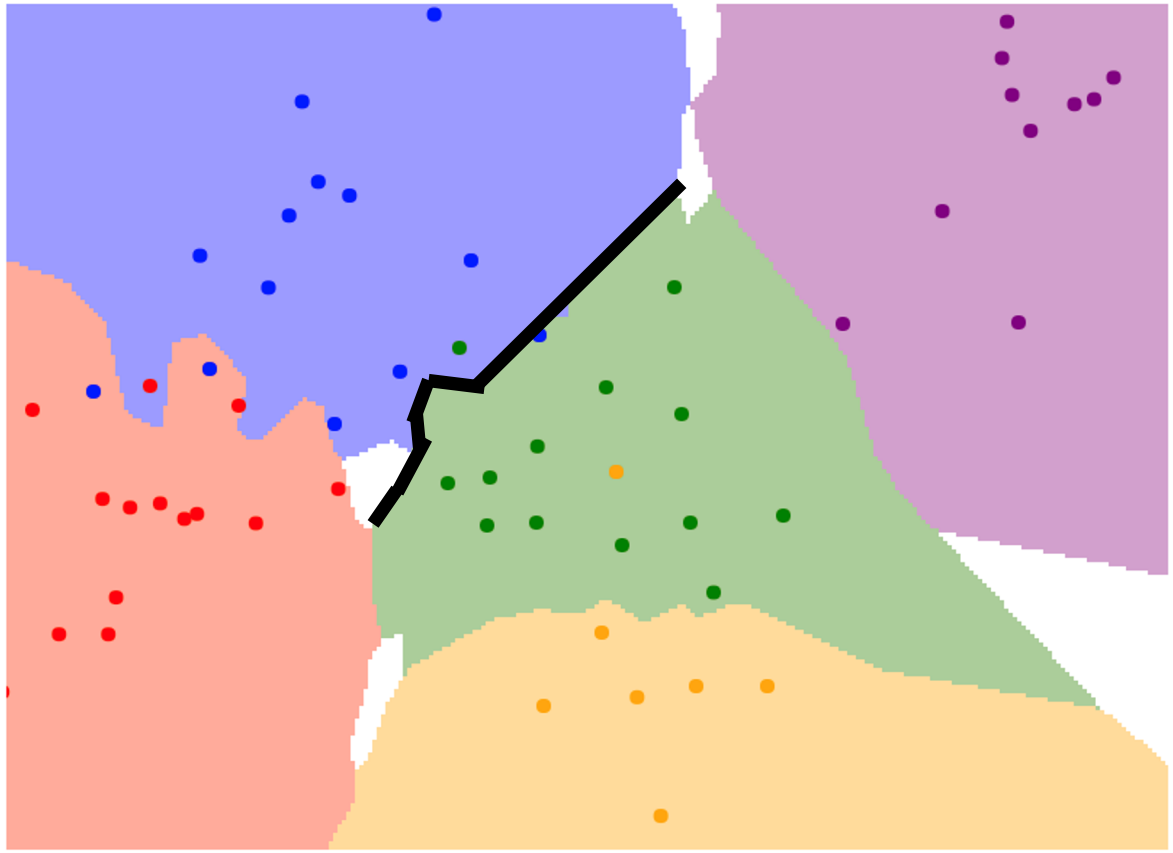
K-Nearest Neighbors

Using more neighbors helps smooth out rough decision boundaries

K = 1



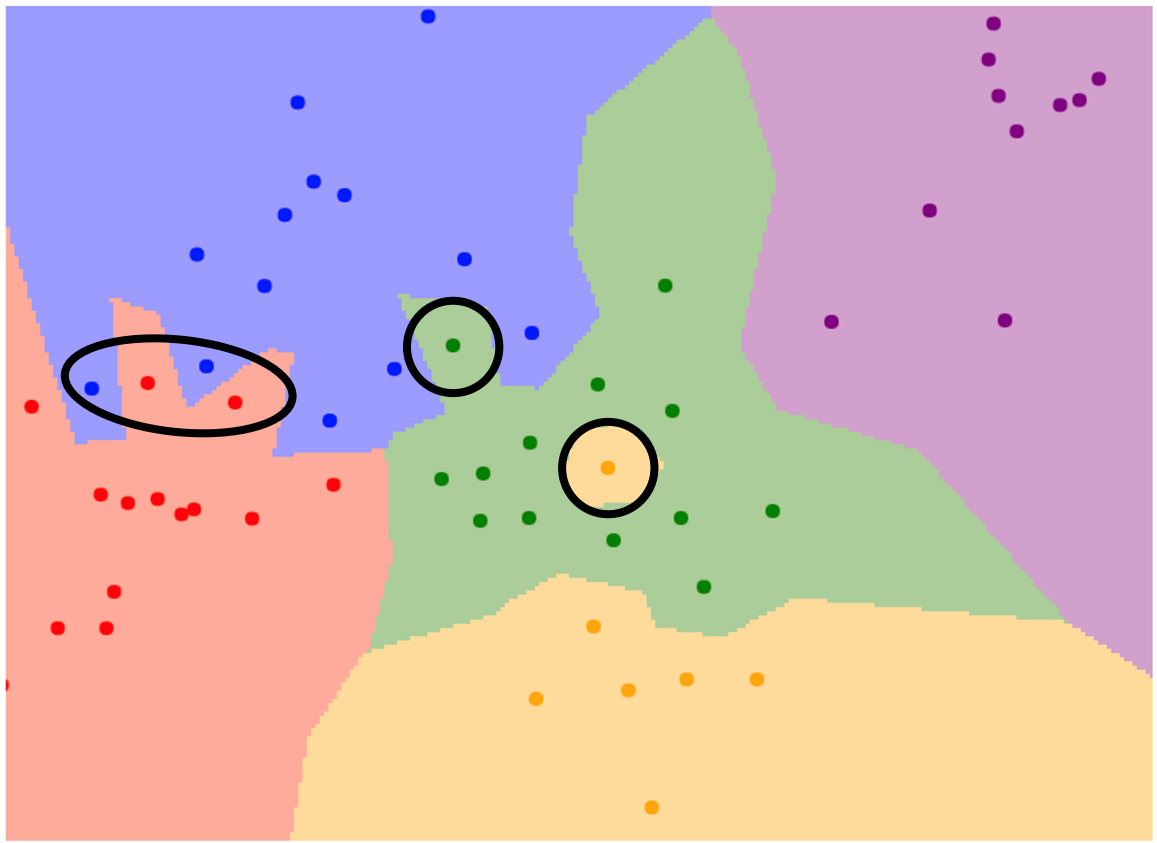
K = 3



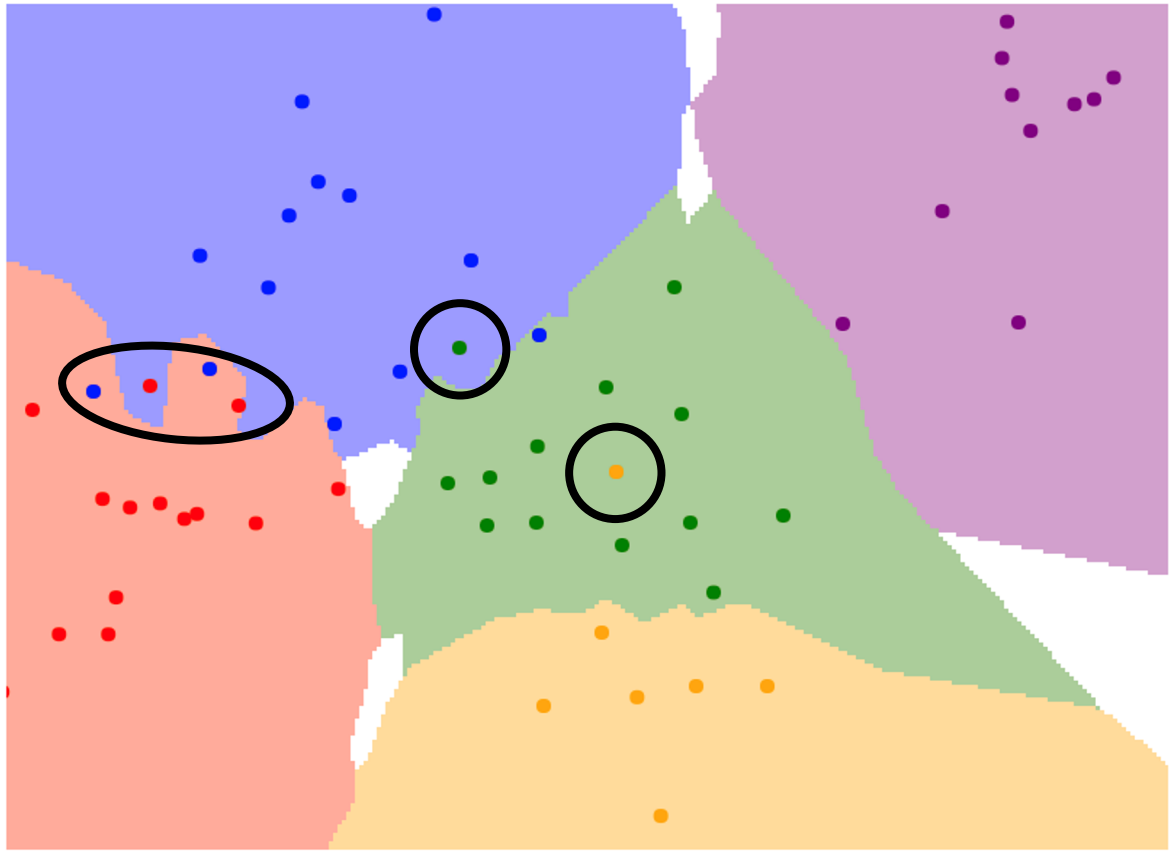
K-Nearest Neighbors

Using more neighbors helps reduce the effect of outliers

K = 1



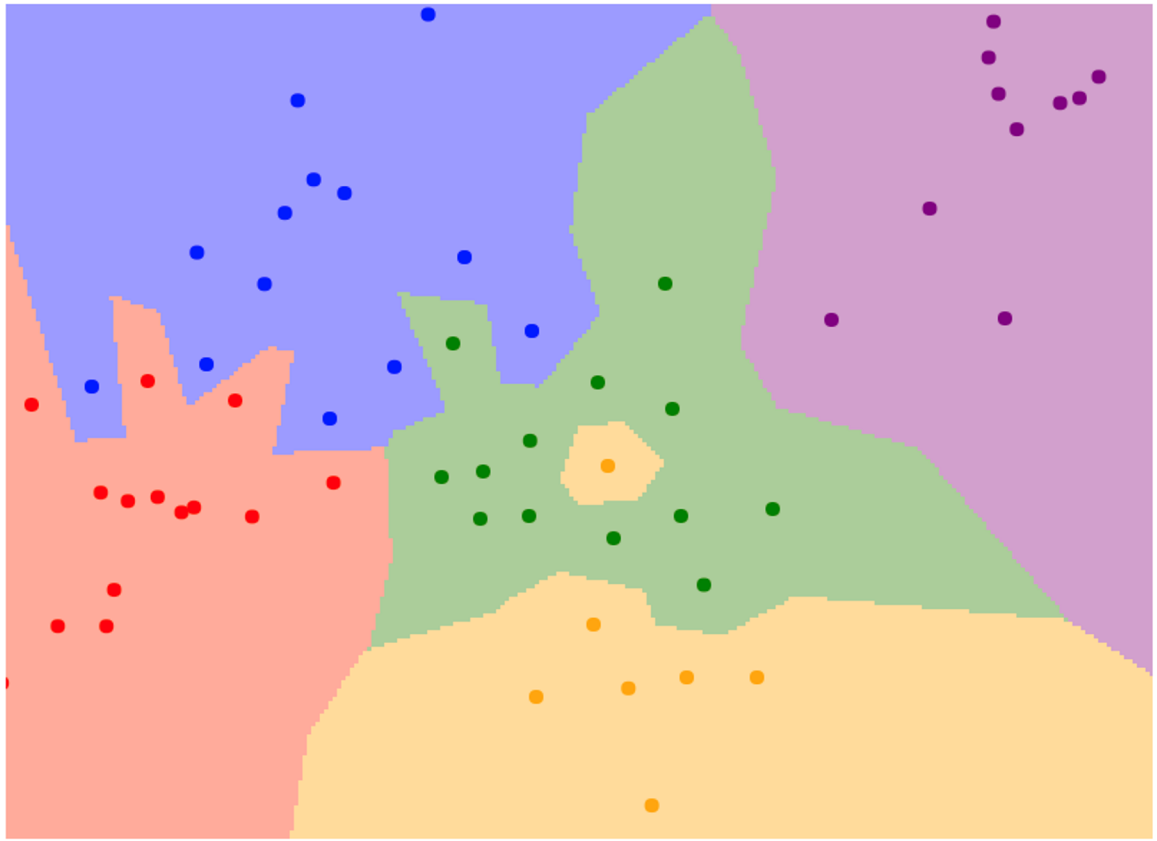
K = 3



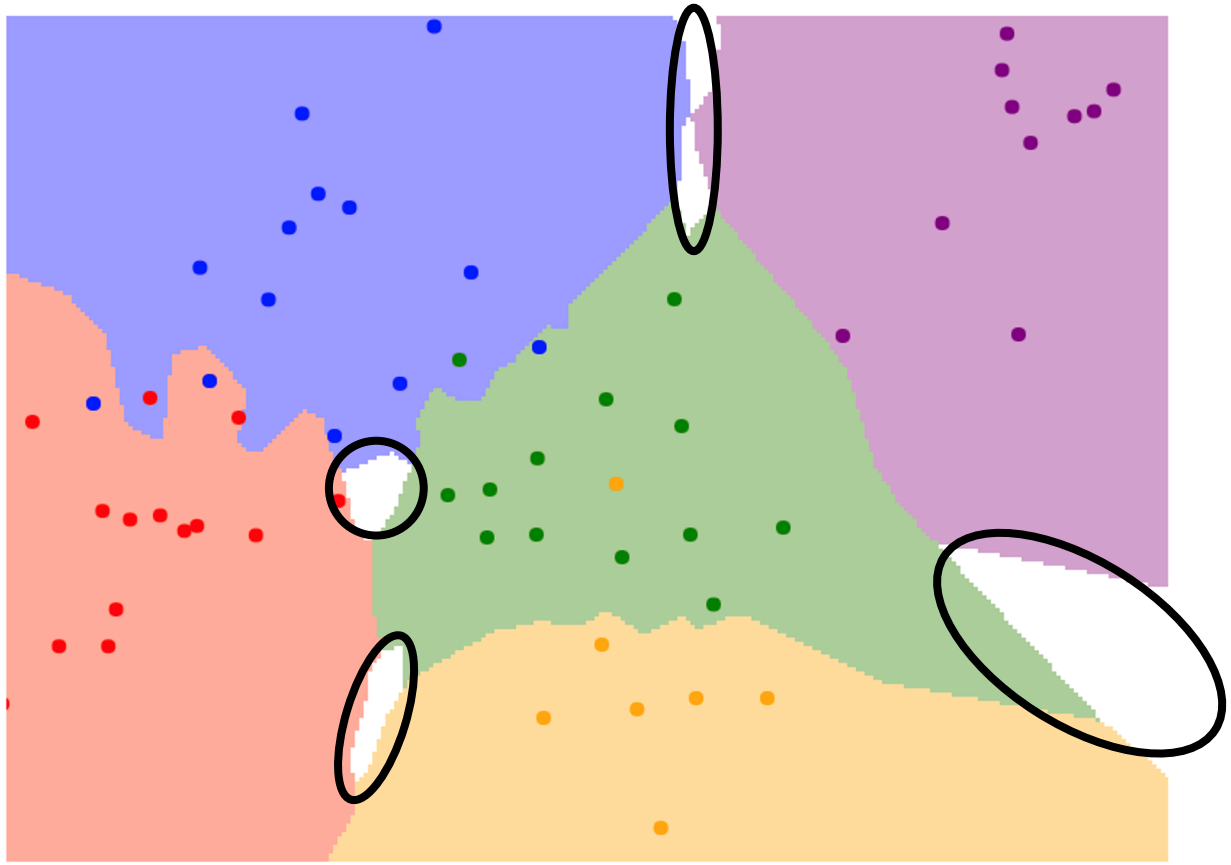
K-Nearest Neighbors

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

K = 1



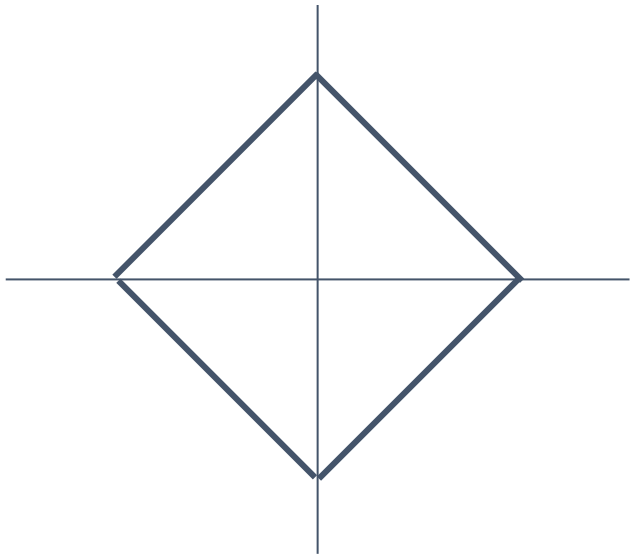
K = 3



K-Nearest Neighbors: Distance Metric

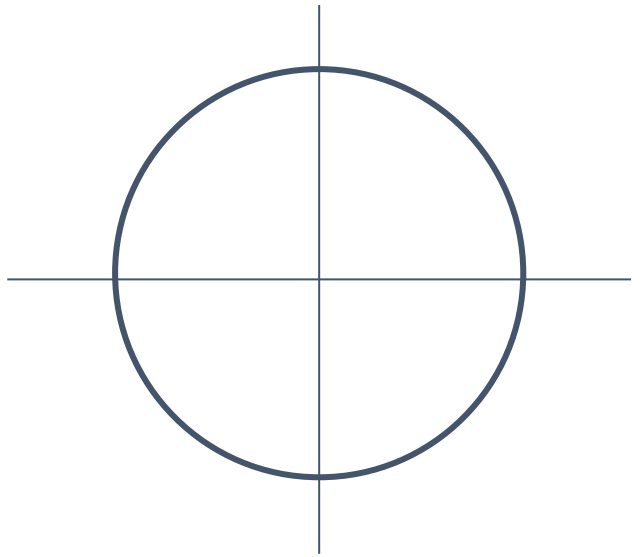
L1 (Manhattan) distance

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



L2 (Euclidean) distance

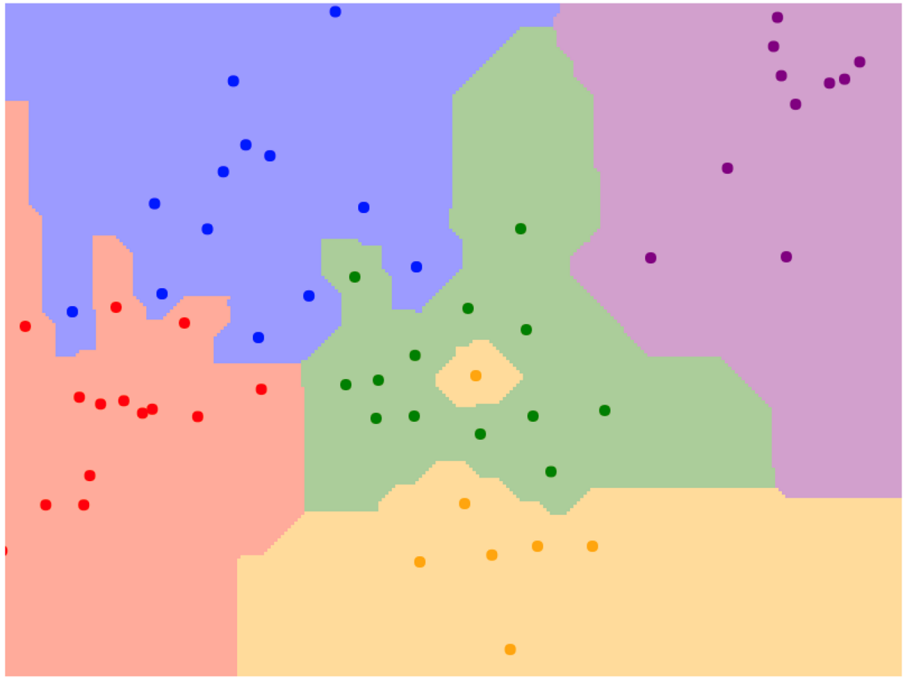
$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

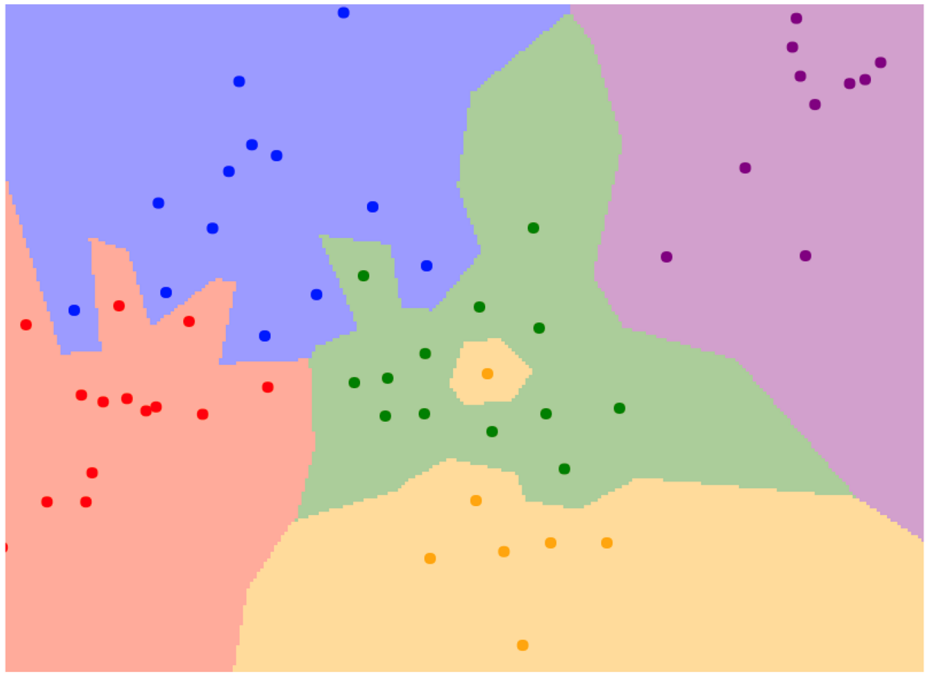
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

L2 (Euclidean) distance

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



K = 1

K-Nearest Neighbors: Distance Metric

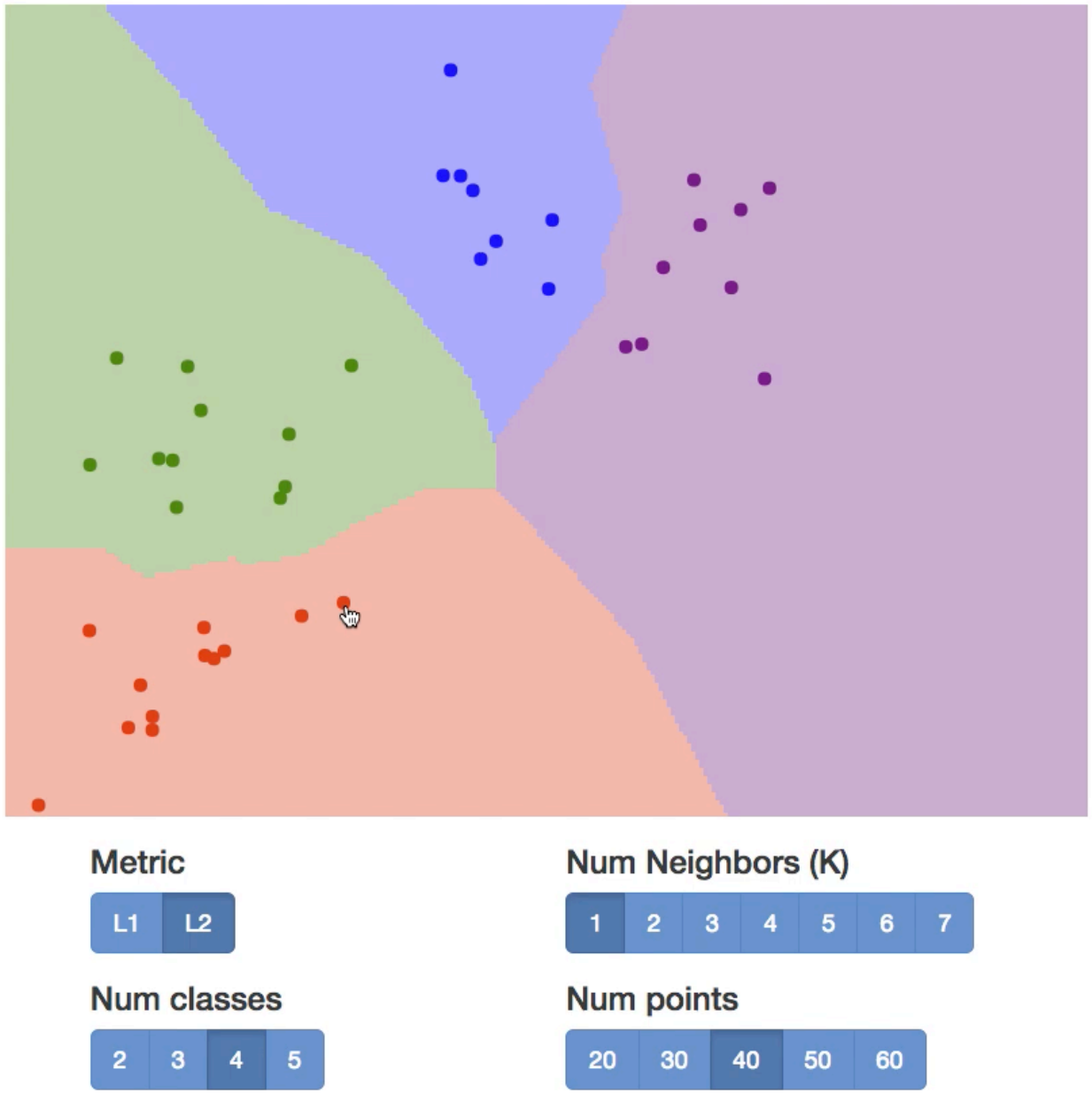
With the right choice of distance metric, we can apply K-Nearest Neighbor to any type of data!

K-Nearest Neighbors: Web Demo

Interactively move points around and see decision boundaries change

Play with L1 vs L2 metrics

Play with changing number of training points, value of K



http://vision.stanford.edu/teaching/cs231n-demos/knn/

Hyperparameters

What is the best value of **K** to use?
What is the best **distance metric** to use?

These are examples of **hyperparameters**: choices about our learning algorithm that we don't learn from the training data; instead we set them at the start of the learning process

Hyperparameters

What is the best value of **K** to use?
What is the best **distance metric** to use?

These are examples of **hyperparameters**: choices about our learning algorithm that we don't learn from the training data; instead we set them at the start of the learning process

Very problem-dependent. In general need to try them all and see what works best for our data / task.

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data



Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data



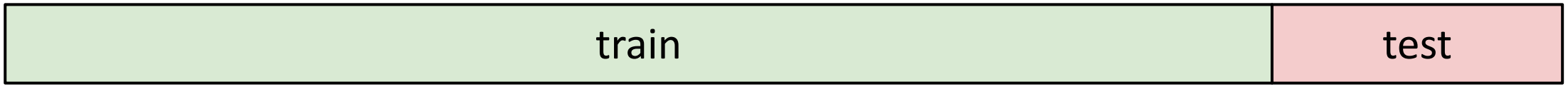
Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data



Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data



Setting Hyperparameters

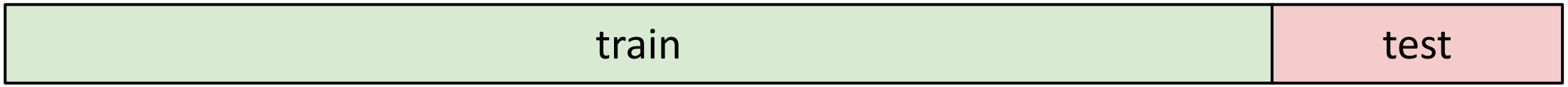
Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data



Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data



Setting Hyperparameters

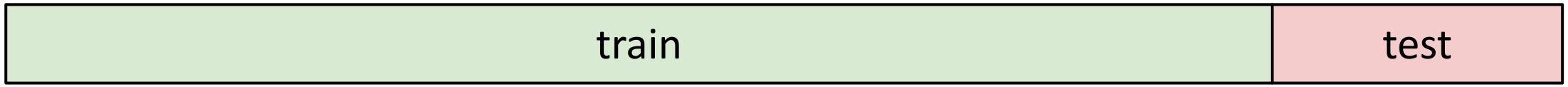
Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data



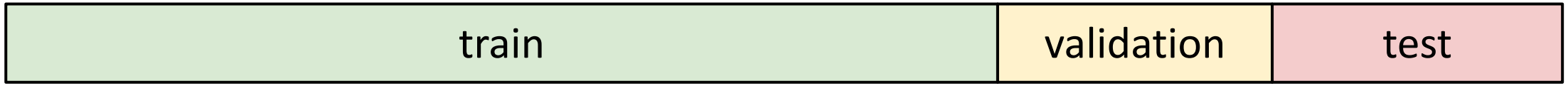
Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data



Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!



Setting Hyperparameters

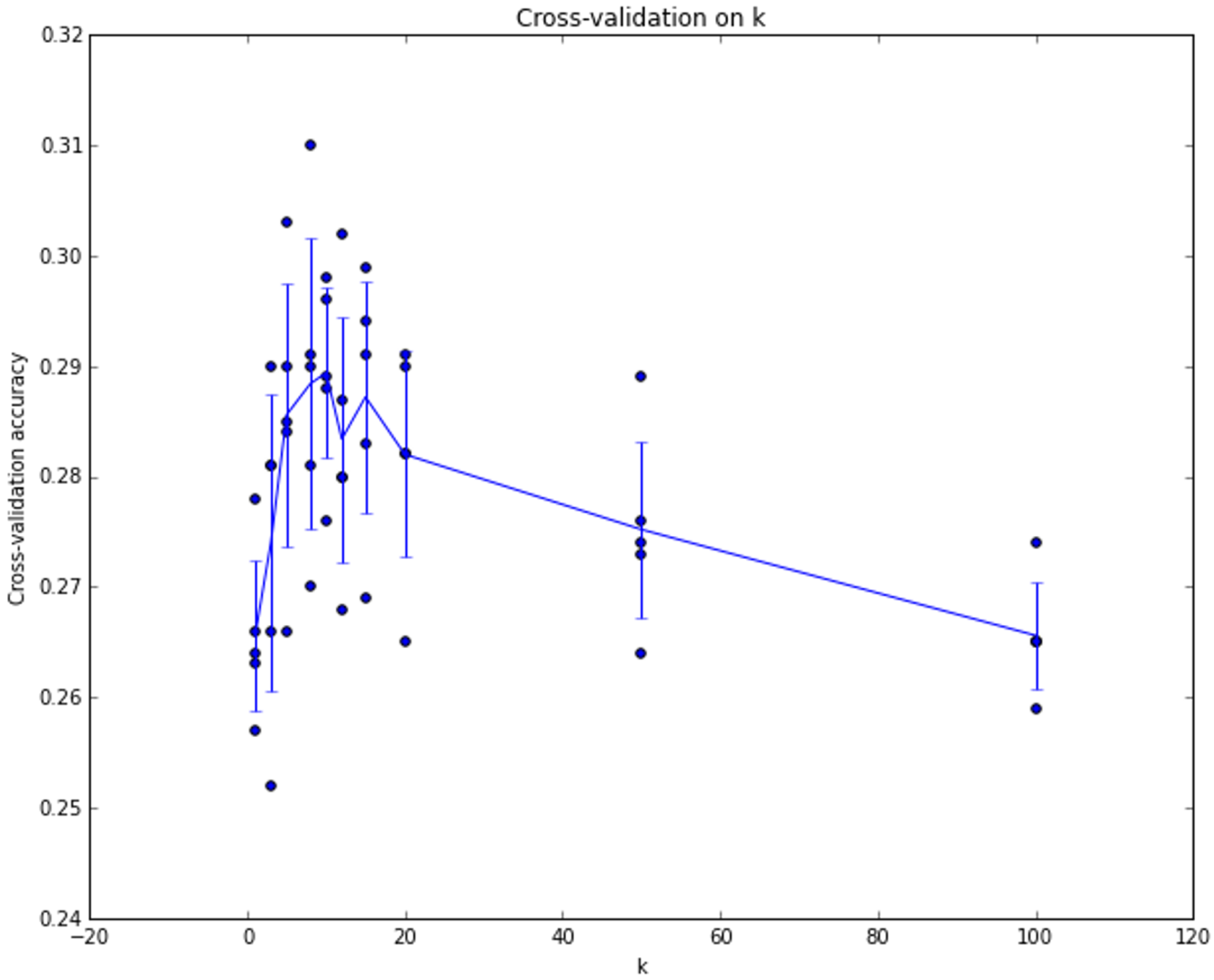
Your Dataset

Idea #4: **Cross-Validation**: Split data into **folds**, try each fold as validation and average the results

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

Useful for small datasets, but (unfortunately) not used too frequently in deep learning

Setting Hyperparameters



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 ~ 7 works best for this data)

K-Nearest Neighbor on raw pixels is **seldom 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)

Original image is CC0 public domain

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.



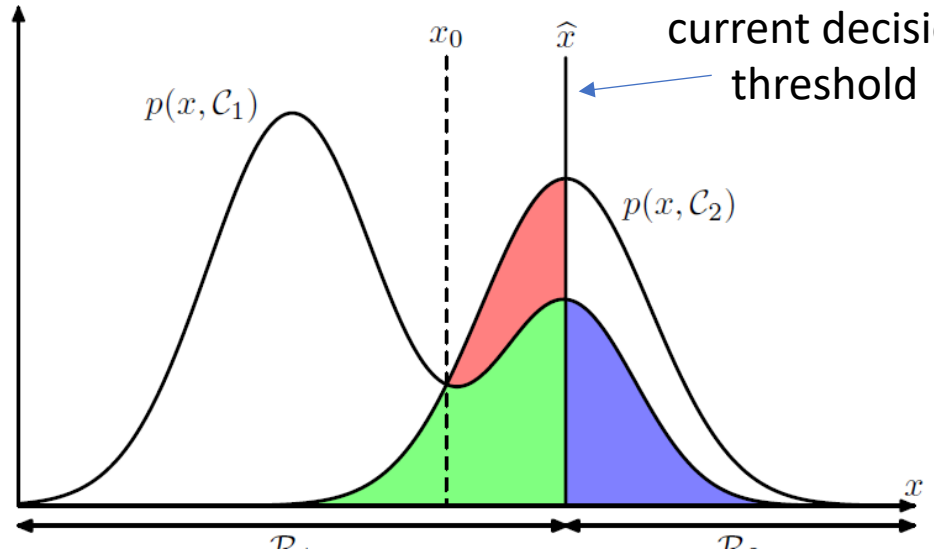
A wooden bench in front of a building.

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

Machine learning

- Why learning?
- Nearest neighbors
- Bayesian classification
- Logistic regression
- Support vector machines
- Clustering
- Principal component analysis

Bayesian classification

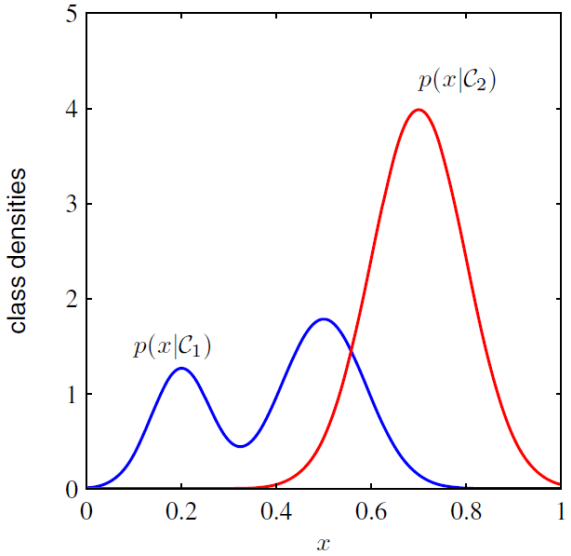


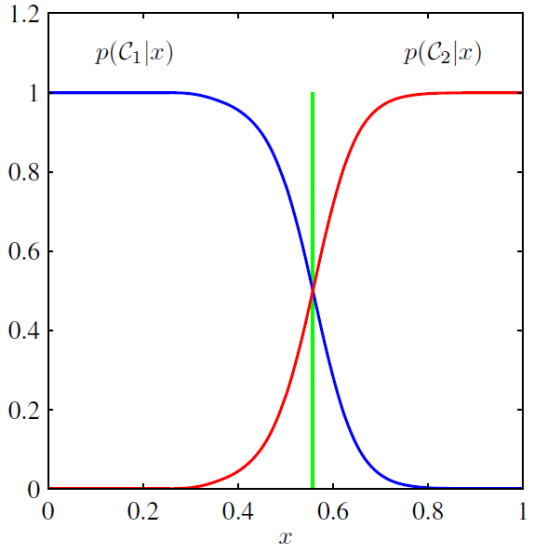
current decision threshold

What do the colors mean?

Bayes' Rule

$$y_k = p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)}$$





Bayesian classification

5.1.2 Bayesian classification

For some simple machine learning problems, e.g., if we have an analytic model of feature construction and noising, or if we can gather enough samples, we can determine the probablity distributions of the feature vectors for each class $p(\mathbf{x}|\mathcal{C}_k)$ as well as the prior class likelihoods $p(\mathcal{C}_k)$.[4] According to Bayes' Rule (4.33), the likelihood of class $\mathcal{C}_k$ given a feature vector $\mathbf{x}$ is given by

$$y_k = p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)} \tag{5.2}$$

$$= \frac{\exp l_k}{\sum_j \exp l_j}, \tag{5.3}$$

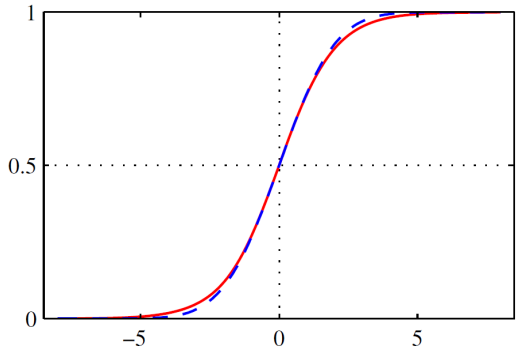
where the second form (using the exp functions) is known as the *normalized exponential* or *softmax* function. The quantity

$$l_k = \log p(\mathbf{x}|\mathcal{C}_k) + \log p(\mathcal{C}_k) \tag{5.4}$$

is the *log-likelihood* of sample $\mathbf{x}$ being from class $\mathcal{C}_k$.[5] It is sometimes convenient to denote the softmax function itself as a vector-to-vector valued function,

$$\mathbf{y} = \text{softmax}(\mathbf{l}). \tag{5.5}$$

Logistic sigmoid function



For the binary (two class) classification task, we can re-write (5.3) as

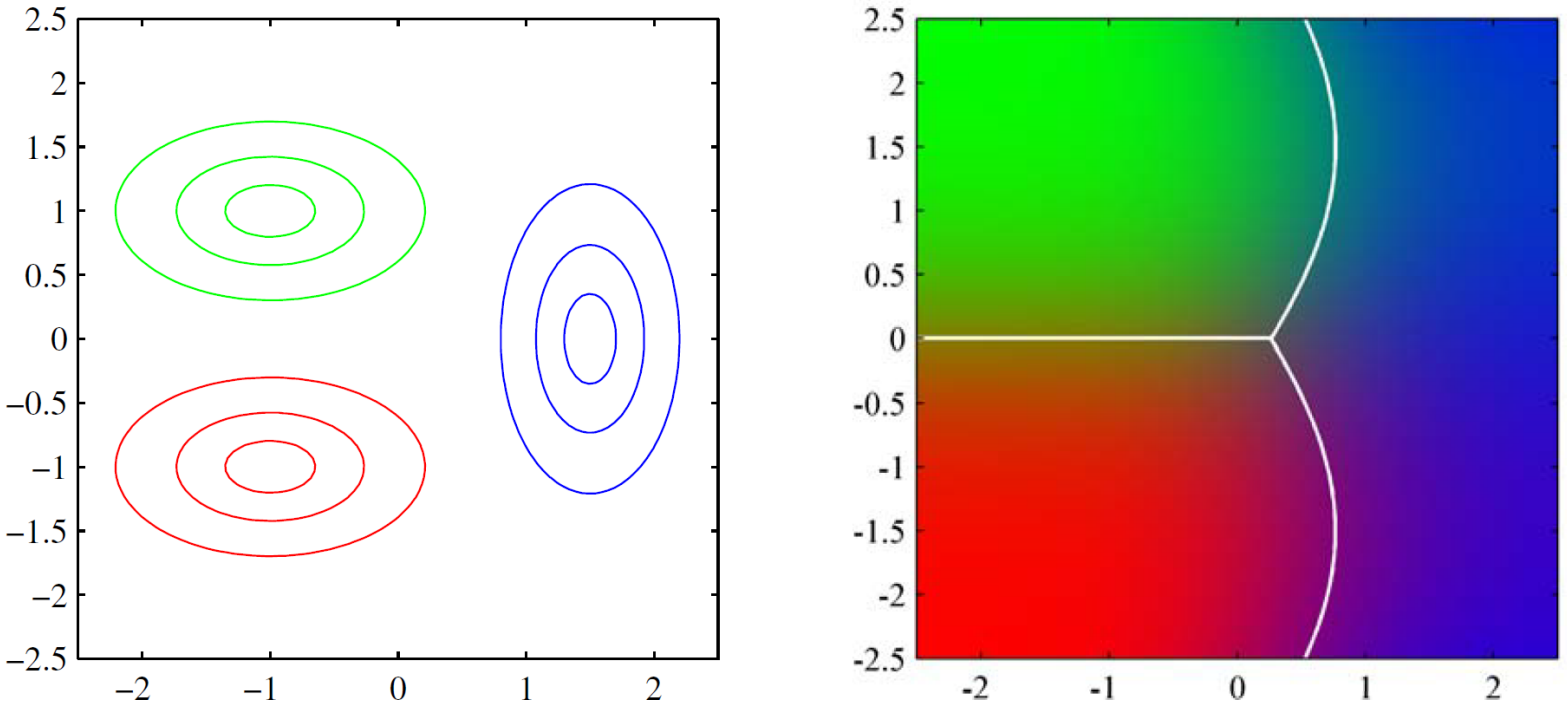
$$p(\mathcal{C}_0|\mathbf{x}) = \frac{1}{1+\exp(-l)} = \sigma(l), \tag{5.7}$$

where $l = l_0 - l_1$ is the difference between the two class log likelihood and is known as the *log odds* or *logit*.

The $\sigma(l)$ function is called the *logistic sigmoid function* (or simply the *logistic function* or *logistic curve*), where *sigmoid* means an S-shaped curve (Figure 5.6). The sigmoid was a popular *activation function* in earlier neural networks, although it has now been replaced with other functions, as discussed in Section 5.3.2.

Multivariate Gaussian distributions

- How do we determine the class posterior?



Multivariate Gaussian distribution

For Gaussians with identical covariance matrices $\mathbf{\Sigma}$, we have

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{\|\mathbf{\Sigma}\|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k).\right\} \tag{5.8}$$

In the case of two classes (binary classification), we obtain (Bishop 2006, Section 4.2.1)

$$p(\mathcal{C}_0|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b), \tag{5.9}$$

with

$$\mathbf{w} = \mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1), \quad \text{and} \tag{5.10}$$

$$b = \frac{1}{2}\boldsymbol{\mu}_0^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_0 + \frac{1}{2}\boldsymbol{\mu}_1^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 + \log\frac{p(\mathcal{C}_0)}{p(\mathcal{C}_1)}. \tag{5.11}$$

Logistic regression

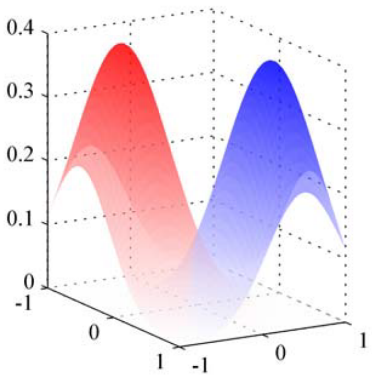
$$p(\mathcal{C}_0|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b), \tag{5.9}$$

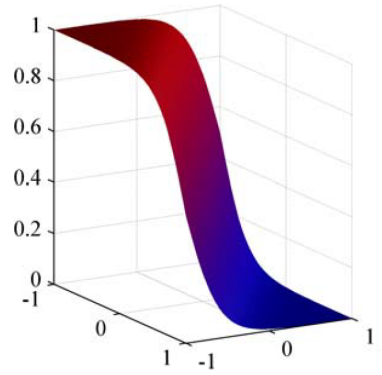
Equation (5.9), which we will revist shortly in the context of non-generative (discriminative) classification (5.18), is called *logistic regression*, since we pass the output of a linear regression formula

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{5.12}$$

through the logistic function to obtain a class probability. Figure 5.7 illustrates this in two dimensions, there the posterior likelihood of the red class $p(\mathcal{C}_0|\mathbf{x})$ is shown on the right side.

In linear regression (5.12), $\mathbf{w}$ plays the role a the *weight* vector along which we project the feature vector $\mathbf{x}$, and b plays the role of the *bias*, which determines where to set the





Multiple classes & linear discriminant analysis

For $K > 2$ classes, the softmax function (5.3) can be applied to the linear regression log likelihoods,

$$l_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k,$$

$$y_k = p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)}$$

with

$$\mathbf{w}_k = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k, \quad \text{and} \tag{5.14}$$

$$b_k = -\frac{1}{2}\boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \log p(\mathcal{C}_k). \tag{5.15}$$

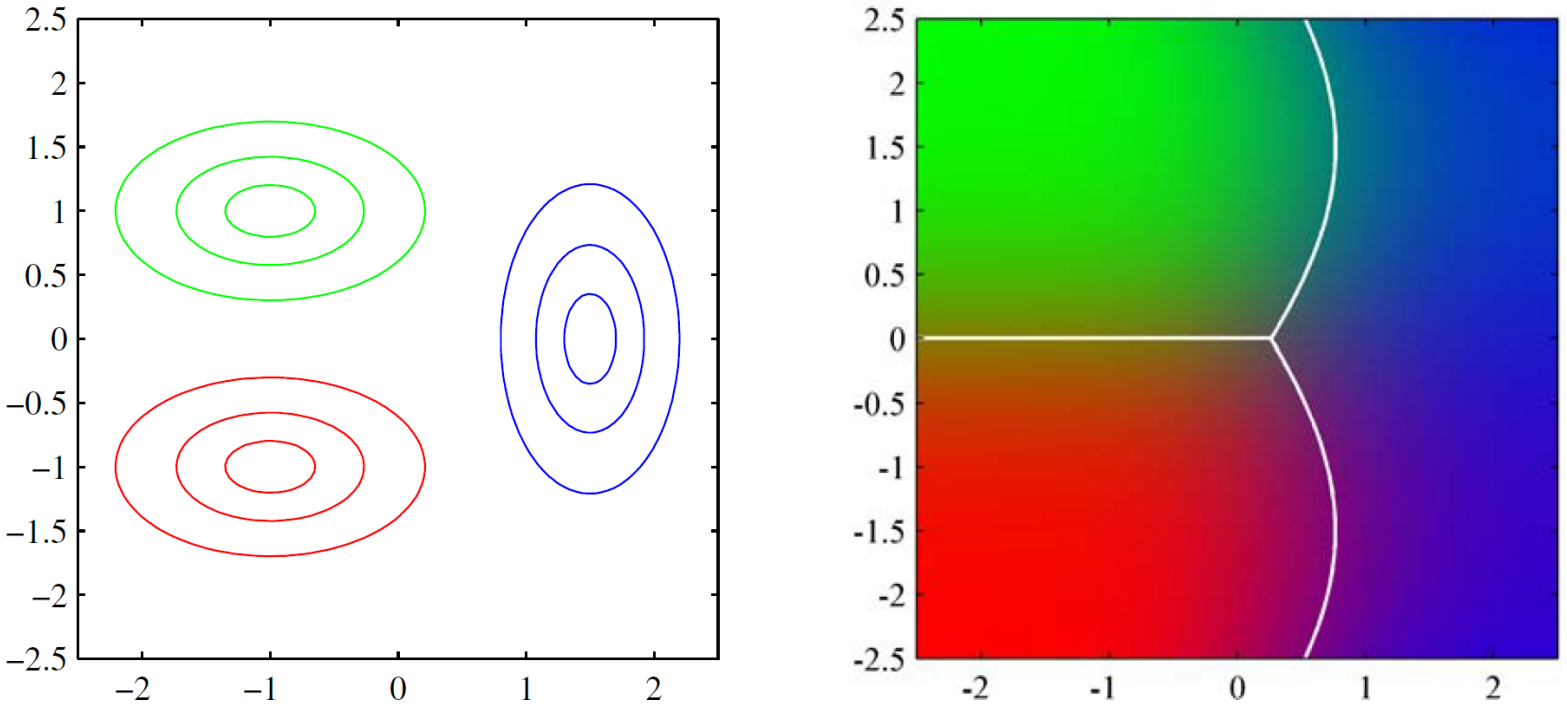
Because the decision boundaries along which the classification switches from one class to another are linear,

$$\mathbf{w}_k \mathbf{x} + b_k > \mathbf{w}_l \mathbf{x} + b_l, \tag{5.16}$$

the technique of classifying examples using such crieteria is known as *linear discriminant analysis* (Bishop 2006, Section 4.1; Murphy 2012, Section 4.2.2).[6]

Quadratic discriminant analysis

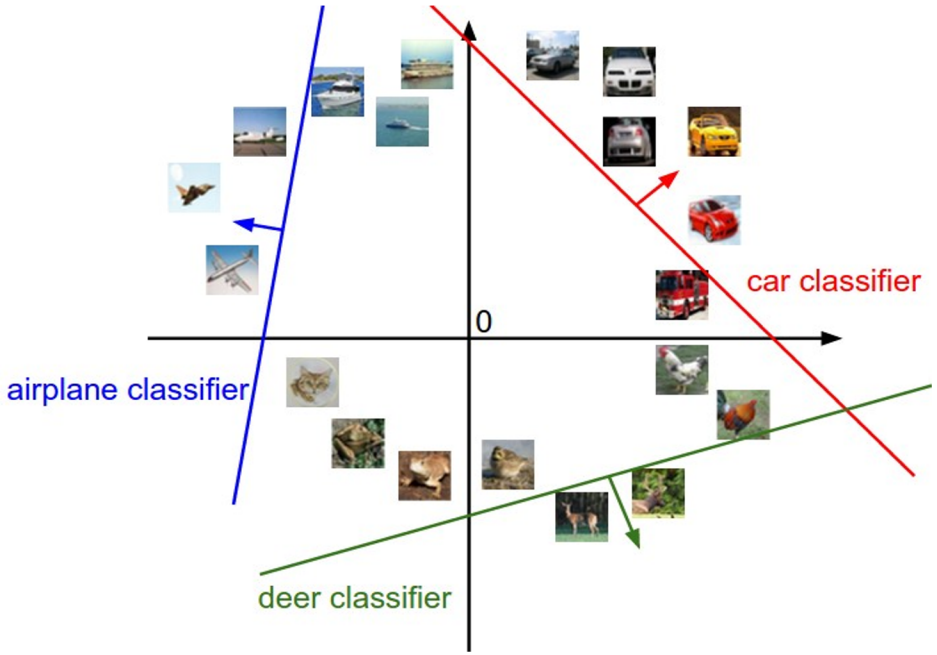
- What are the white lines on the right called?



Machine learning

- Why learning?
- Nearest neighbors
- Bayesian classification
- Logistic regression (linear classifiers)
- Support vector machines
- Clustering
- Principal component analysis

Lecture 3: Linear Classifiers



Neural Network

Linear classifiers



This image is CC0 1.0 public domain

Recall CIFAR10



50,000 training images
each image is **32x32x3**

10,000 test images.

Parametric Approach

Image



Array of **32x32x3** numbers
(3072 numbers total)

f(**x**,**W**)

10 numbers giving class scores

W
parameters or weights

Parametric Approach: Linear Classifier

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

Image



Array of **32x32x3** numbers
(3072 numbers total)

f(**x**,**W**)

10 numbers giving class scores

W
parameters or weights

Parametric Approach: Linear Classifier

(3072,)

$f(x,W) = Wx$

(10,) (10, 3072)

Image



Array of **32x32x3** numbers
(3072 numbers total)

$f(x,W)$

10 numbers giving class scores

W
parameters or weights

Parametric Approach: Linear Classifier

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

(3072,)

(10,)

(10, 3072)

(10,)

Image



Array of **32x32x3** numbers
(3072 numbers total)

f(**x**, **W**)

10 numbers giving class scores

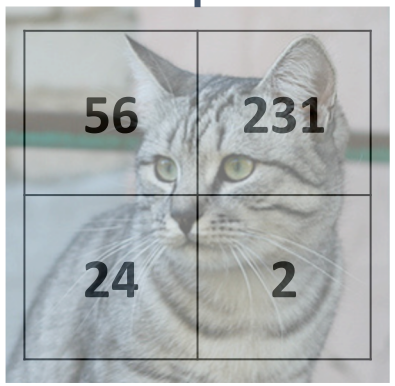
W

parameters or weights

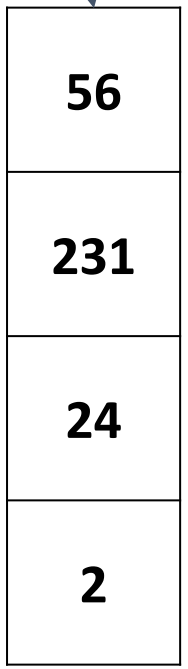
Example for 2x2 image, 3 classes (cat/dog/ship)

Stretch pixels into column

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



Input image
(2, 2)

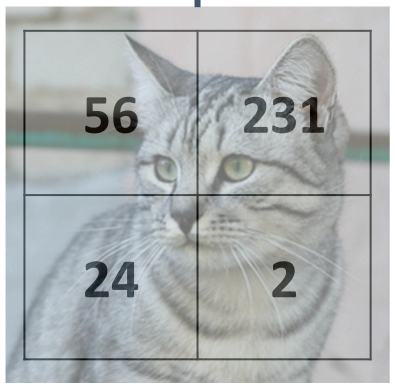


(4,)

Example for 2x2 image, 3 classes (cat/dog/ship)

Stretch pixels into column

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



Input image
(2, 2)

0.2	-0.5	0.1	2.0
1.5	1.3	2.1	0.0
0	0.25	0.2	-0.3

W (3, 4)

56
231
24
2

(4,)

+

1.1
3.2
-1.2

b
(3,)

=

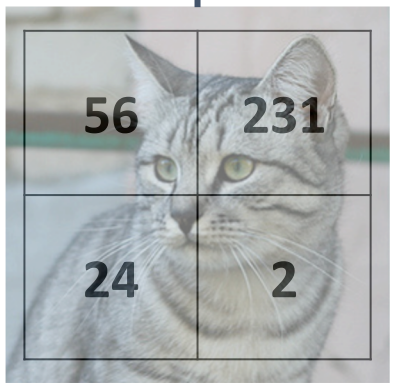
-96.8
437.9
61.95

(3,)

Linear Classifier: Algebraic Viewpoint

Stretch pixels into column

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



Input image
(2, 2)

0.2	-0.5	0.1	2.0
1.5	1.3	2.1	0.0
0	0.25	0.2	-0.3

W (3, 4)

56
231
24
2

(4,)

+

1.1
3.2
-1.2

b
(3,)

=

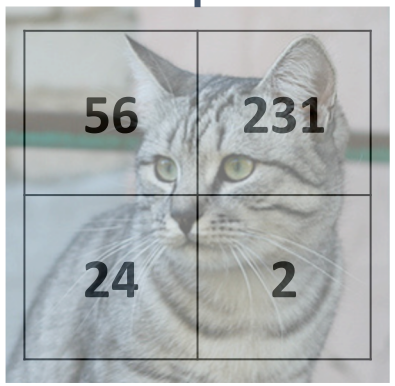
-96.8
437.9
61.95

(3,)

Linear Classifier: Bias Trick

Add extra one to data vector; bias is absorbed into last column of weight matrix

Stretch pixels into column



Input image
(2, 2)

0.2	-0.5	0.1	2.0	1.1
1.5	1.3	2.1	0.0	3.2
0	0.25	0.2	-0.3	-1.2

W (3, 5)

56
231
24
2
1

(5,)

=

-96.8
437.9
61.95

(3,)

Linear Classifier: Predictions are Linear!

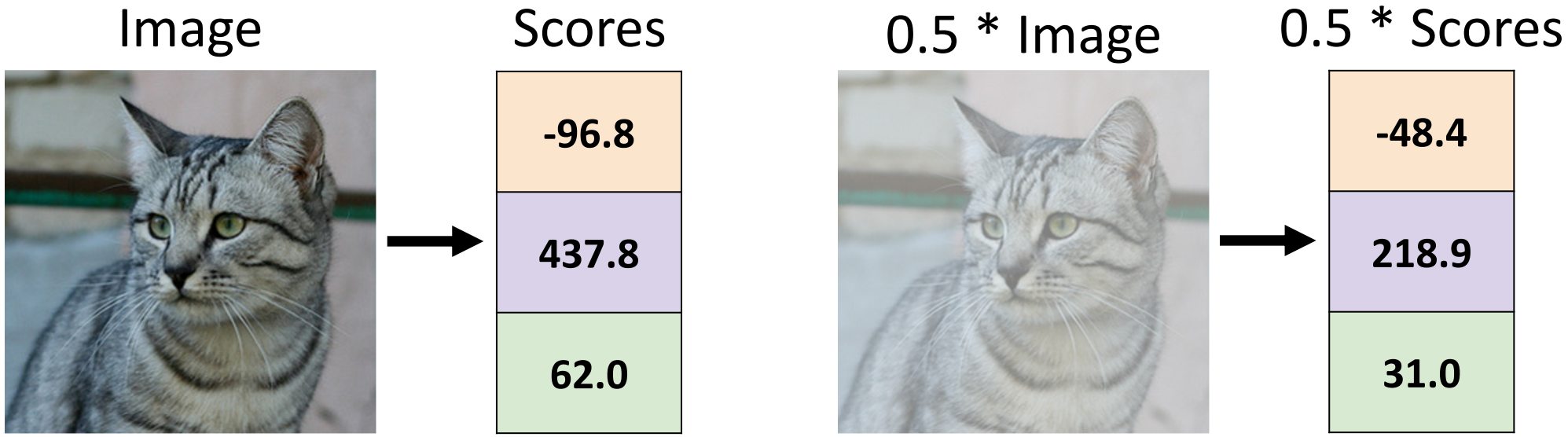
$$f(x, W) = Wx \quad \text{(ignore bias)}$$

$$f(cx, W) = W(cx) = c * f(x, W)$$

Linear Classifier: Predictions are Linear!

$$f(x, W) = Wx \quad \text{(ignore bias)}$$

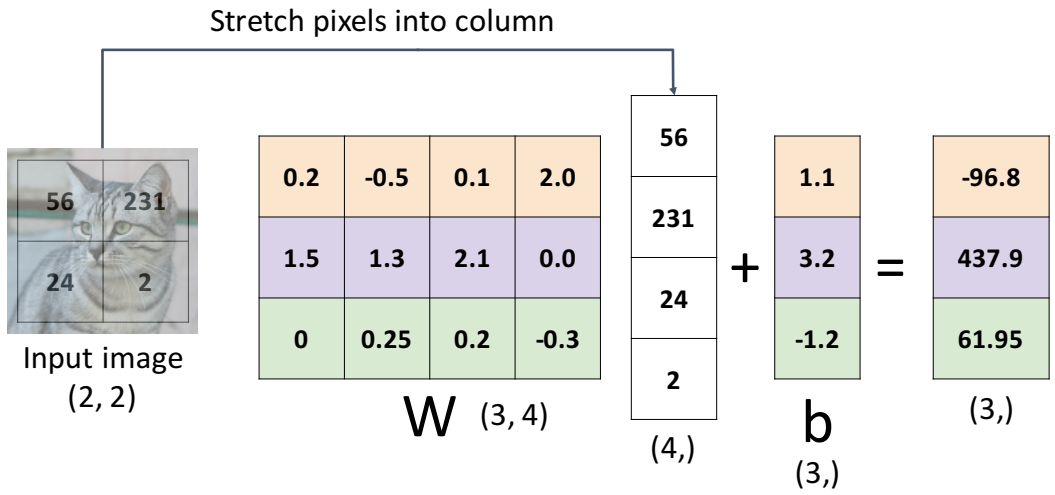
$$f(cx, W) = W(cx) = c * f(x, W)$$



Interpreting a Linear Classifier

Algebraic Viewpoint

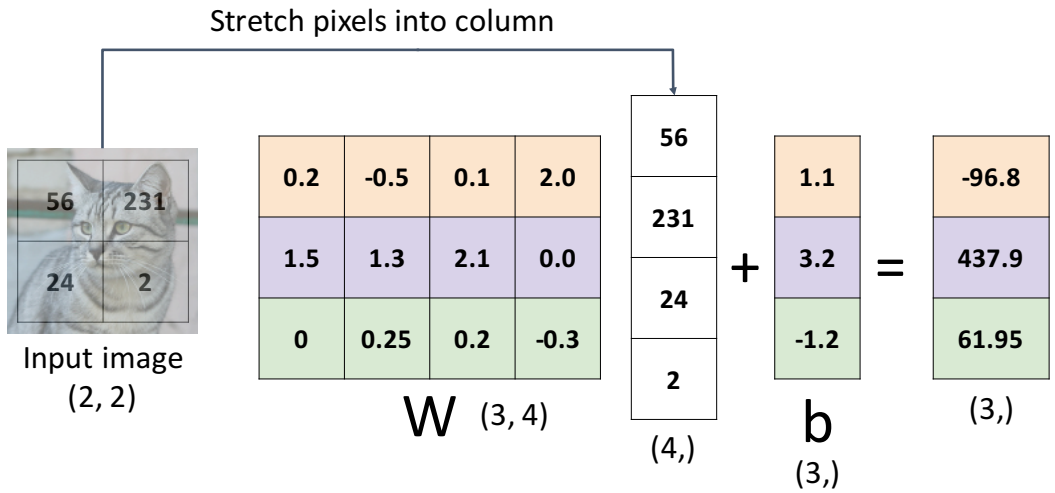
$$f(x,W) = Wx + b$$

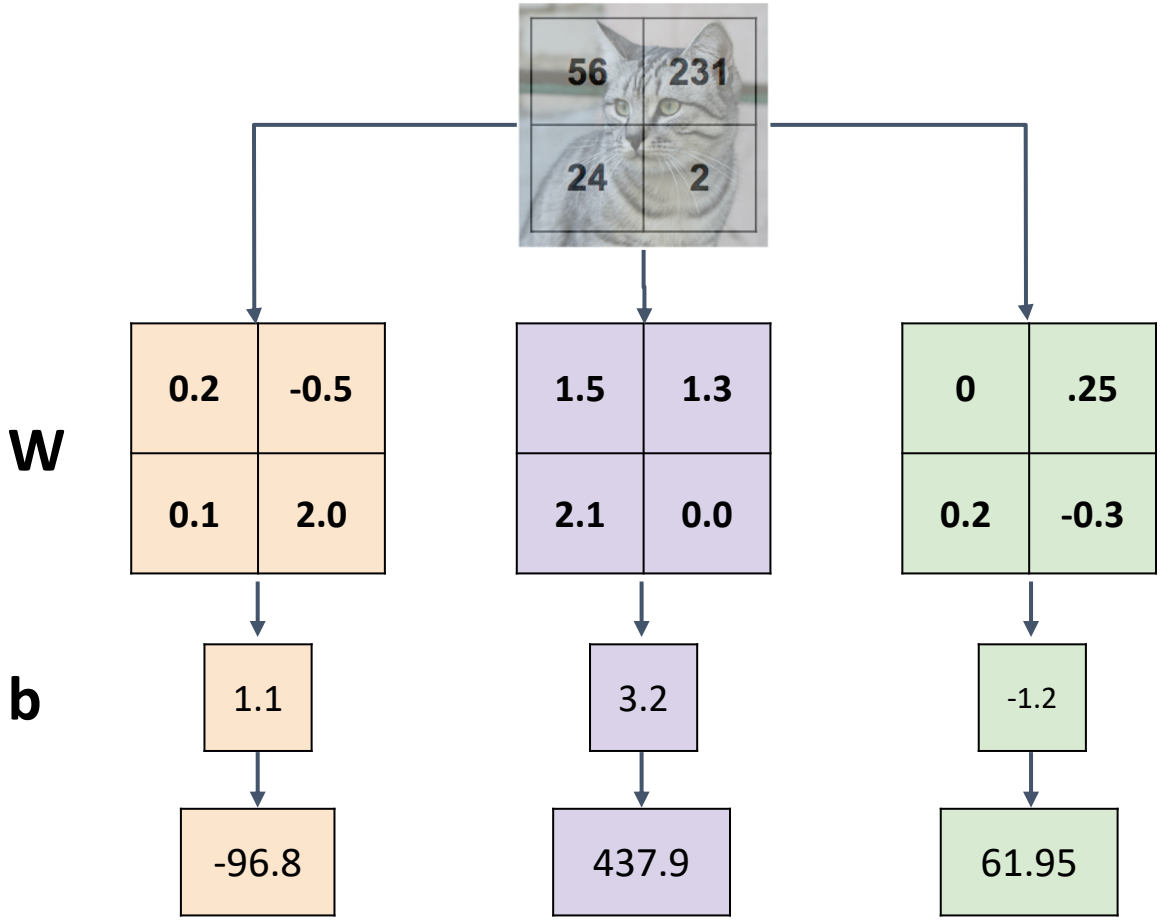


Interpreting a Linear Classifier

Algebraic Viewpoint

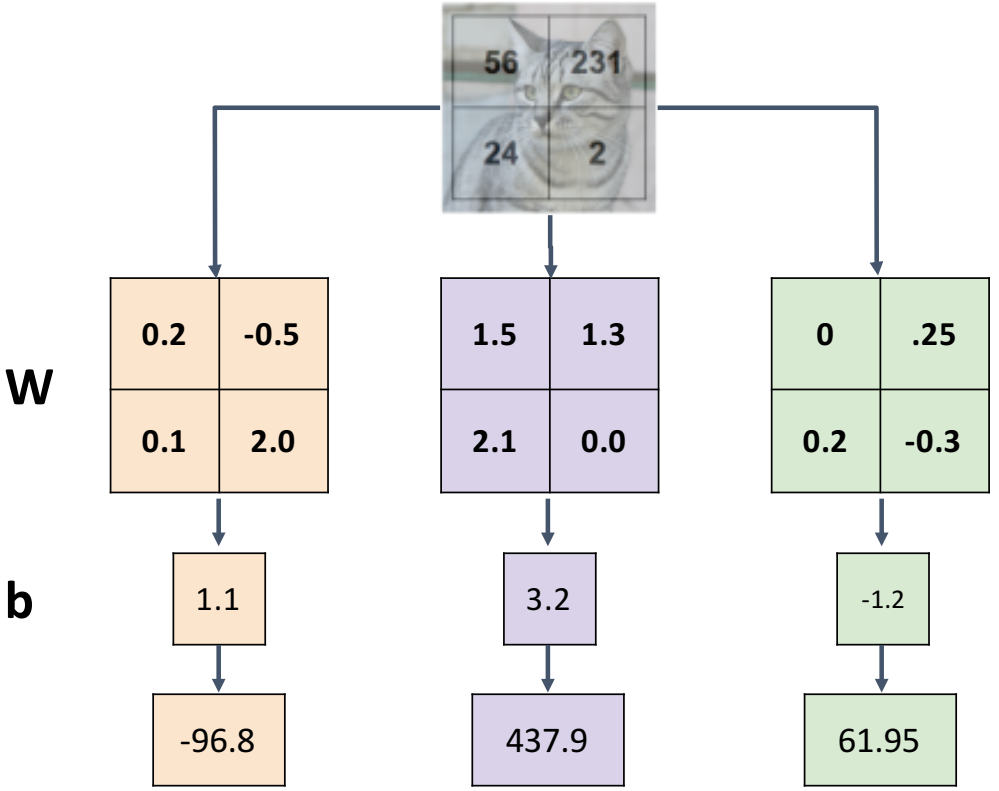
$$f(x,W) = Wx + b$$





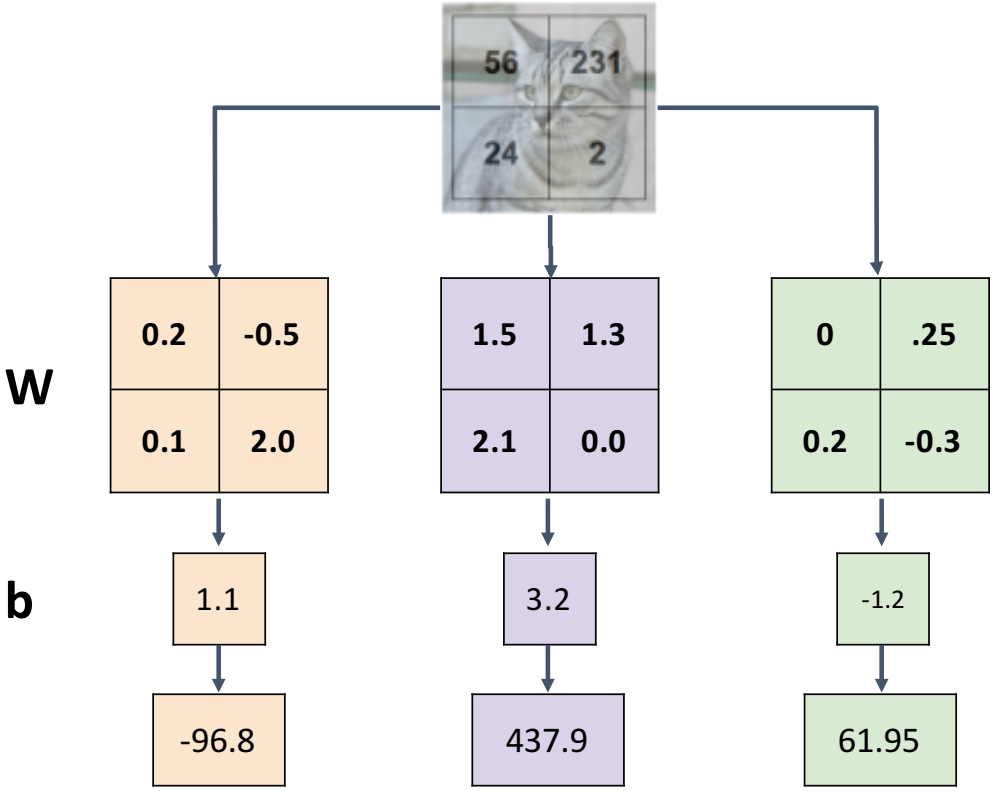
Interpreting an Linear Classifier

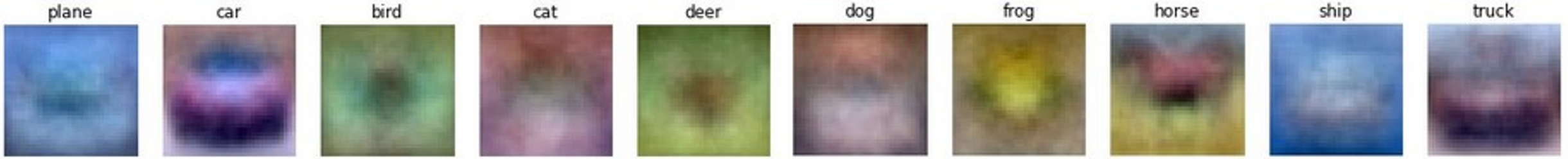




Interpreting an Linear Classifier: Visual Viewpoint

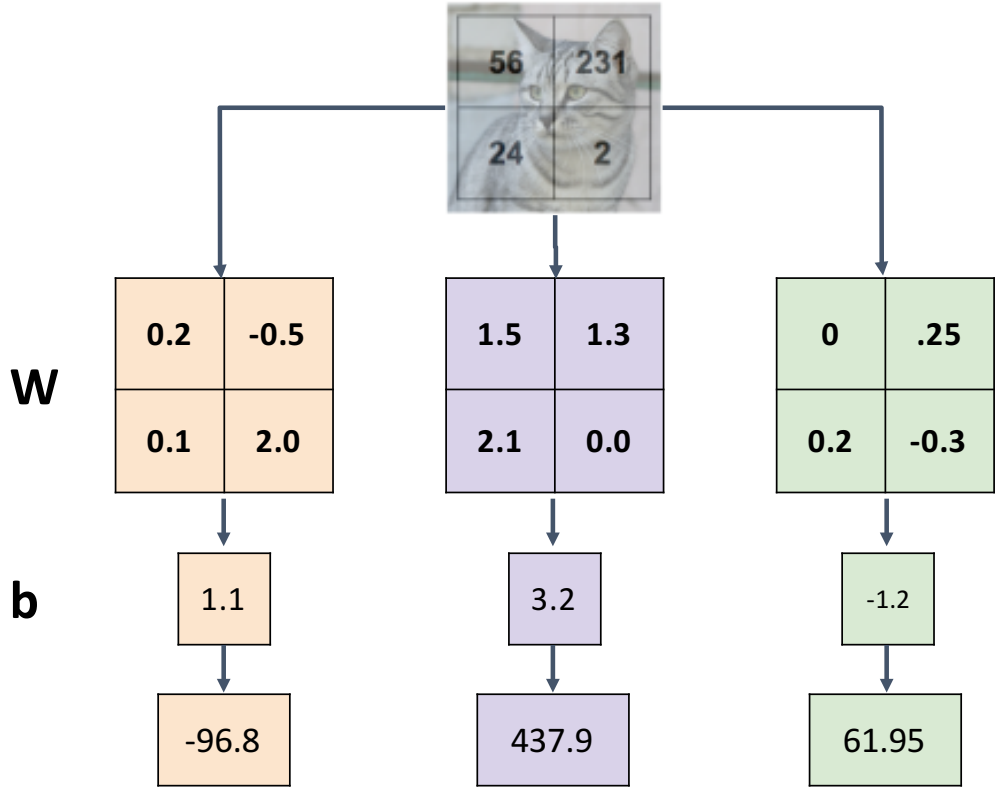


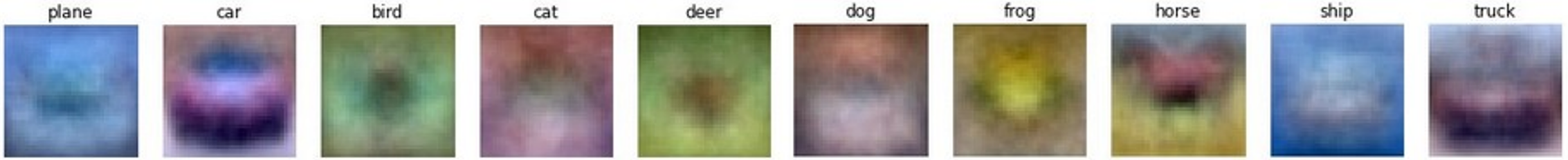




Interpreting an Linear Classifier: Visual Viewpoint

Linear classifier has one "template" per category



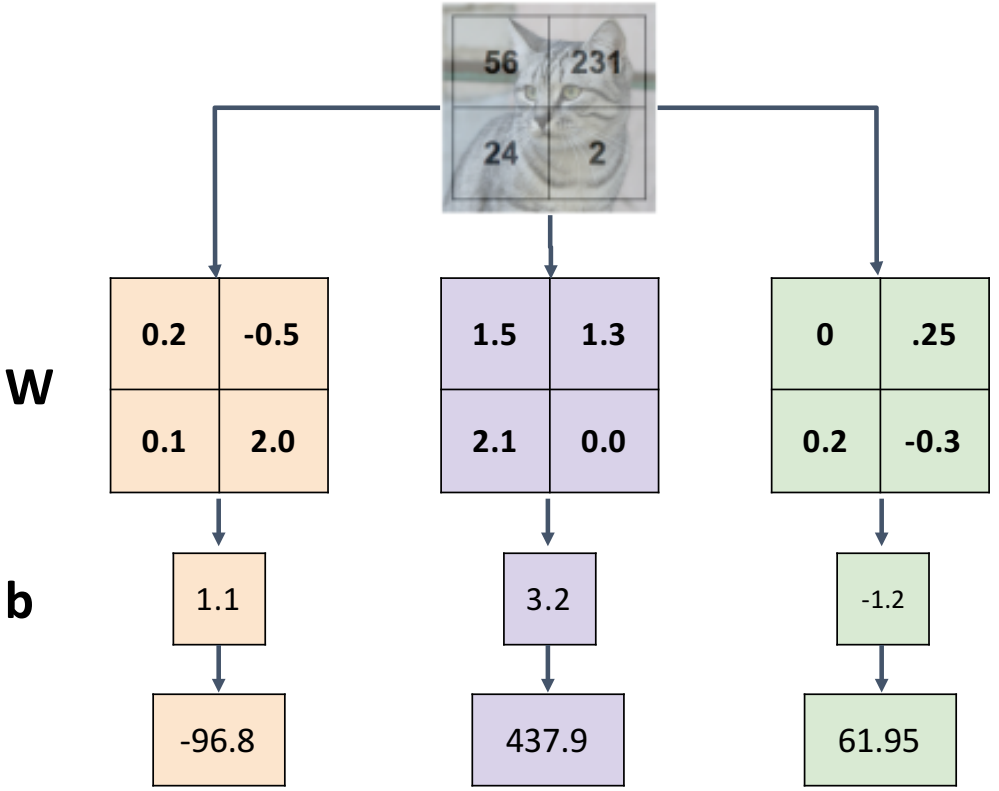


Interpreting an Linear Classifier: Visual Viewpoint

Linear classifier has one "template" per category

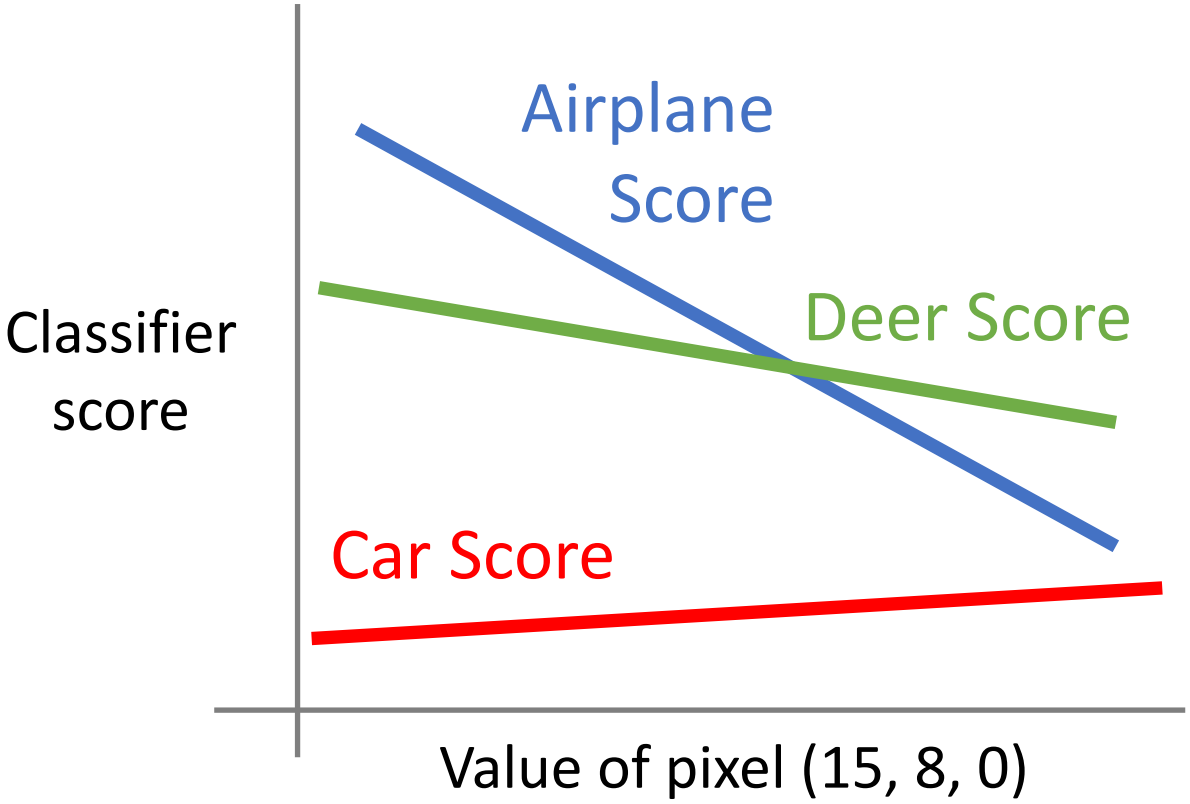
A single template cannot capture multiple modes of the data

e.g. horse template has 2 heads!





Interpreting a Linear Classifier: Geometric Viewpoint

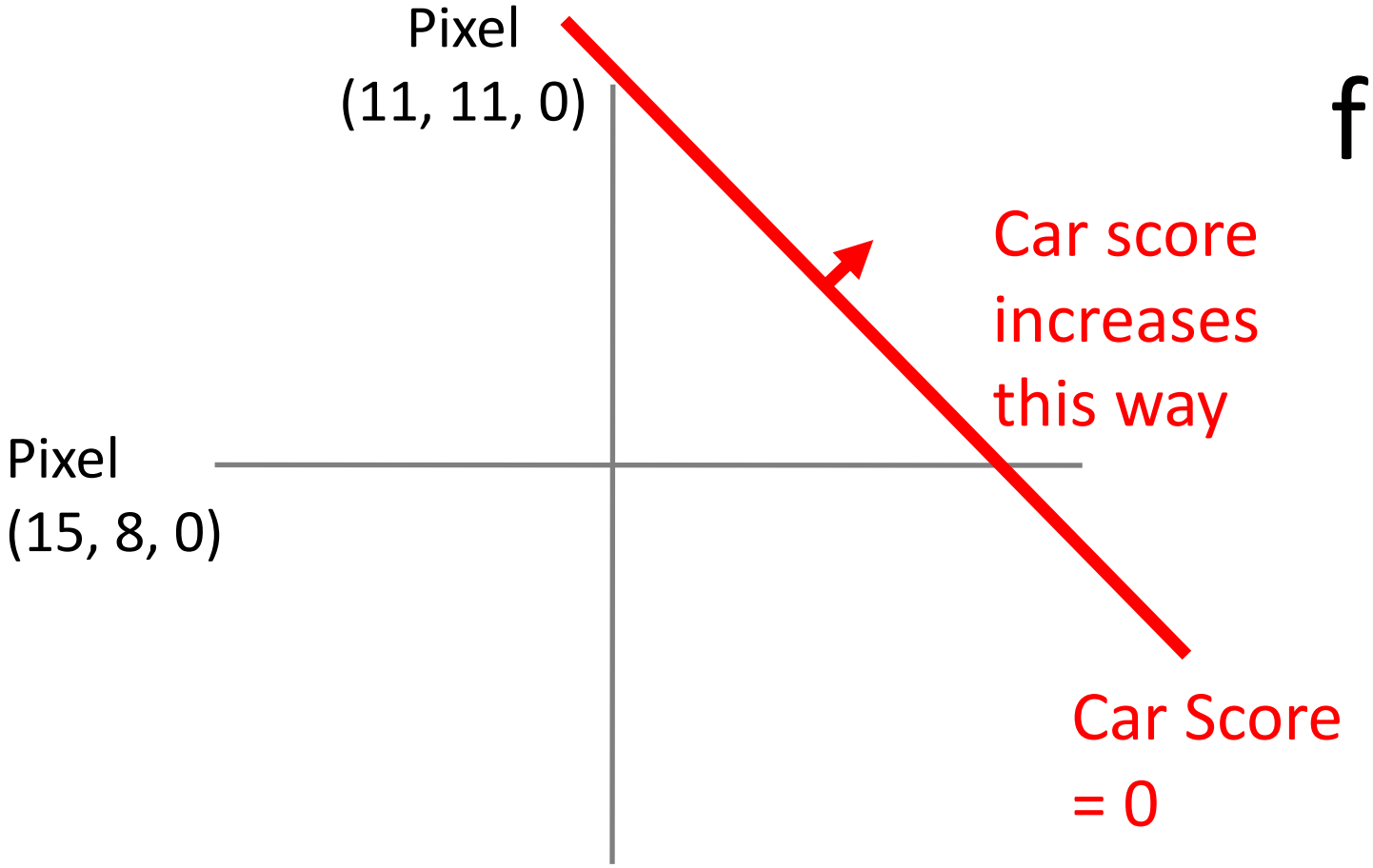


$$f(x,W) = Wx + b$$

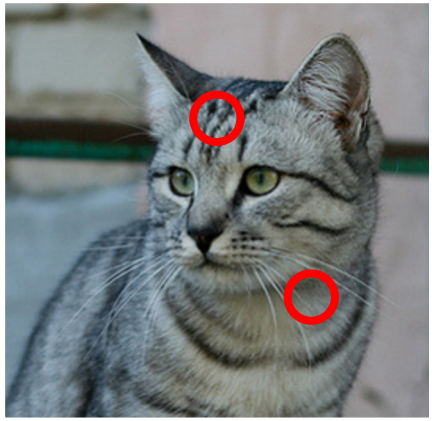


Array of **32x32x3** numbers
(3072 numbers total)

Interpreting a Linear Classifier: Geometric Viewpoint

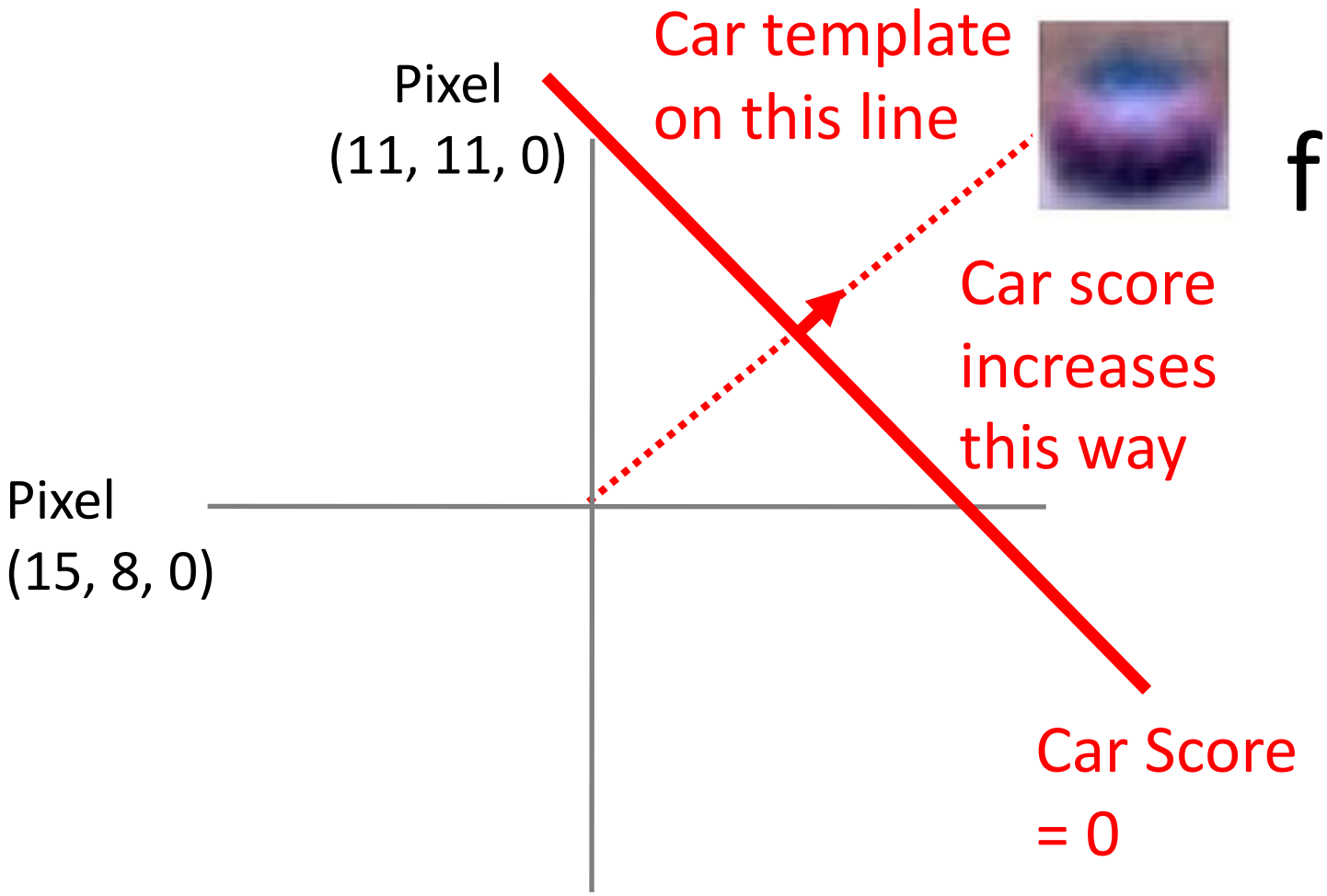


$$f(x,W) = Wx + b$$

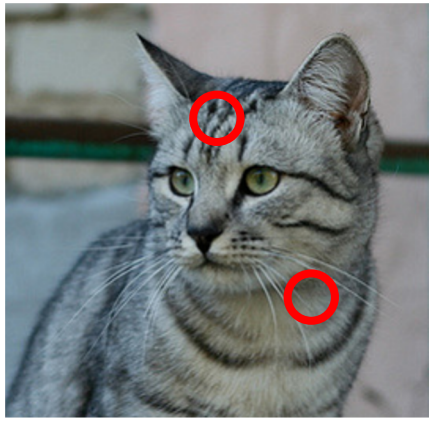


Array of **32x32x3** numbers
(3072 numbers total)

Interpreting a Linear Classifier: Geometric Viewpoint

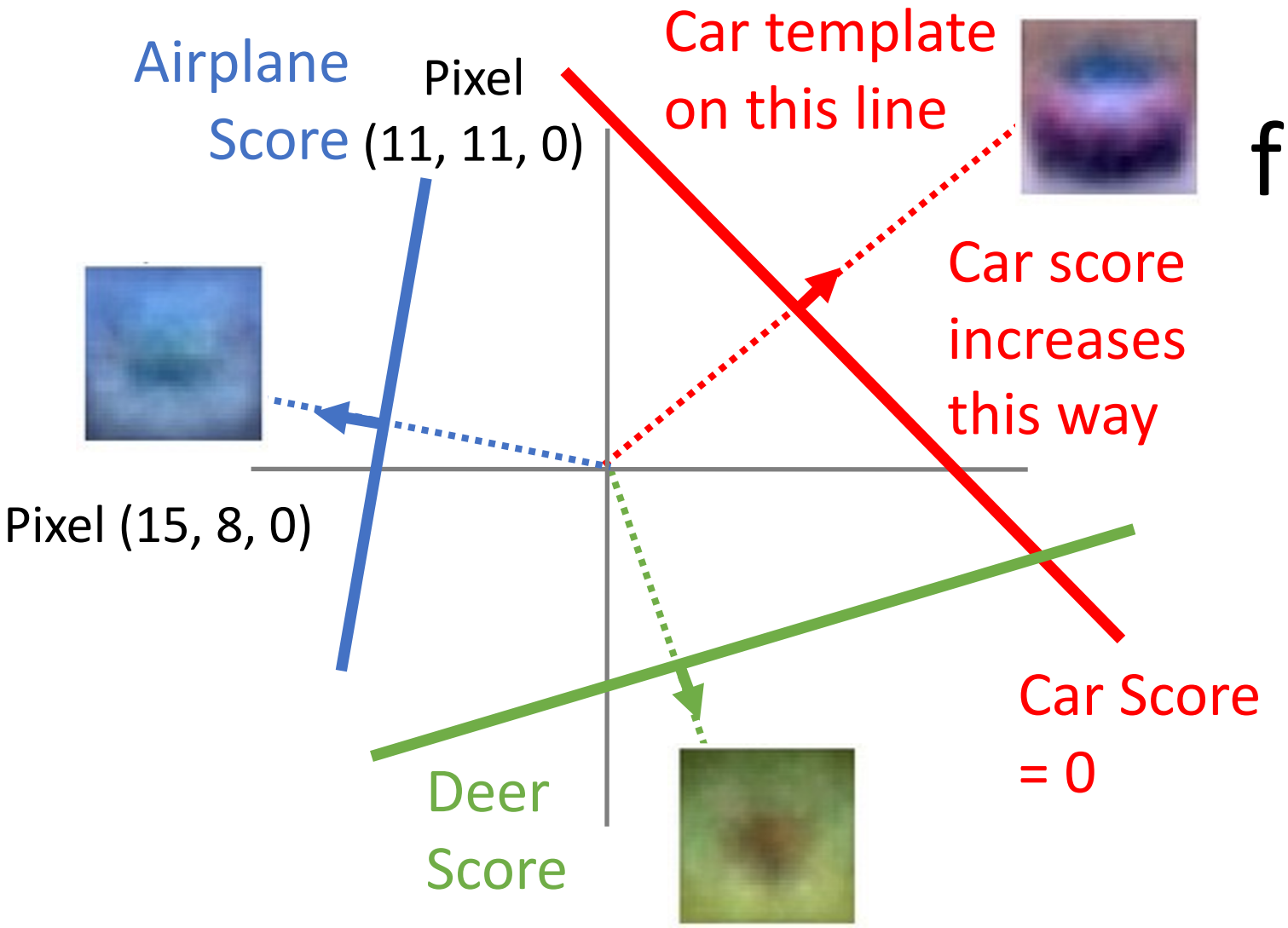


$$f(x,W) = Wx + b$$

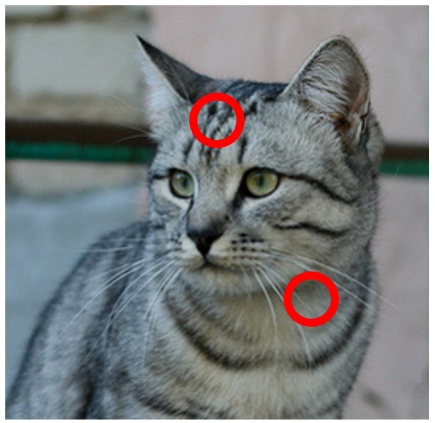


Array of **32x32x3** numbers
(3072 numbers total)

Interpreting a Linear Classifier: Geometric Viewpoint

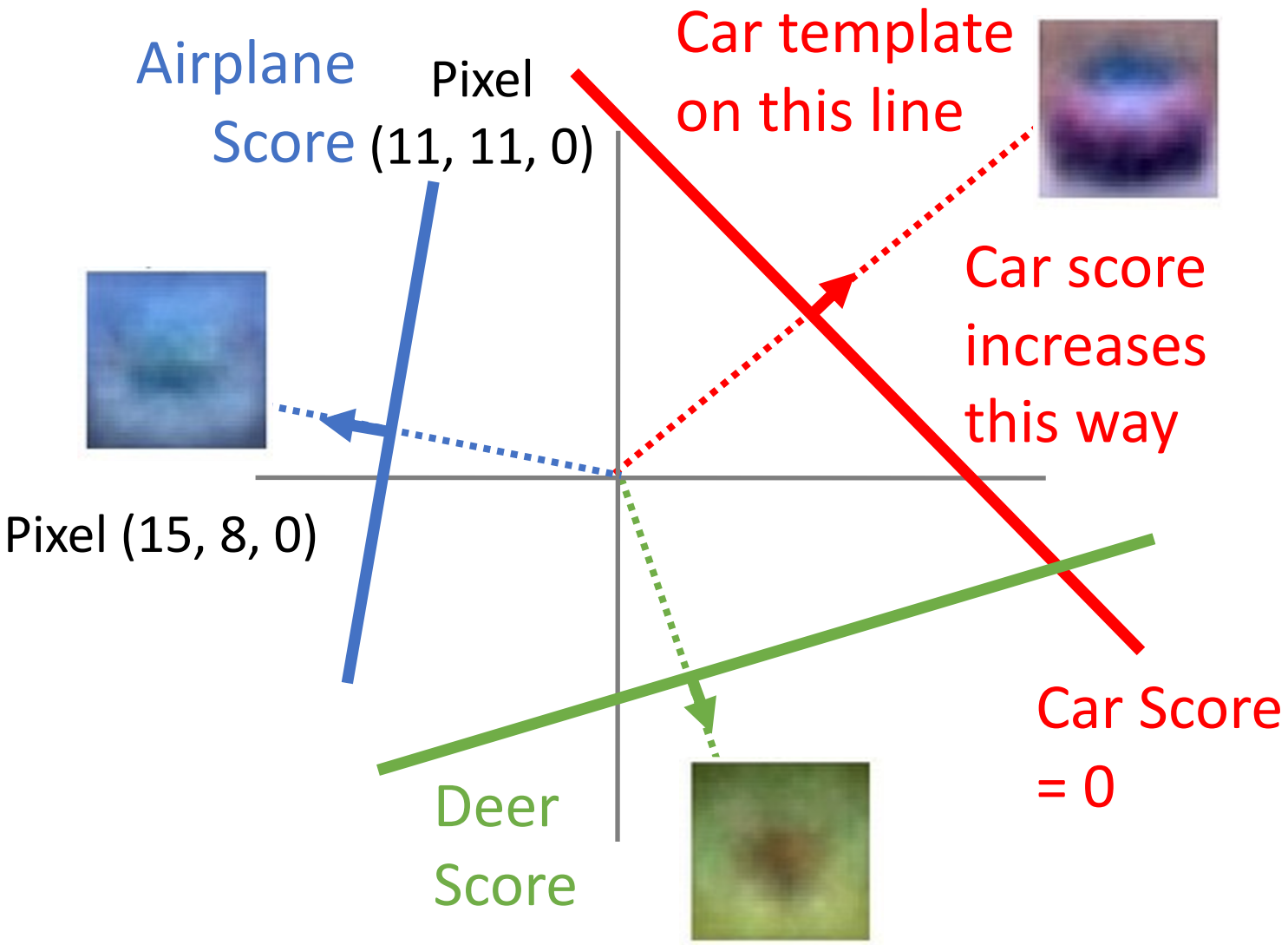


$$f(x,W) = Wx + b$$

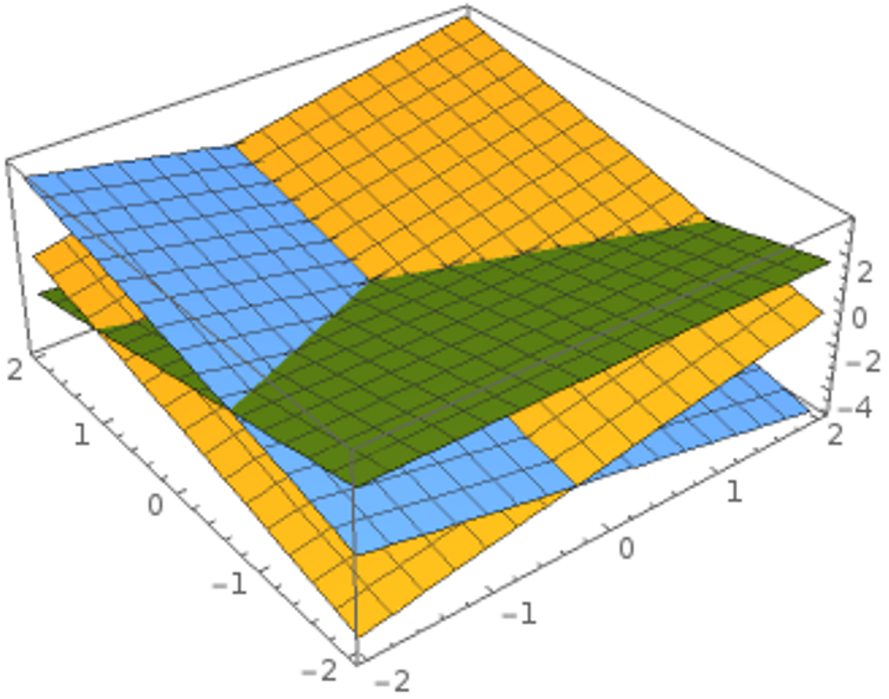


Array of **32x32x3** numbers
(3072 numbers total)

Interpreting a Linear Classifier: Geometric Viewpoint



Hyperplanes carving up a high-dimensional space

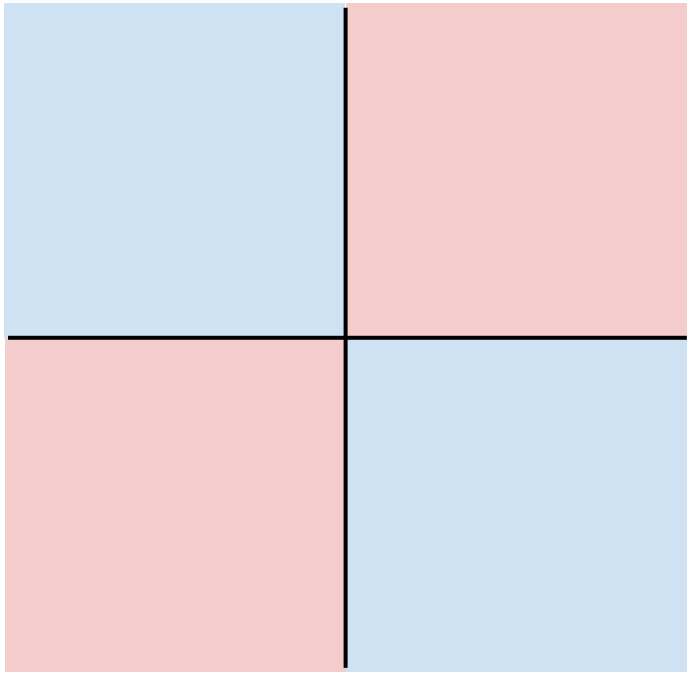


Plot created using Wolfram Cloud

Hard Cases for a Linear Classifier

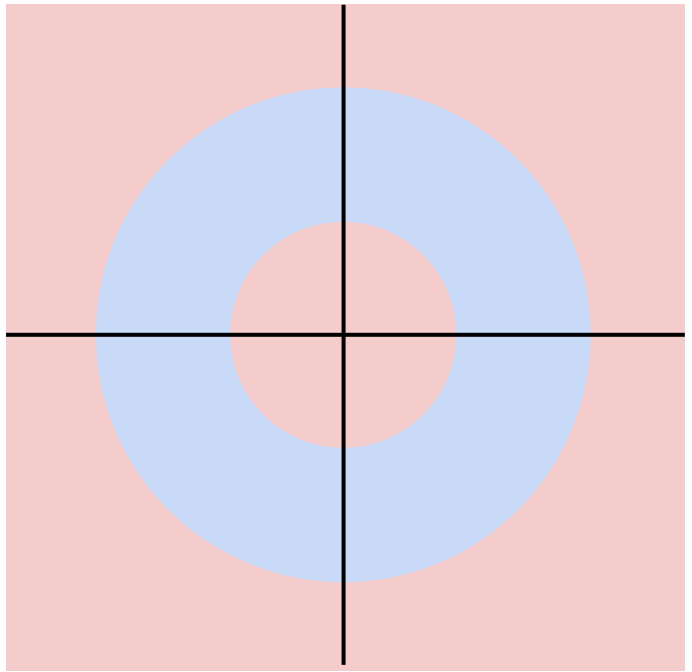
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



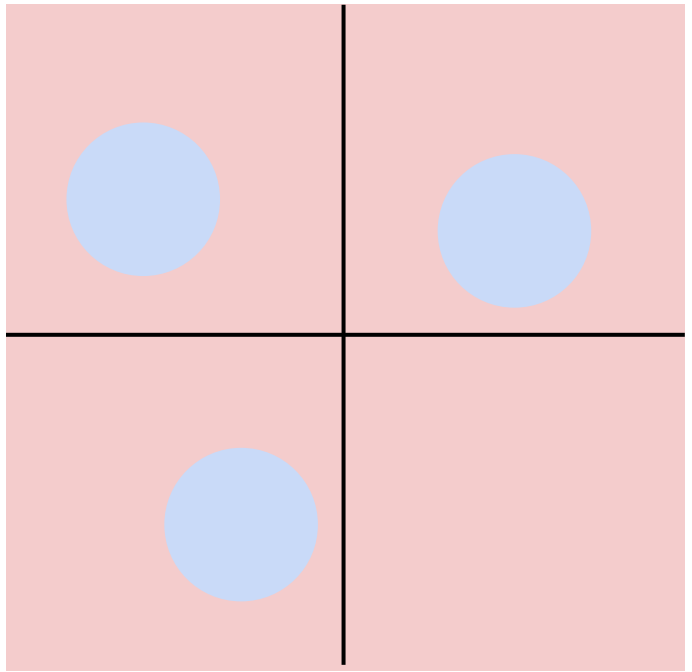
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

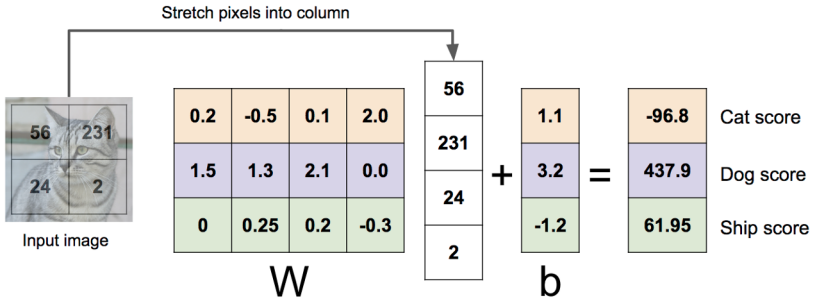
Class 2:
Everything else



Linear Classifier: Three Viewpoints

Algebraic Viewpoint

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



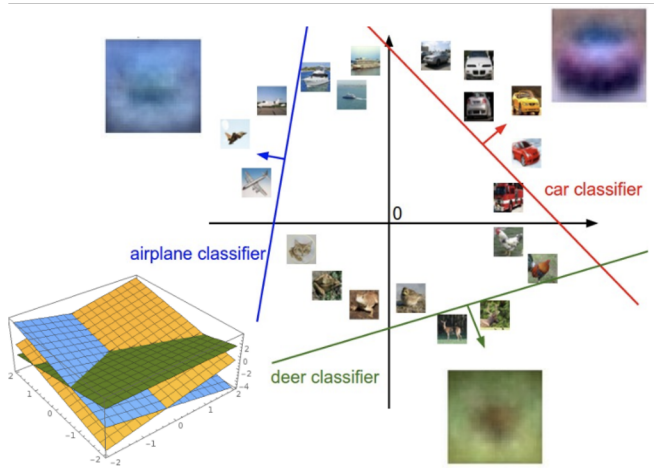
Visual Viewpoint

One template per class



Geometric Viewpoint

Hyperplanes cutting up space



So Far: Defined a linear score function

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





airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

Given a W, we can compute class scores for an image x.

But how can we actually choose a good W?

Cat image by Nikita is licensed under CC-BY 2.0; Car image is CC0 1.0 public domain; Frog image is in the public domain

Choosing a good W

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





airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

TODO:

1. Use a **loss function** to quantify how good a value of W is

2. Find a W that minimizes the loss function (**optimization**)

Loss Function

A **loss function** tells how good our current classifier is

Low loss = good classifier
High loss = bad classifier

(Also called: **objective function**; **cost function**)

Loss Function

A **loss function** tells how good our current classifier is

Low loss = good classifier
High loss = bad classifier

(Also called: **objective function**; **cost function**)

Negative loss function sometimes called **reward function**, **profit function**, **utility function**, **fitness function**, etc

Loss Function

A **loss function** tells how good our current classifier is

Low loss = good classifier
High loss = bad classifier

(Also called: **objective function**; **cost function**)

Negative loss function sometimes called **reward function**, **profit function**, **utility function**, **fitness function**, etc

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss Function

A **loss function** tells how good our current classifier is

Low loss = good classifier
High loss = bad classifier

(Also called: **objective function**; **cost function**)

Negative loss function sometimes called **reward function**, **profit function**, **utility function**, **fitness function**, etc

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss for a single example is

$$L_i(f(x_i, W), y_i)$$

Loss Function

A **loss function** tells how good our current classifier is

Low loss = good classifier
High loss = bad classifier

(Also called: **objective function**; **cost function**)

Negative loss function sometimes called **reward function**, **profit function**, **utility function**, **fitness function**, etc

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss for a single example is

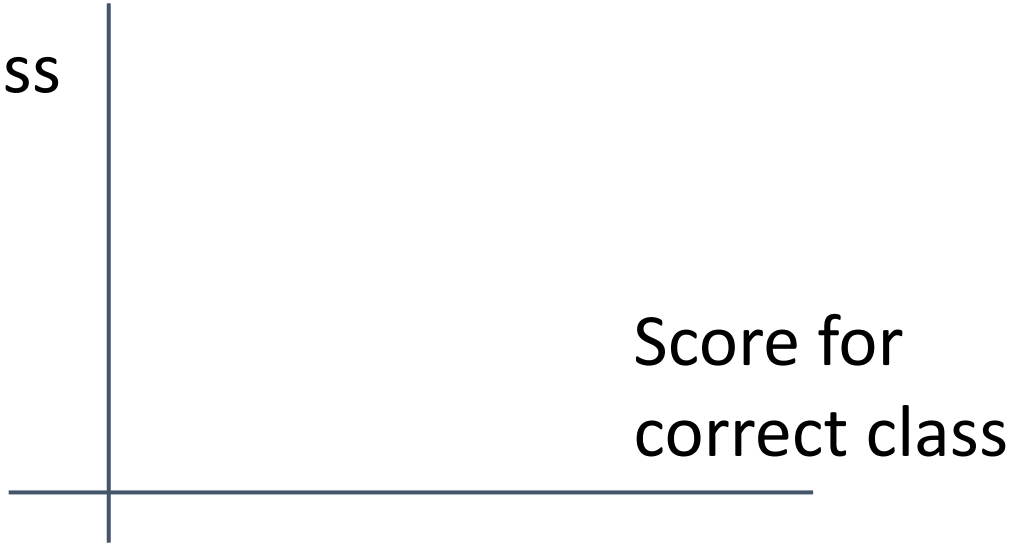
$$L_i(f(x_i, W), y_i)$$

Loss for the dataset is average of per-example losses:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

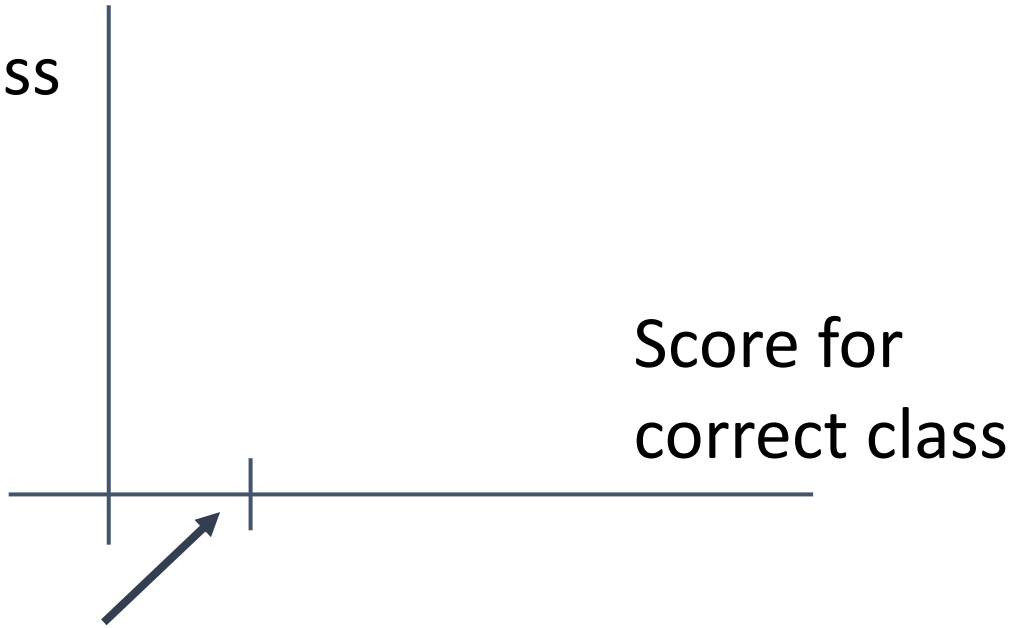
Multiclass SVM Loss

”The score of the correct class should be higher than all the other scores”



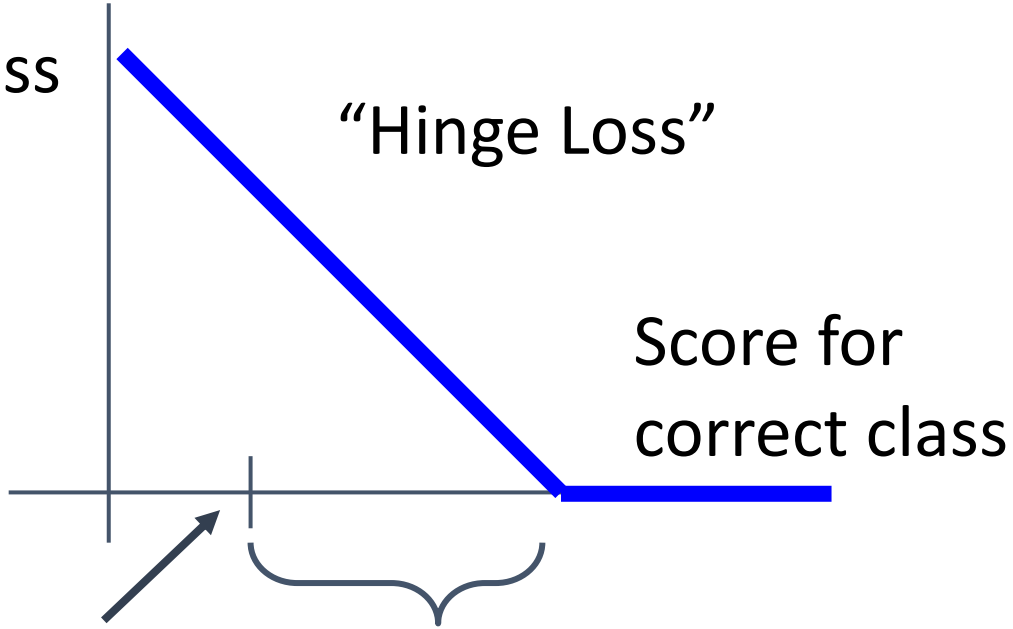
Multiclass SVM Loss

”The score of the correct class should be higher than all the other scores”



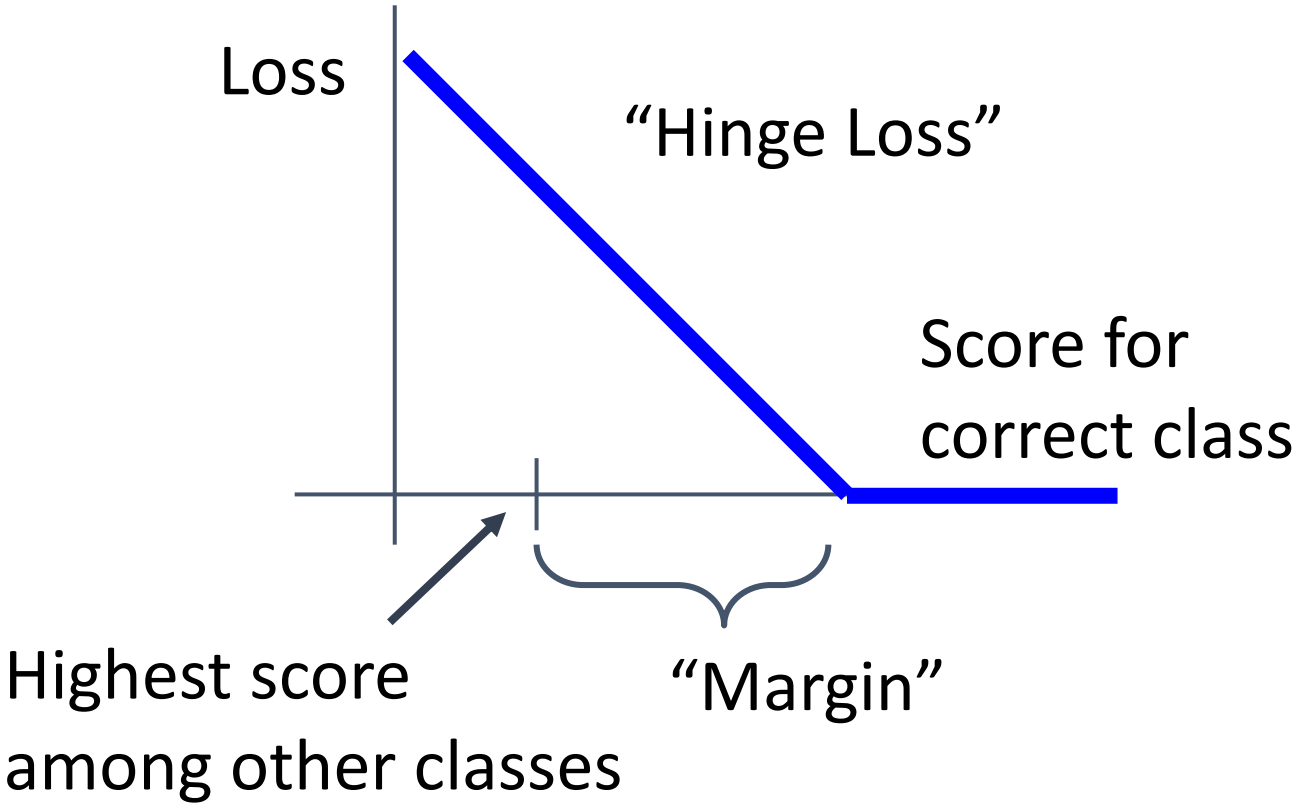
Multiclass SVM Loss

”The score of the correct class should be higher than all the other scores”



Multiclass SVM Loss

"The score of the correct class should be higher than all the other scores"



Given an example (x_i, y_i)
(x_i is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

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

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

Data loss: Model predictions should match training data

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:

Dropout

Batch normalization

Cutout, Mixup, Stochastic depth, etc…

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

λ = regularization strength (hyperparameter)

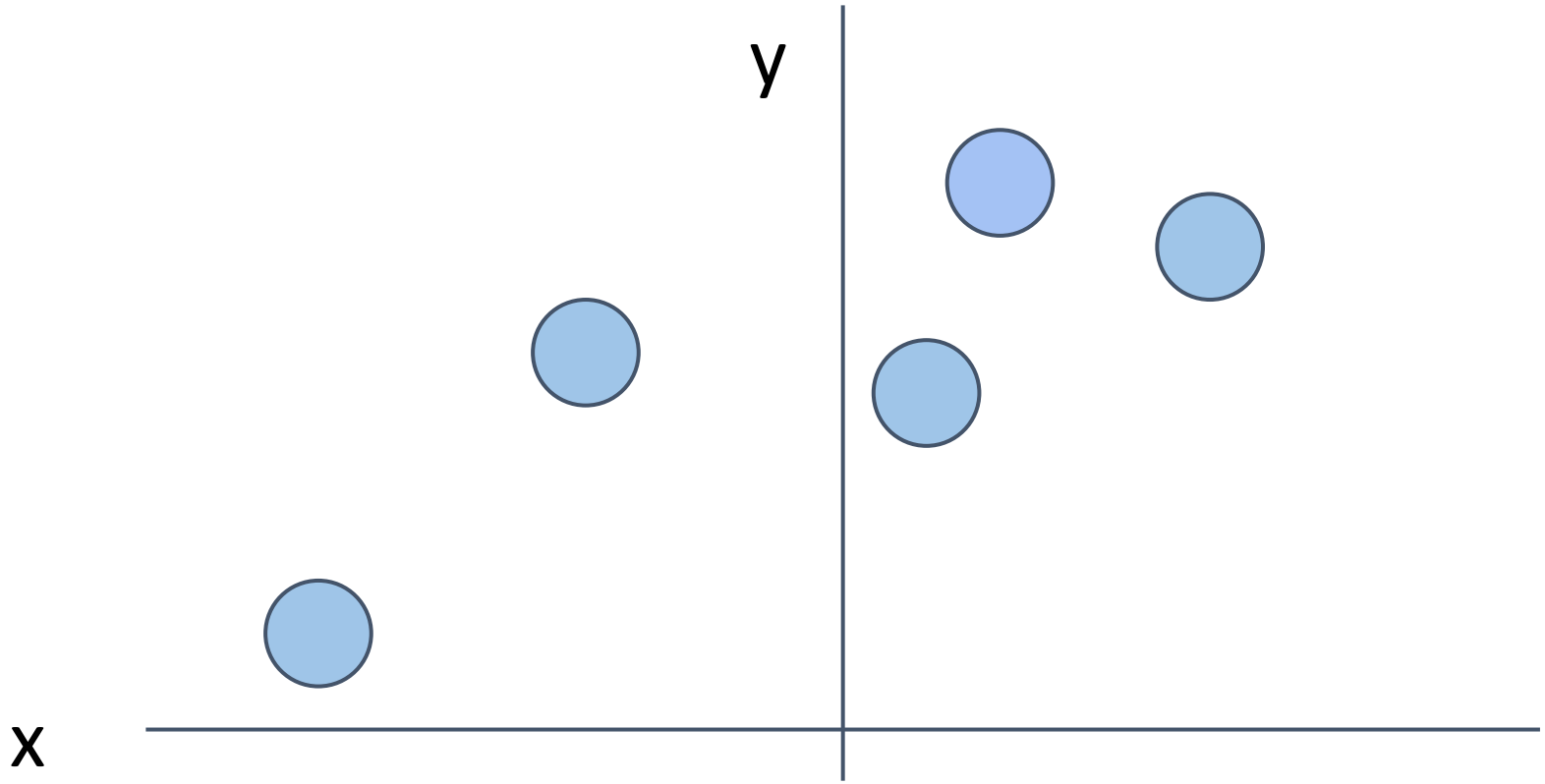
Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

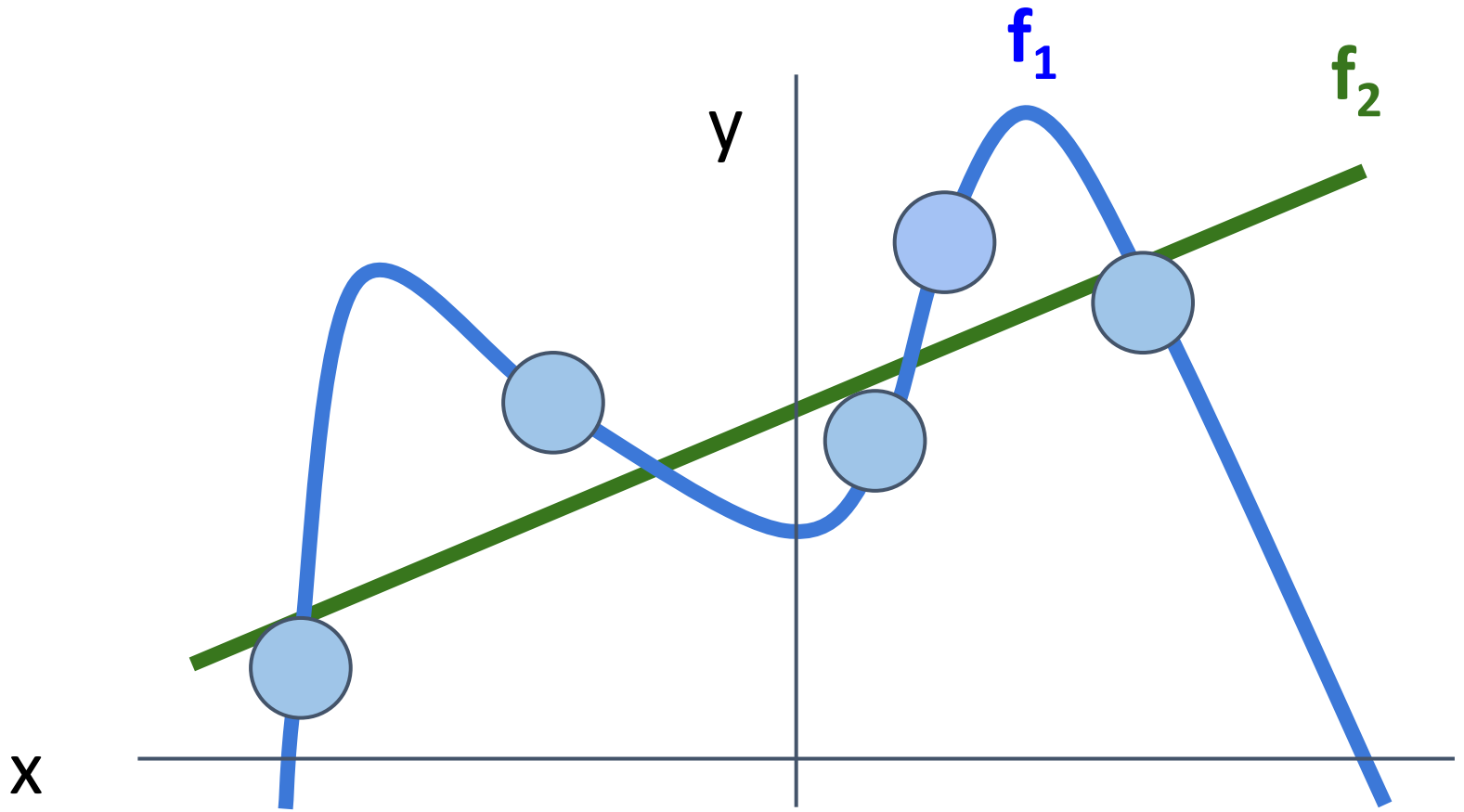
Purpose of Regularization:

- Express preferences in among models beyond ”minimize training error”
- Avoid **overfitting**: Prefer simple models that generalize better
- Improve optimization by adding curvature

Regularization: Prefer Simpler Models



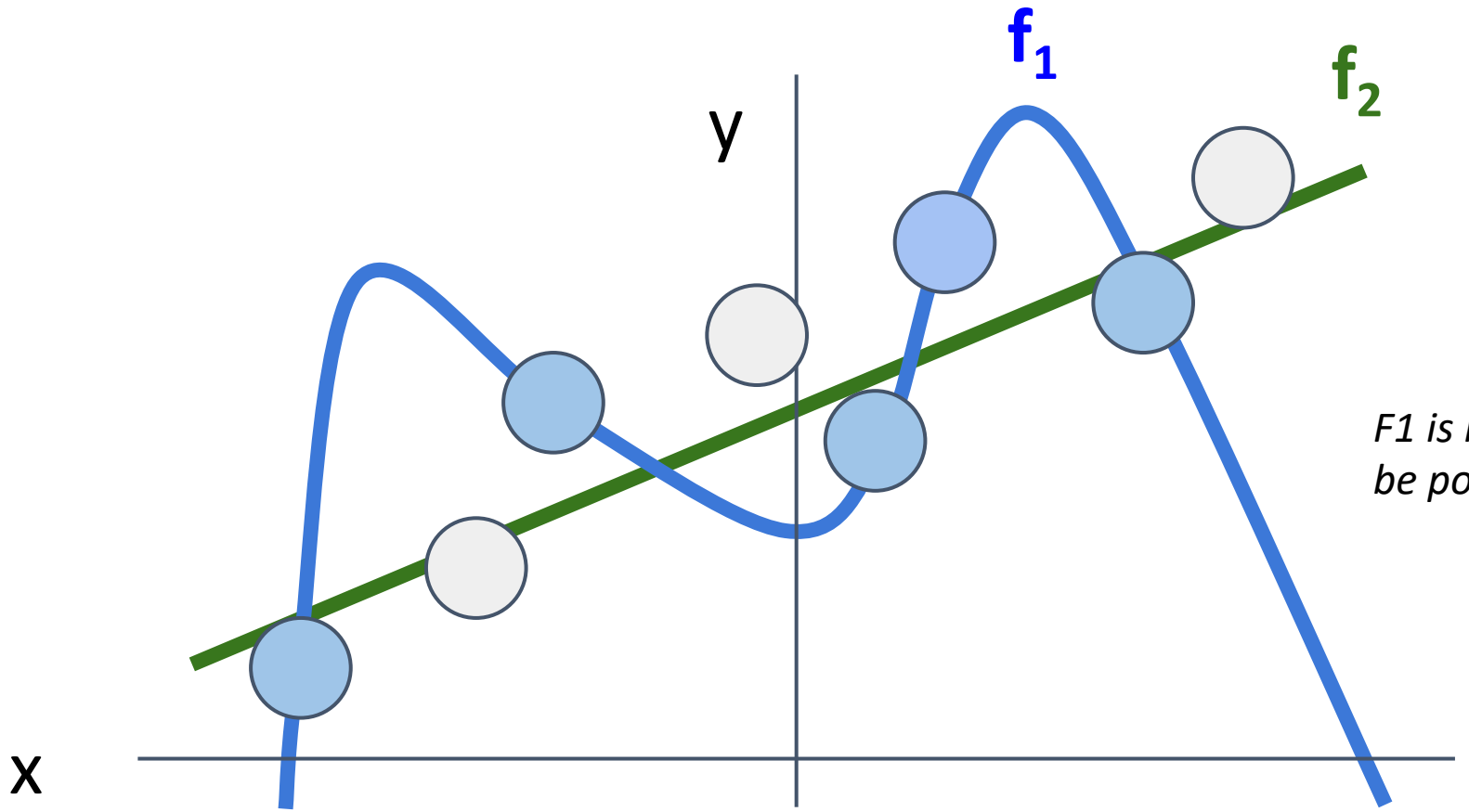
Regularization: Prefer Simpler Models



The model f_1 fits the training data perfectly

The model f_2 has training error, but is simpler

Regularization: Prefer Simpler Models

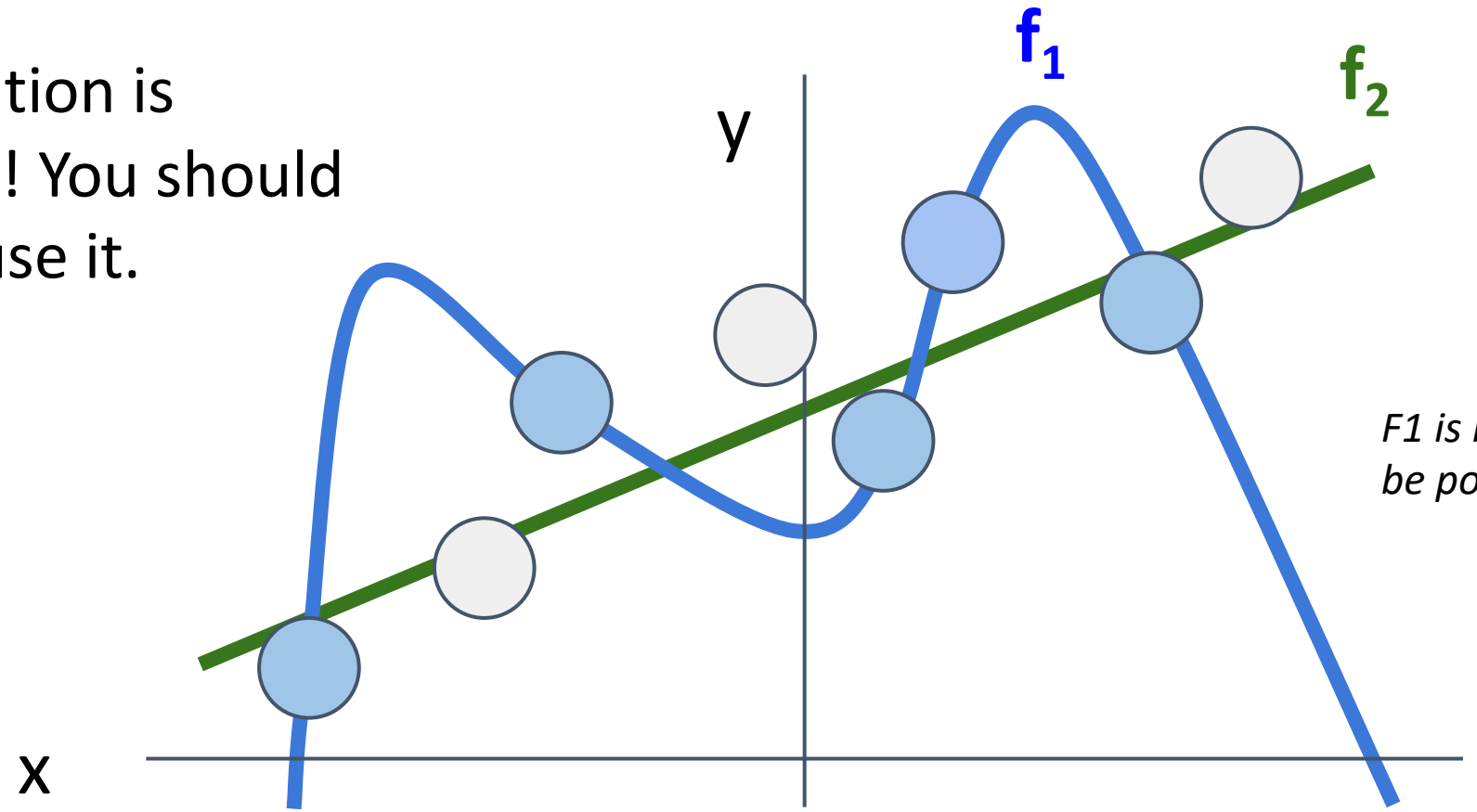


F1 is not a linear model; could be polynomial regression, etc

Regularization pushes against fitting the data *too* well so we don't fit noise in the data

Regularization: Prefer Simpler Models

Regularization is important! You should (usually) use it.



F1 is not a linear model; could be polynomial regression, etc

Regularization pushes against fitting the data *too* well so we don't fit noise in the data

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



cat	**3.2**
car	5.1
frog	-1.7

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Unnormalized log-probabilities / logits

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

Probabilities must be >= 0

	Unnormalized log-probabilities / logits	exp	unnormalized probabilities
cat	**3.2**		**24.5**
car	5.1	→	164.0
frog	-1.7		0.18

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0 unnormalized probabilities	normalize	Probabilities must sum to 1 probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

$$L_i = -\log P(Y = y_i|X = x_i)$$

	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0 unnormalized probabilities	normalize	Probabilities must sum to 1 probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

$L_i = -\log(0.13)$
$= \mathbf{2.04}$

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

$$L_i = -\log P(Y = y_i|X = x_i)$$

	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0	normalize	Probabilities must sum to 1
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00
	Unnormalized log-probabilities / logits		unnormalized probabilities		probabilities

$L_i = -\log(0.13)$
$= \mathbf{2.04}$

Maximum Likelihood Estimation
Choose weights to maximize the likelihood of the observed data (next lecture)

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

$$L_i = -\log P(Y = y_i|X = x_i)$$

	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0	unnormalized probabilities	normalize	Probabilities must sum to 1	probabilities	Compare	Correct probs
cat	**3.2**	→	**24.5**		→	**0.13**		→	**1.00**
car	5.1		164.0			0.87			0.00
frog	-1.7		0.18			0.00			0.00

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

$$L_i = -\log P(Y = y_i|X = x_i)$$

	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0 unnormalized probabilities	normalize	Probabilities must sum to 1 probabilities	Compare	Correct probs
cat	**3.2**	→	**24.5**	→	**0.13**	→	**1.00**
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Kullback–Leibler divergence

$$D_{KL}(P||Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

$$L_i = -\log P(Y = y_i|X = x_i)$$

	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0: unnormalized probabilities	normalize	Probabilities must sum to 1: probabilities	Compare	Correct probs
cat	**3.2**	→	**24.5**	→	**0.13**	→	**1.00**
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Cross Entropy

$$H(P, Q) = H(p) + D_{KL}(P\|Q)$$

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Maximize probability of correct class

$$L_i = -\log P(Y = y_i|X = x_i)$$

Putting it all together:

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Maximize probability of correct class

$$L_i = -\log P(Y = y_i|X = x_i)$$

Putting it all together:

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Q: What is the min / max possible loss L_i?

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Maximize probability of correct class

$$L_i = -\log P(Y = y_i|X = x_i)$$

Putting it all together:

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

Q: What is the min / max possible loss L_i?

A: Min 0, max +infinity

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Maximize probability of correct class

$$L_i = -\log P(Y = y_i|X = x_i)$$

Putting it all together:

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Q: If all scores are small random values, what is the loss?

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Maximize probability of correct class

$$L_i = -\log P(Y = y_i|X = x_i)$$

Putting it all together:

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

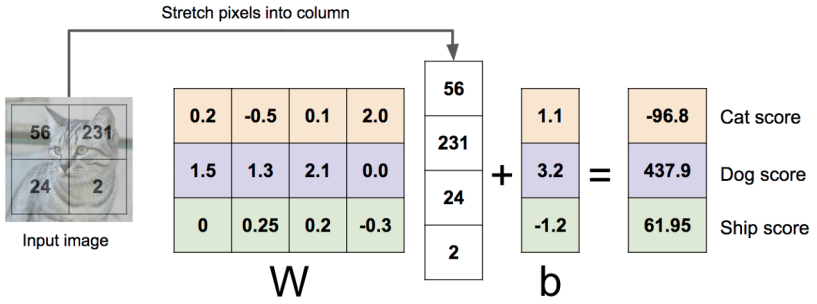
Q: If all scores are small random values, what is the loss?

A: -log(1/C)
log(10) ≈ 2.3

Recap: Three ways to think about linear classifiers

Algebraic Viewpoint

$f(x,W) = Wx$



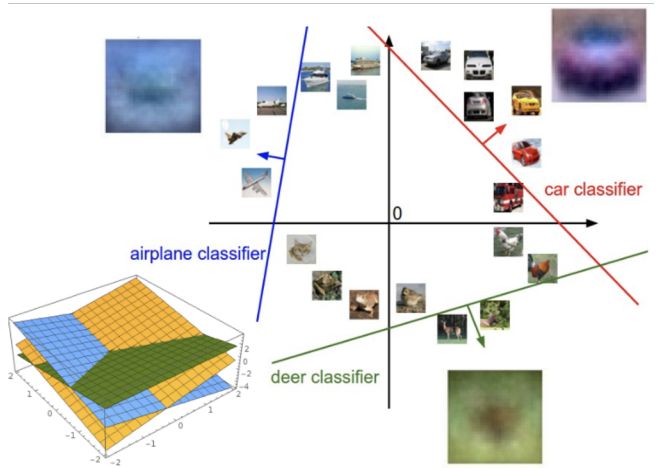
Visual Viewpoint

One template per class



Geometric Viewpoint

Hyperplanes cutting up space



Recap: Loss Functions quantify preferences

- We have some dataset of (x, y)
- We have a **score function:**
- We have a **loss function**:

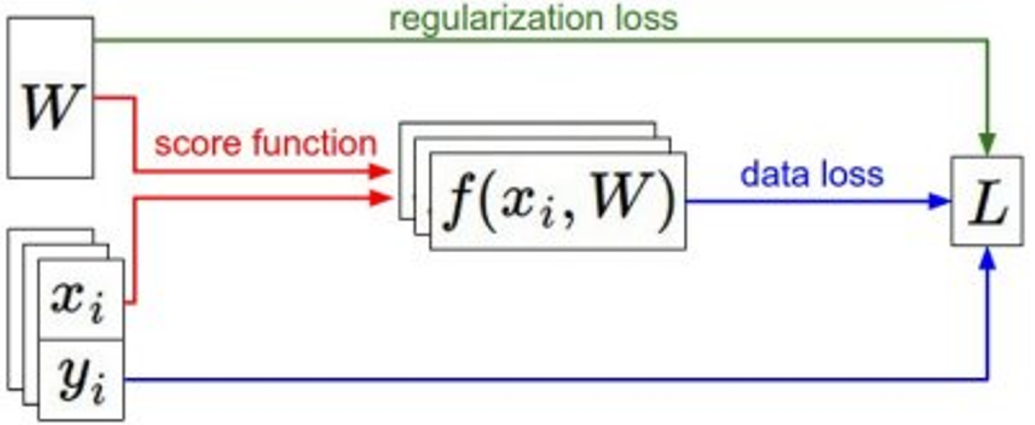
$$s = f(x; W) = Wx$$

Linear classifier

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$ Softmax

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

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss



Recap: Loss Functions quantify preferences

- We have some dataset of (x, y)
- We have a **score function:**
- We have a **loss function**:

Q: How do we find the best W?

$$s = f(x; W) = Wx$$

Linear classifier

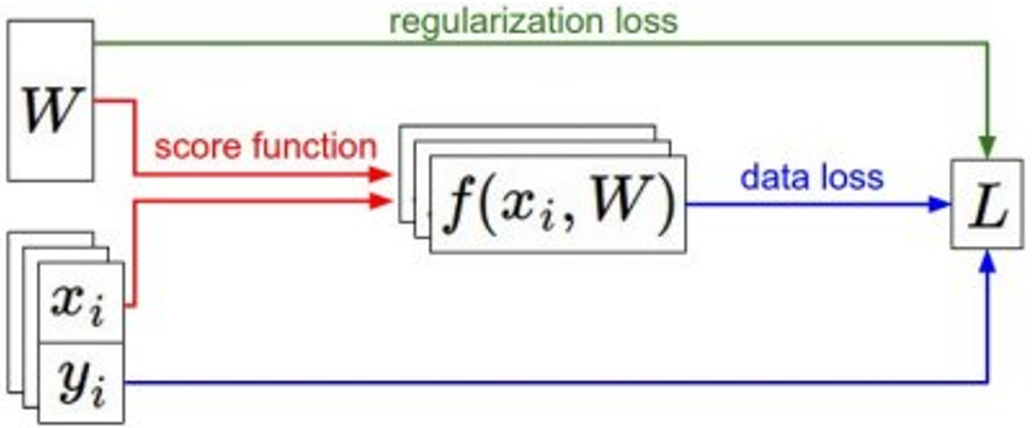
A: Next lecture

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$ Softmax

SVM

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

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss

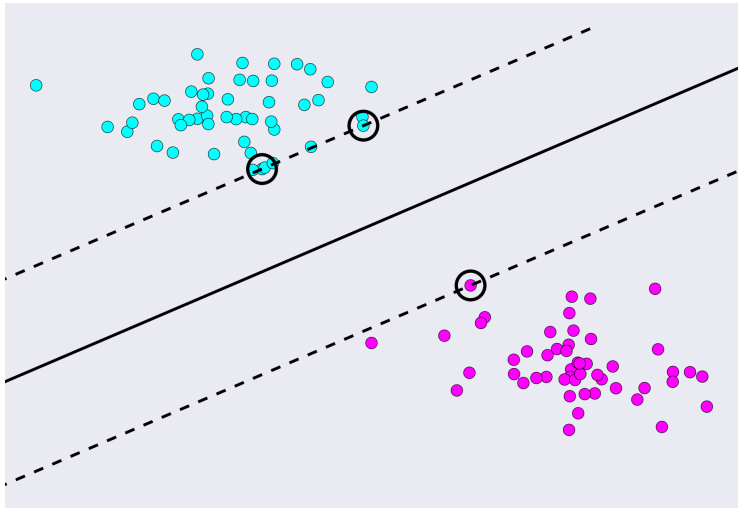


Machine learning

- Why learning?
- Nearest neighbors
- Bayesian classification
- Logistic regression
- Support vector machines & random forests (lightening speed)
- Clustering
- Principal component analysis

Support vector machines (SVMs)

- Maximize the *margin* between the decision surfaces
- The only points that matter are the circled *support vectors*



n margin classifier, we need to
$l_i = \mathbf{w} \cdot \mathbf{x}_i + b$ (5.17) have an
ıote this more compactly, let

$$\hat{t}_i = 2t_i - 1, \quad \hat{t}_i \in \{-1, 1\}$$

an now re-write the inequality

$$\hat{t}_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1.$$

simply find the smallest norm
ization problem

$$\arg\min_{\mathbf{w},b} \|\mathbf{w}\|^2$$

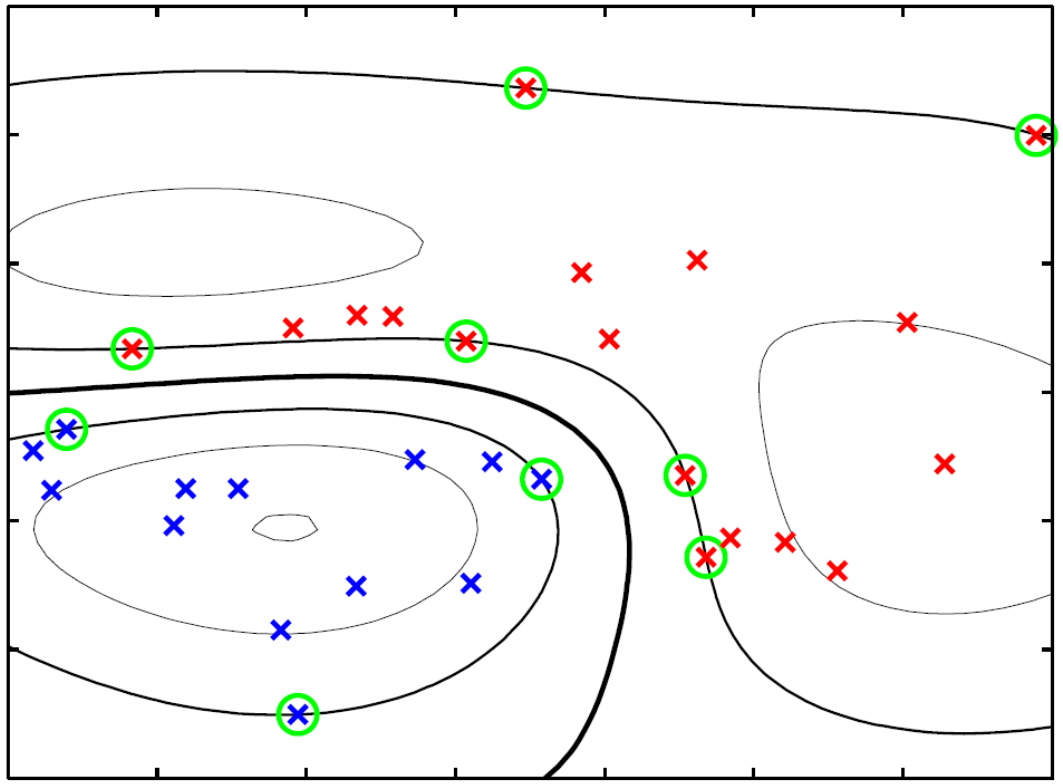
assic *quadratic programming*

Kernelized support vector machines

- Replace linear function with a sum of *kernel functions* (e.g., Gaussian bumps)

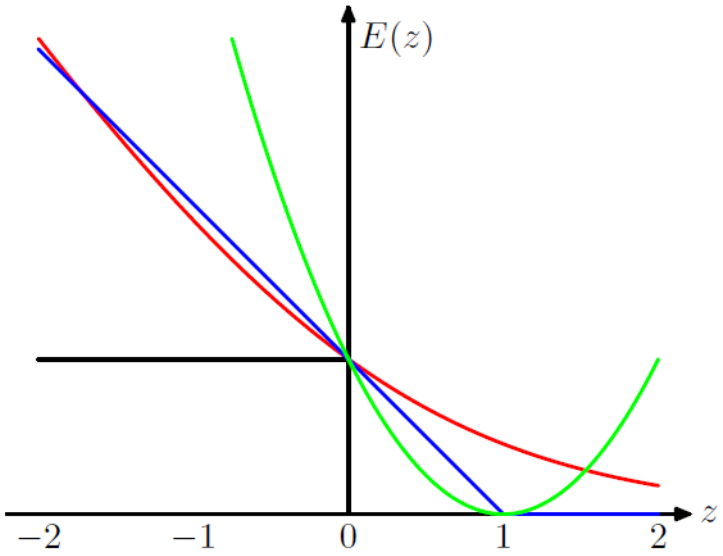
$$l = f(\mathbf{x}) = \sum_k \mathbf{w}_k \phi(\|\mathbf{x} - \mathbf{x}_k\|).$$

- Circled points are *support vectors*, lie on $f = \pm 1$ surfaces ($f = 0$ is the dark curve)



Hinge loss vs. logistic regression

Figure 7.5 Plot of the 'hinge' error function used in support vector machines, shown in blue, along with the error function for logistic regression, rescaled by a factor of $1/\ln(2)$ so that it passes through the point $(0, 1)$, shown in red. Also shown are the misclassification error in black and the squared error in green.



remaining points we have $\xi_n = 1 - y_n t_n$. Thus the objective function (7.21) can be written (up to an overall multiplicative constant) in the form

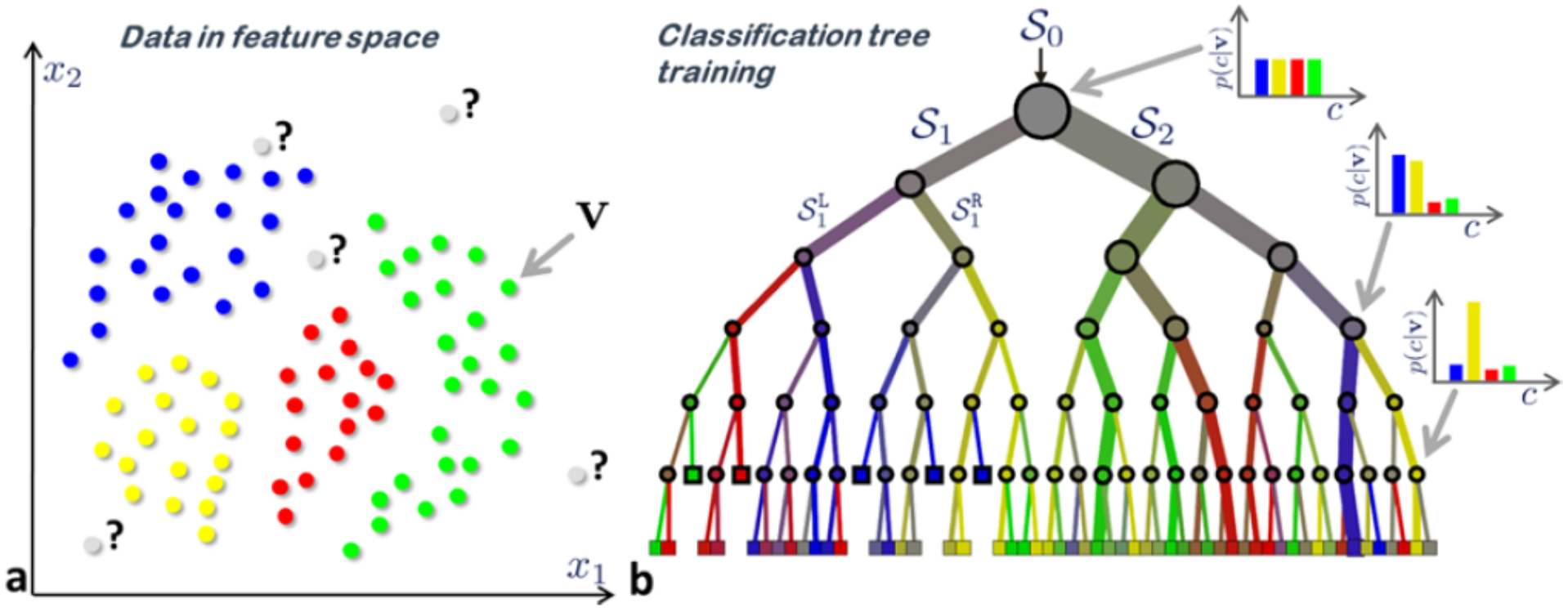
$$\sum_{n=1}^{N} E_{\mathrm{SV}}(y_n t_n) + \lambda \|\mathbf{w}\|^2 \tag{7.44}$$

where $\lambda = (2C)^{-1}$, and $E_{\mathrm{SV}}(\cdot)$ is the *hinge* error function defined by

$$E_{\mathrm{SV}}(y_n t_n) = [1 - y_n t_n]_+ \tag{7.45}$$

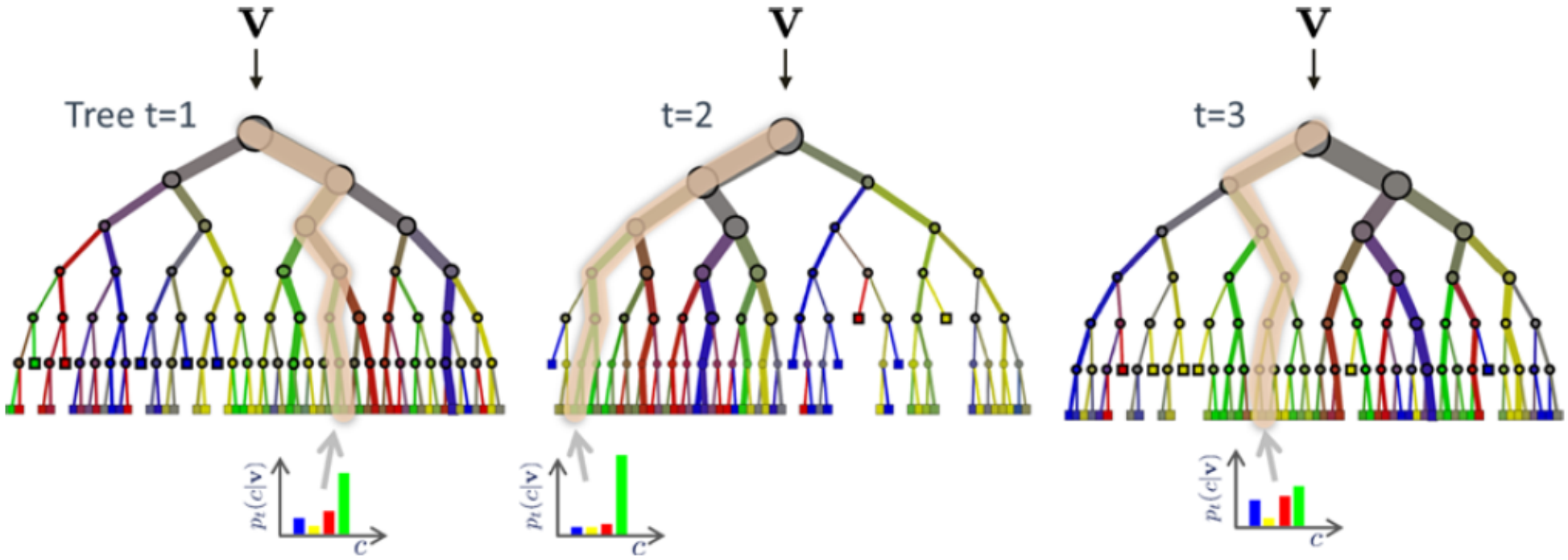
where $[\cdot]_+$ denotes the positive part. The hinge error function, so-called because of its shape, is plotted in Figure 7.5. It can be viewed as an approximation to the misclassification error, i.e., the error function that ideally we would like to minimize, which is also shown in Figure 7.5.

Decision trees (lighting speed)



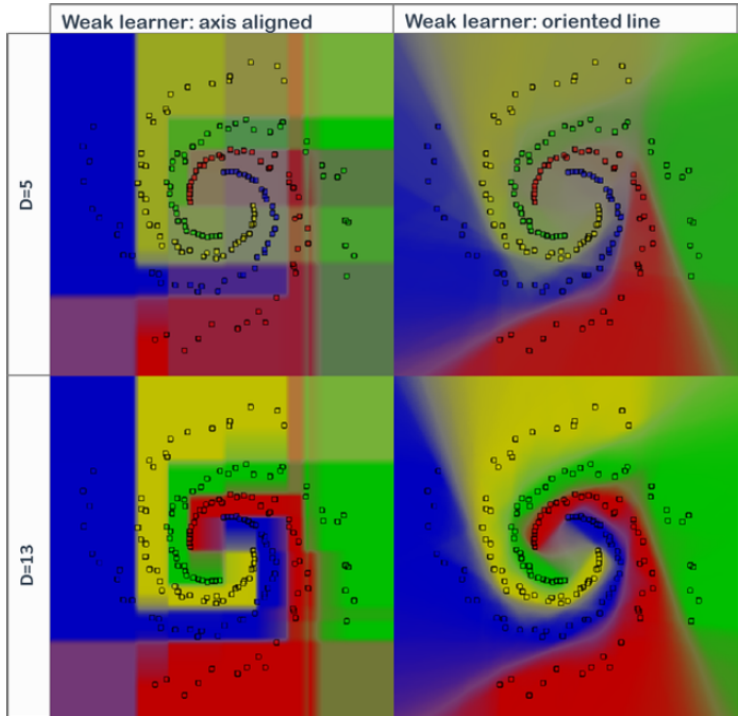
D = tree depth

Random forest

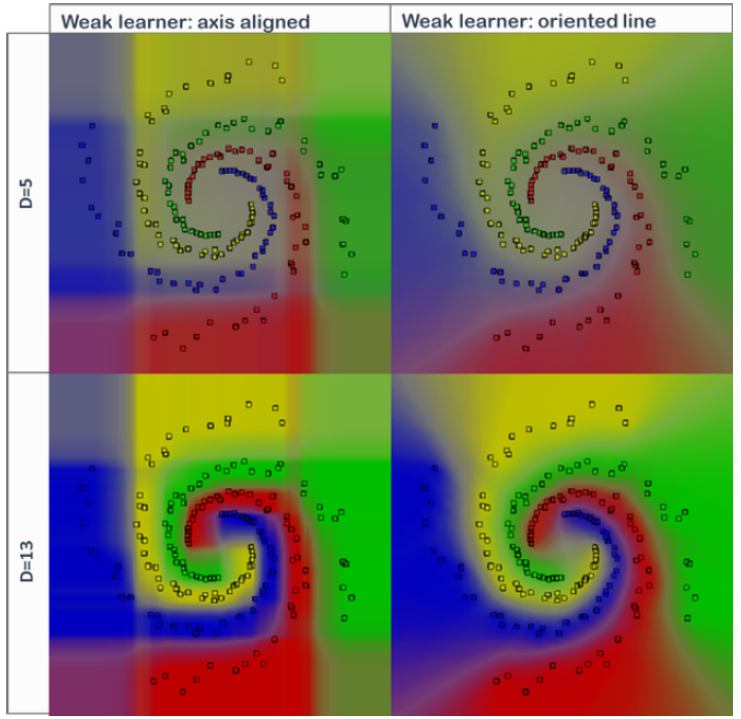


T = # trees

Random forest



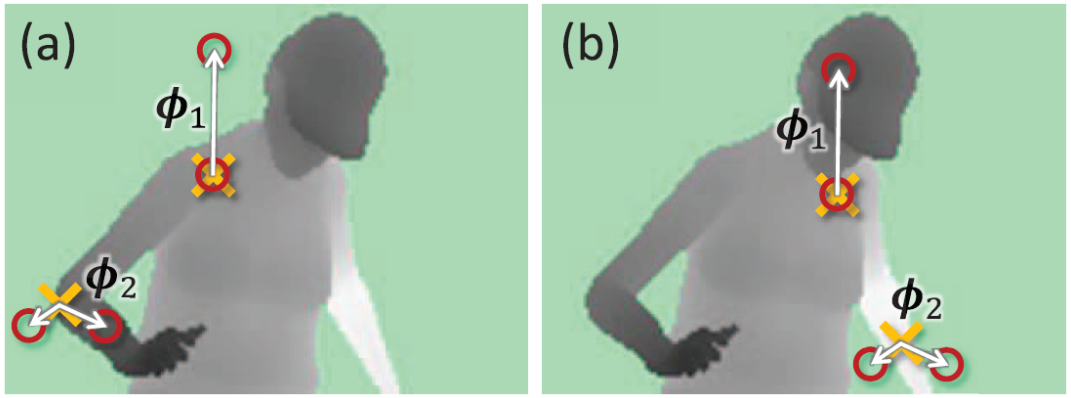
ρ = 500

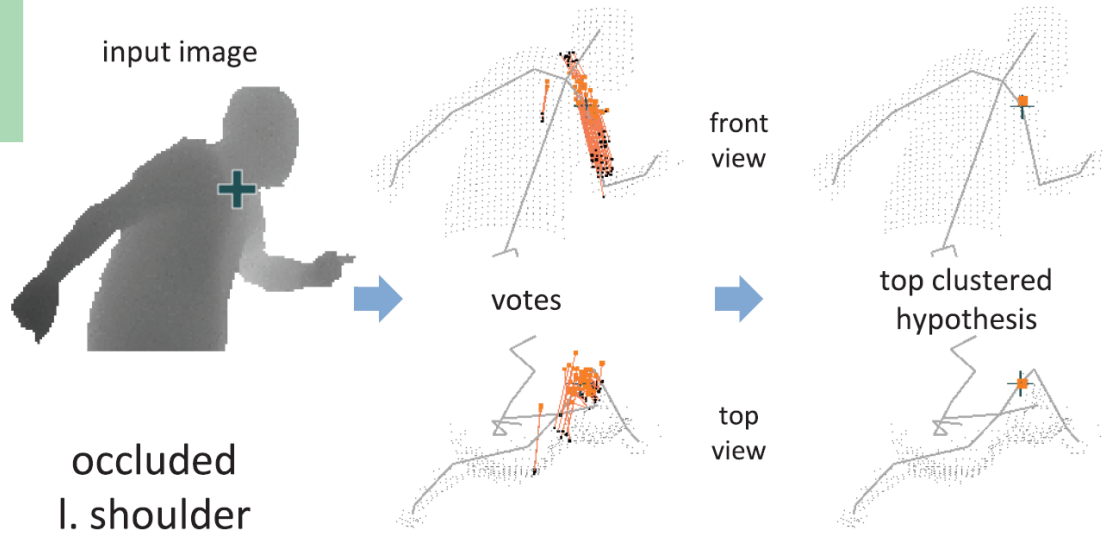


ρ = 5

ρ = # samples at each node at construction time

Application: Kinect body pose estimation





Lecture recap: machine learning

- Why learning?
- Nearest neighbors
- Bayesian classification
- Logistic regression
- Support vector machines
- Unsupervised learning:
  - Clustering
  - Principal component analysis