Lecture 2:
Image Classification

Assignment 1 Released

- http://web.eecs.umich.edu/~justincj/teaching/eecs498/assignment1.html
- Uses Python, PyTorch, and Google Colab
- Introduction to PyTorch Tensors
- K-Nearest Neighbor classification
- Due **Sunday September 15, 11:59pm EDT**

Assignment 1 Released

Q1: PyTorch 101 (50 points)

Open in Colab

The notebook **pytorch101.ipynb** will walk you through the basics of working with tensors in PyTorch.

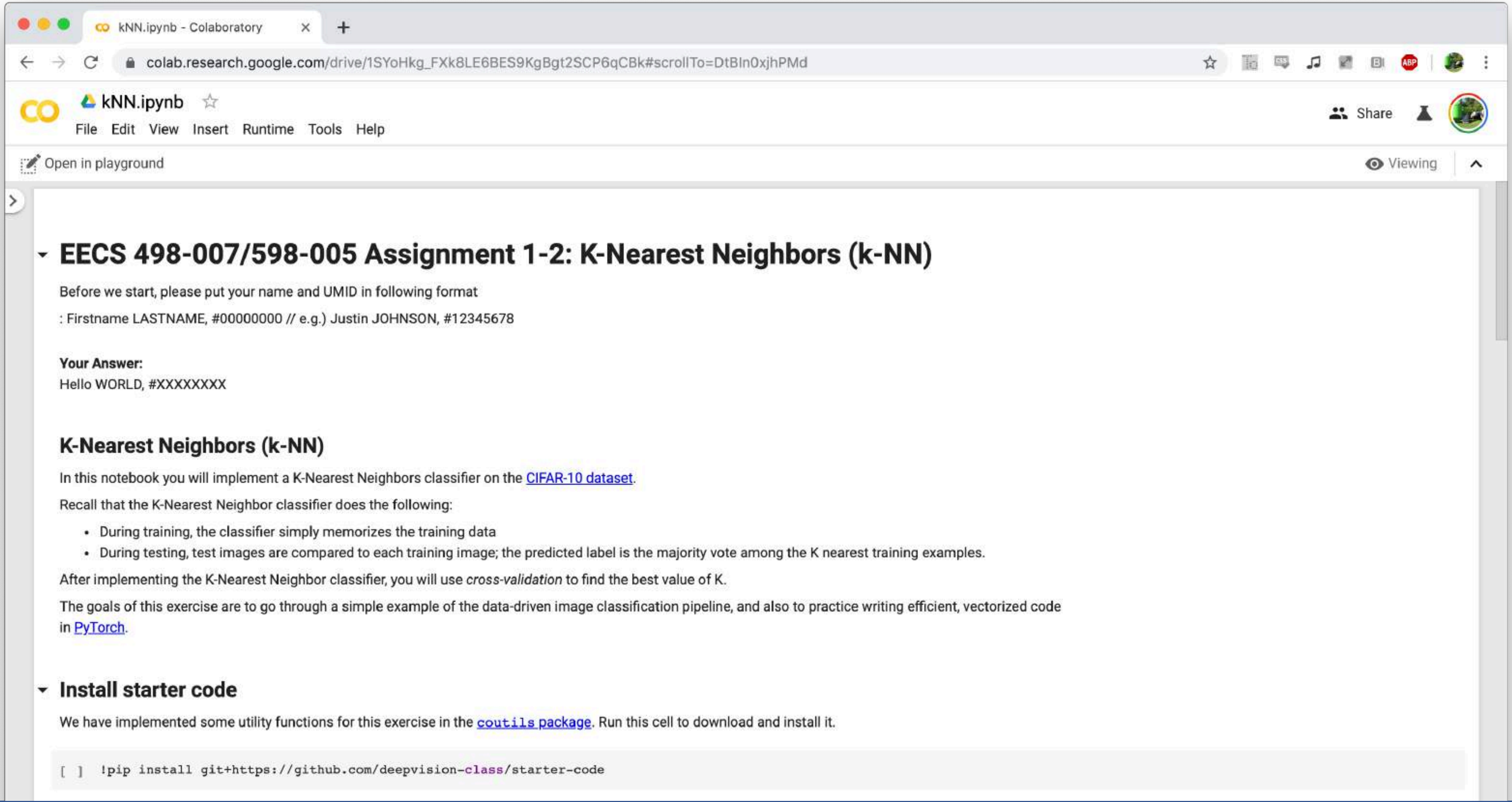
Q2: k-Nearest Neighbor classifier (50 points)

Open in Colab

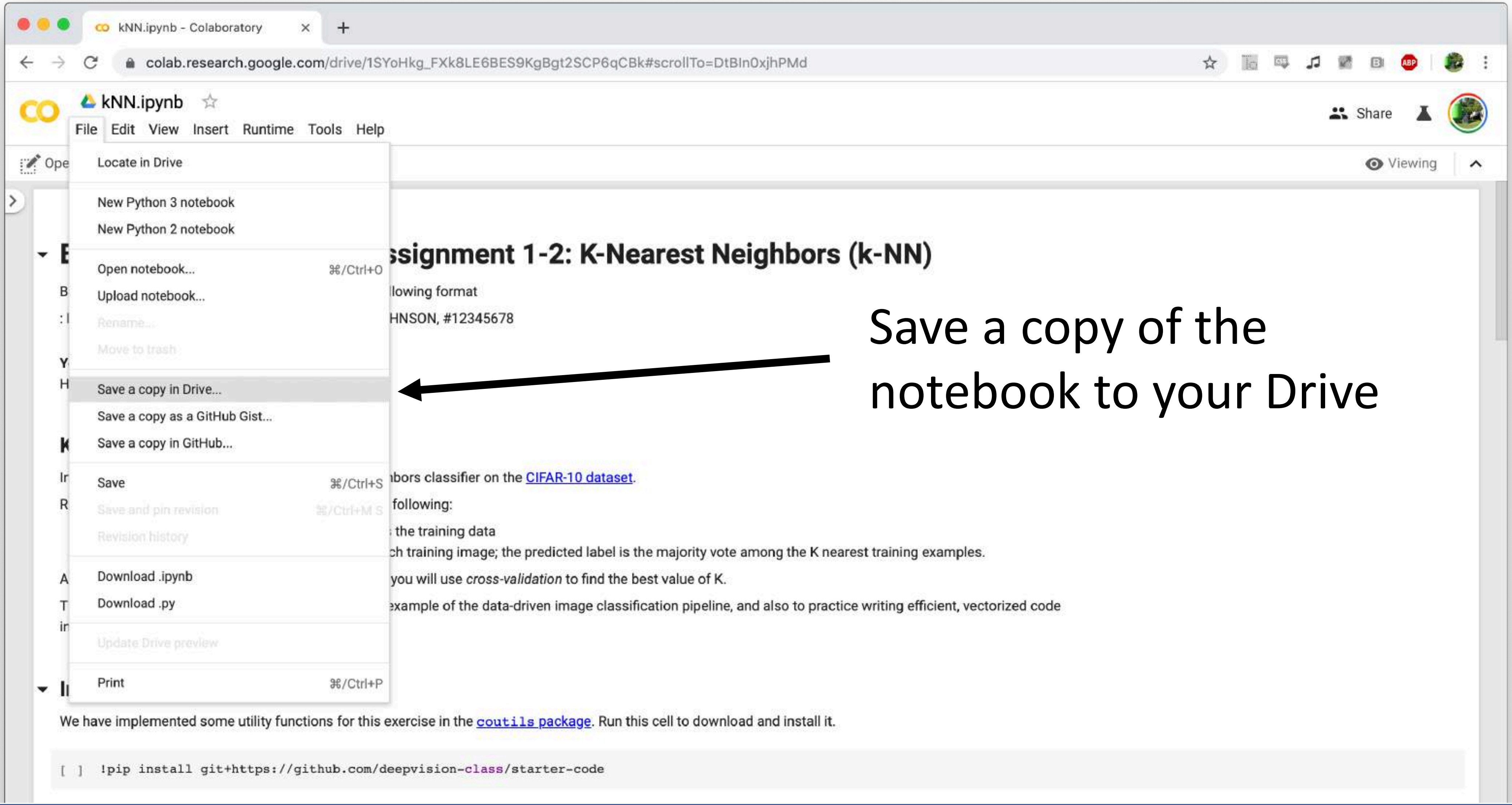
Click Here

The notebook **kNN.ipynb** will walk you through implementing a kNN classifier.

Google Colab: Cloud Computing in the Browser

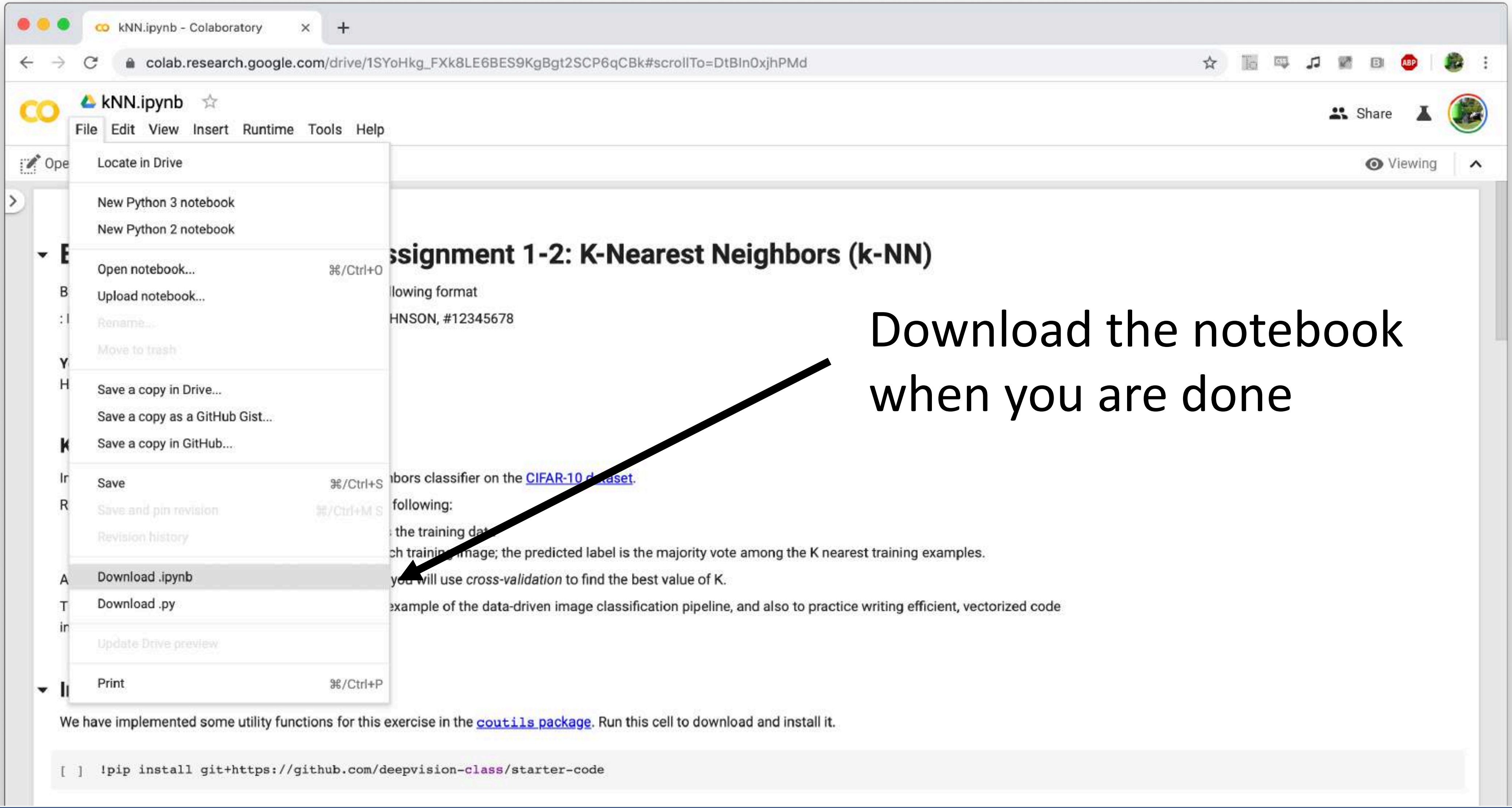


Google Colab: Cloud Computing in the Browser



Save a copy of the notebook to your Drive

Google Colab: Cloud Computing in the Browser



Download the notebook when you are done

Image Classification: A core computer vision task

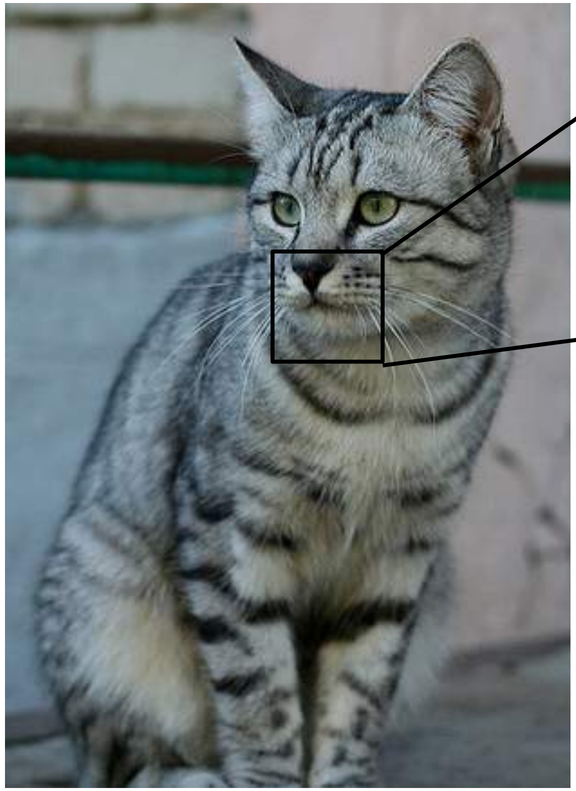
Input: image

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

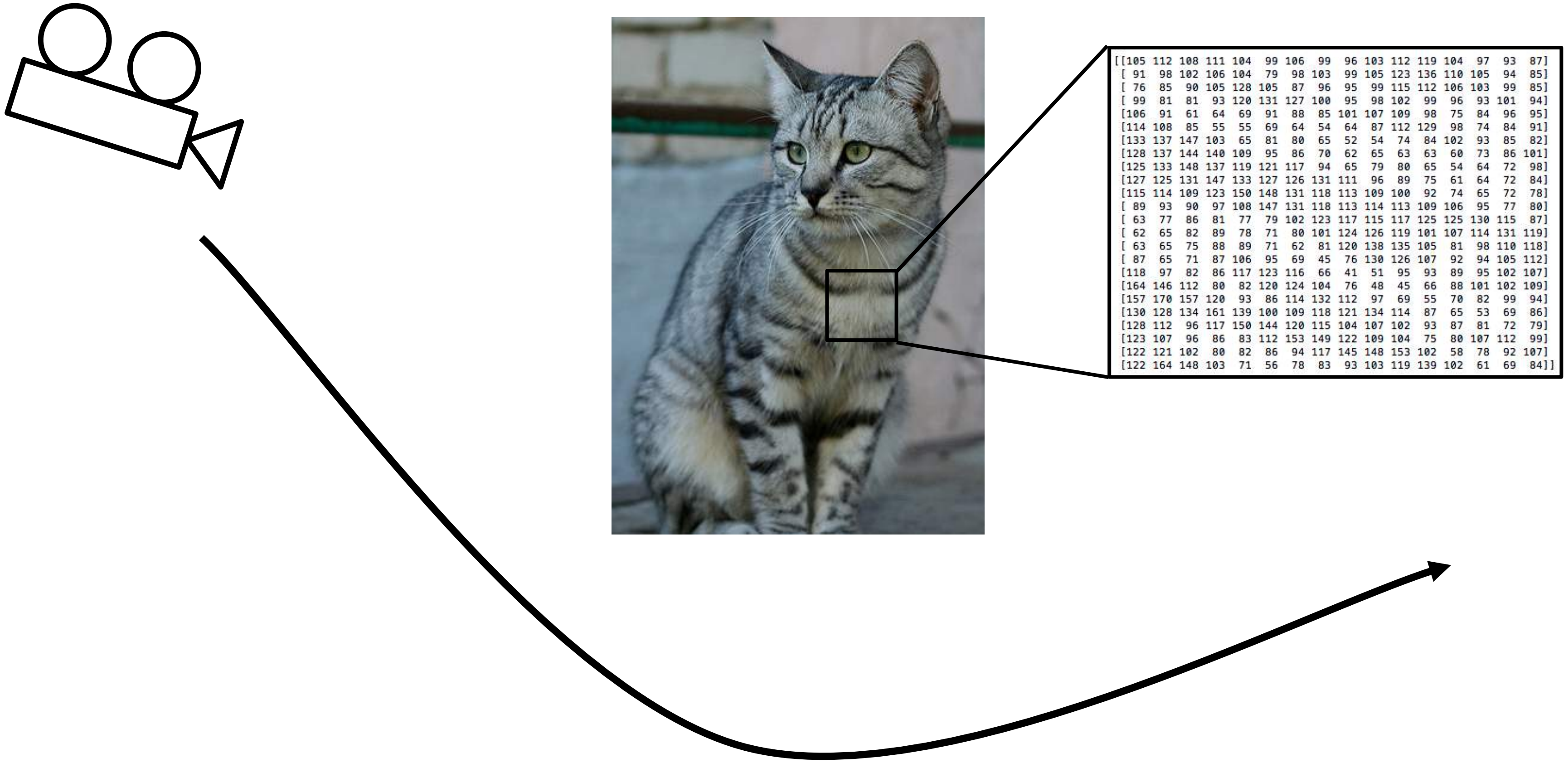
```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  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



All pixels change when the camera moves!

Challenges: Intraclass Variation

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

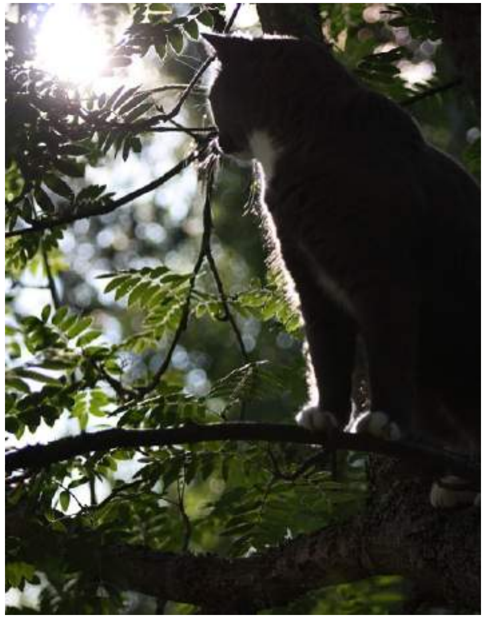
This image is CC0 1.0 public domain

Challenges: Illumination Changes

This image is CC0 1.0 public domain

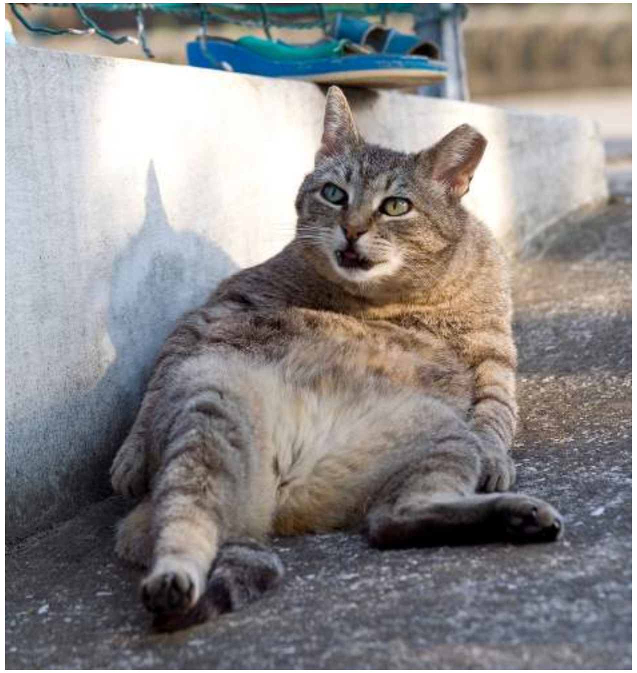
This image is CC0 1.0 public domain

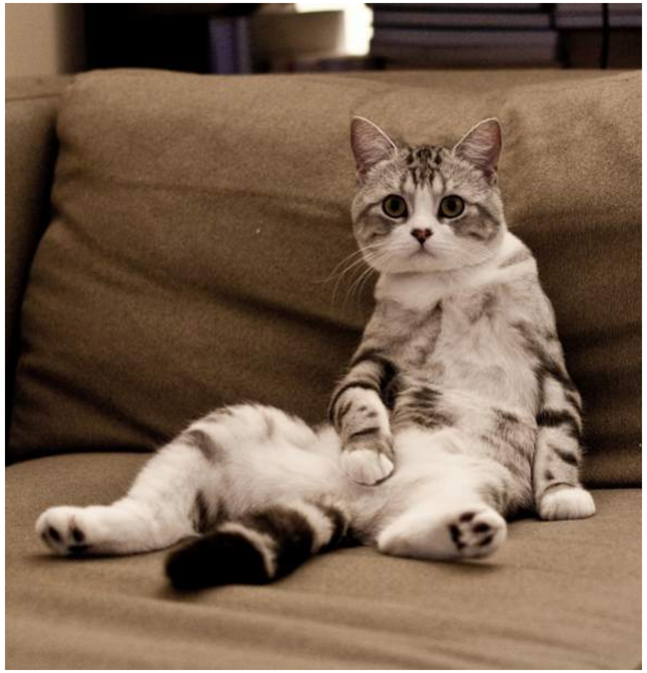
This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

Challenges: Deformation





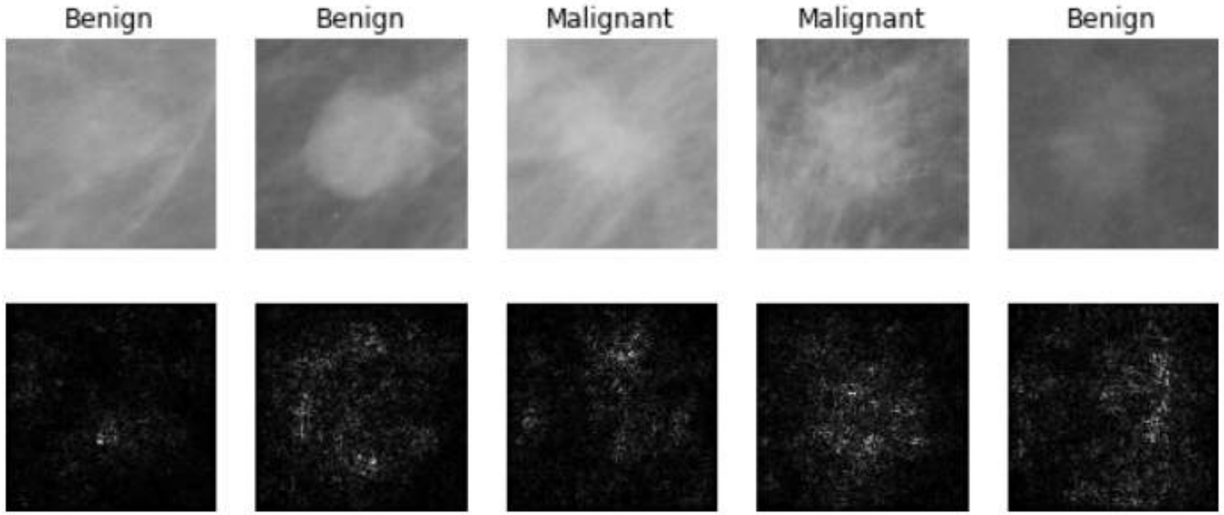
Challenges: Occlusion

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Image Classification: Very Useful!

Medical Imaging



Levy et al, 2016 Figure reproduced with permission

Galaxy Classification

Dieleman et al, 2014

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

Whale recognition

Kaggle Challenge

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Image Classification: Building Block for other tasks!

Example: Object Detection

→

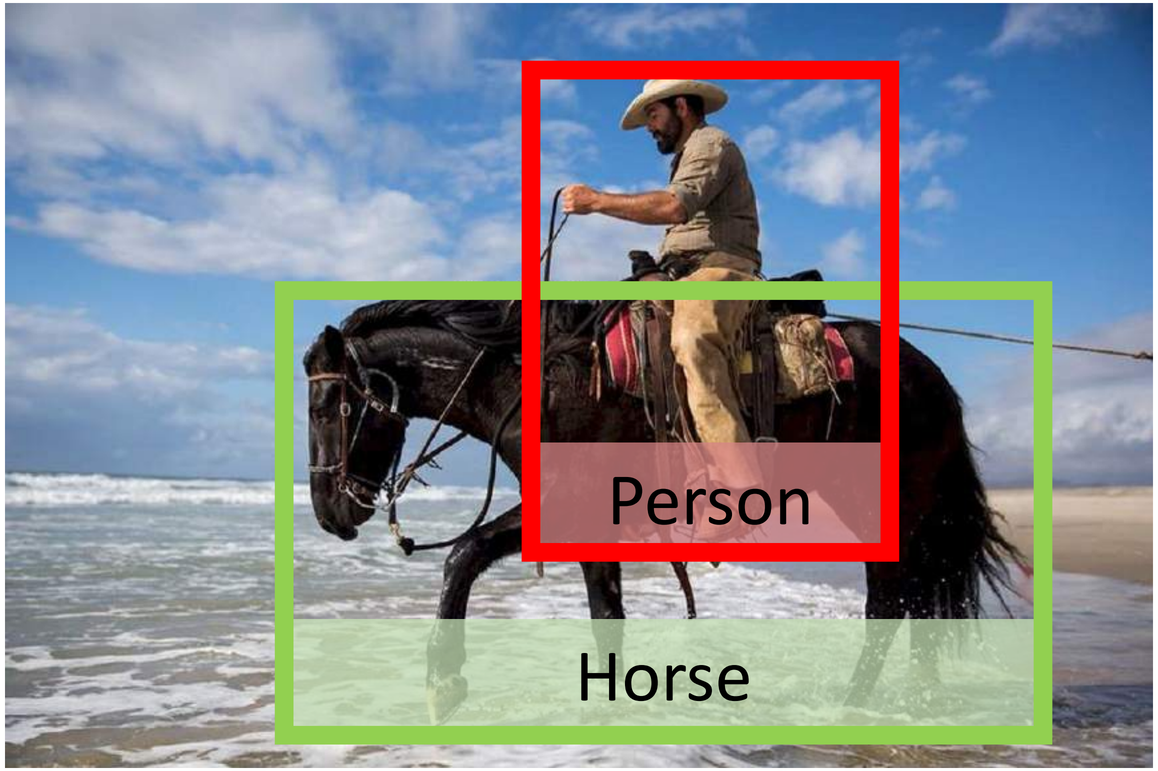


Image Classification: Building Block for other tasks!

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

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

Image Classification: Building Block for other tasks!

Example: Playing Go

This image is CC0 public domain

→

(1, 1)

(1, 2)

...

(1, 19)

...

(19, 19)

Where to play next?

An Image Classifier

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

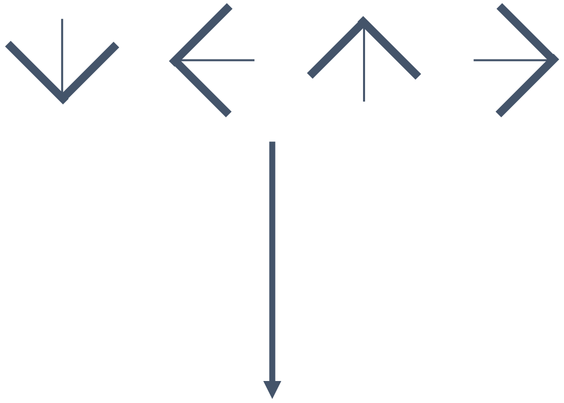
Unlike e.g. sorting a list of numbers,

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

airplane
automobile
bird
cat
deer

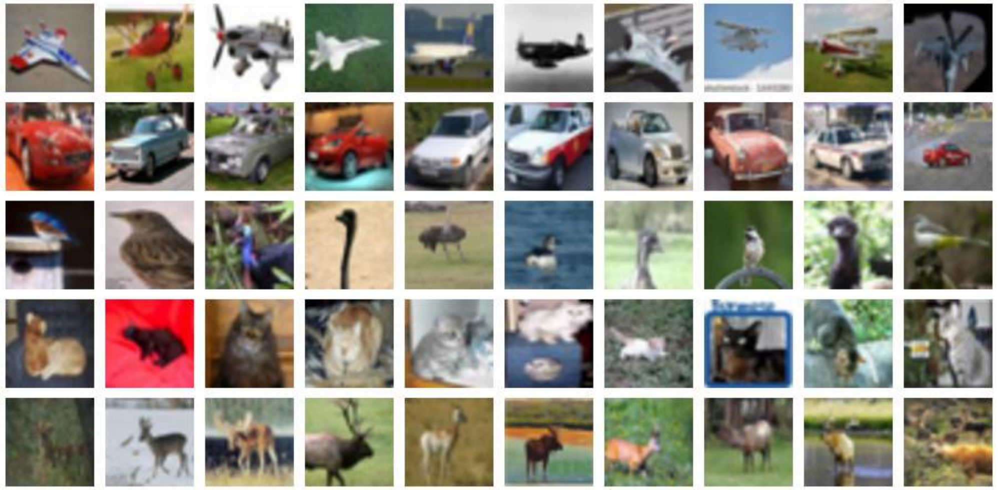
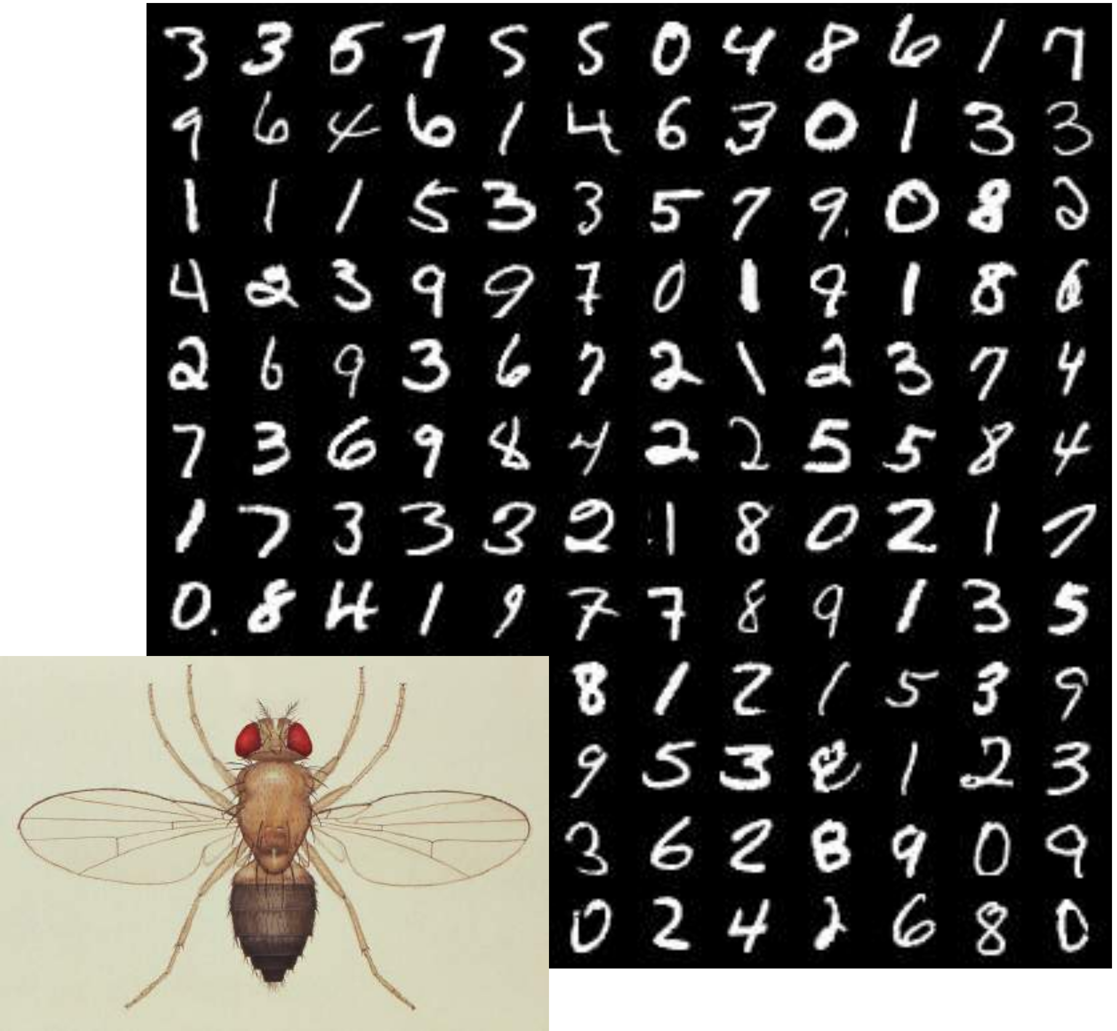


Image Classification Datasets: MNIST

10 classes: Digits 0 to 9
28x28 grayscale images
50k training images
10k test images

Image Classification Datasets: MNIST



10 classes: Digits 0 to 9
28x28 grayscale images
50k training images
10k test images

"Drosophila of computer vision"

Results from MNIST often do not hold on more complex datasets!

Image Classification Datasets: **CIFAR10**

10 classes
50k training images (5k per class)
10k testing images (1k per class)
32x32 RGB images

We will use this dataset for homework assignments

Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

Image Classification Datasets: **CIFAR100**

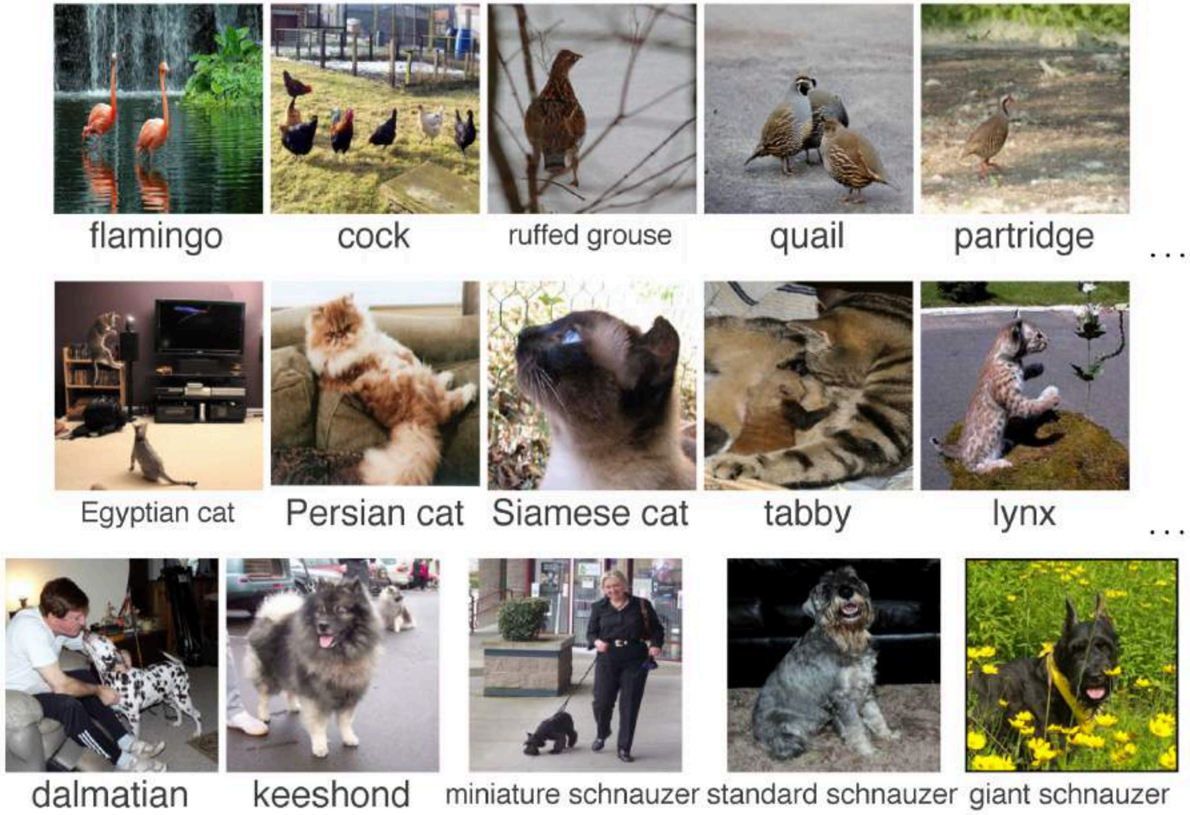
100 classes
50k training images (500 per class)
10k testing images (100 per class)
32x32 RGB images

20 superclasses with 5 classes each:

Aquatic mammals: beaver, dolphin, otter, seal, whale
Trees: Maple, oak, palm, pine, willow

Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Image Classification Datasets: **ImageNet**



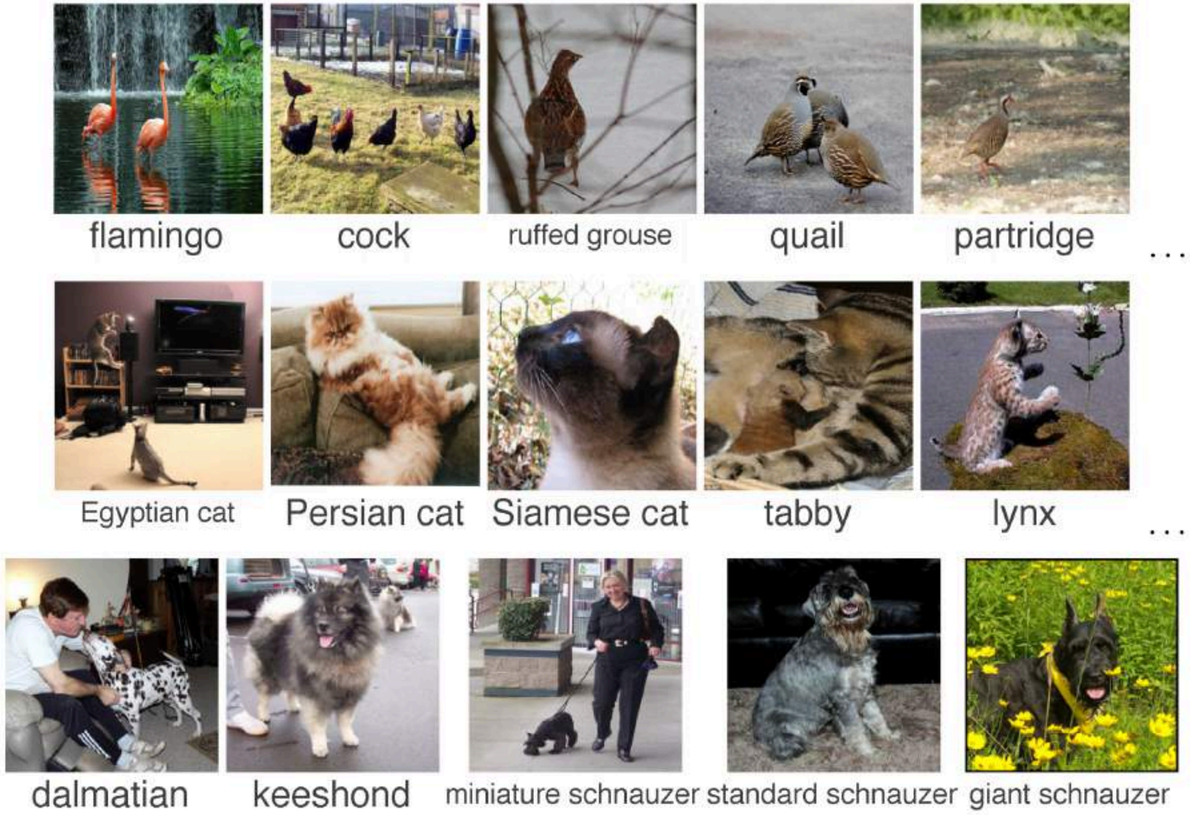
1000 classes

~1.3M training images (~1.3K per class)
50K validation images (50 per class)
100K test images (100 per class)

Performance metric: **Top 5 accuracy**
Algorithm predicts 5 labels for each image; one of them needs to be right

Deng et al, "ImageNet: A Large-Scale Hierarchical Image Database", CVPR 2009
Russakovsky et al, "ImageNet Large Scale Visual Recognition Challenge", IJCV 2015

Image Classification Datasets: **ImageNet**



1000 classes

~1.3M training images (~1.3K per class)
50K validation images (50 per class)
100K test images (100 per class)
test labels are secret!

Images have variable size, but often resized to **256x256** for training

There is also a 22k category version of ImageNet, but less commonly used

Deng et al, “ImageNet: A Large-Scale Hierarchical Image Database”, CVPR 2009
Russakovsky et al, “ImageNet Large Scale Visual Recognition Challenge”, IJCV 2015

Image Classification Datasets: **MIT Places**

365 classes of different scene types

~8M training images
18.25K val images (50 per class)
328.5K test images (900 per class)

Images have variable size, often resize to **256x256** for training

Zhou et al, "Places: A 10 million Image Database for Scene Recognition", TPAMI 2017

Classification Datasets: **Number of Training Pixels**

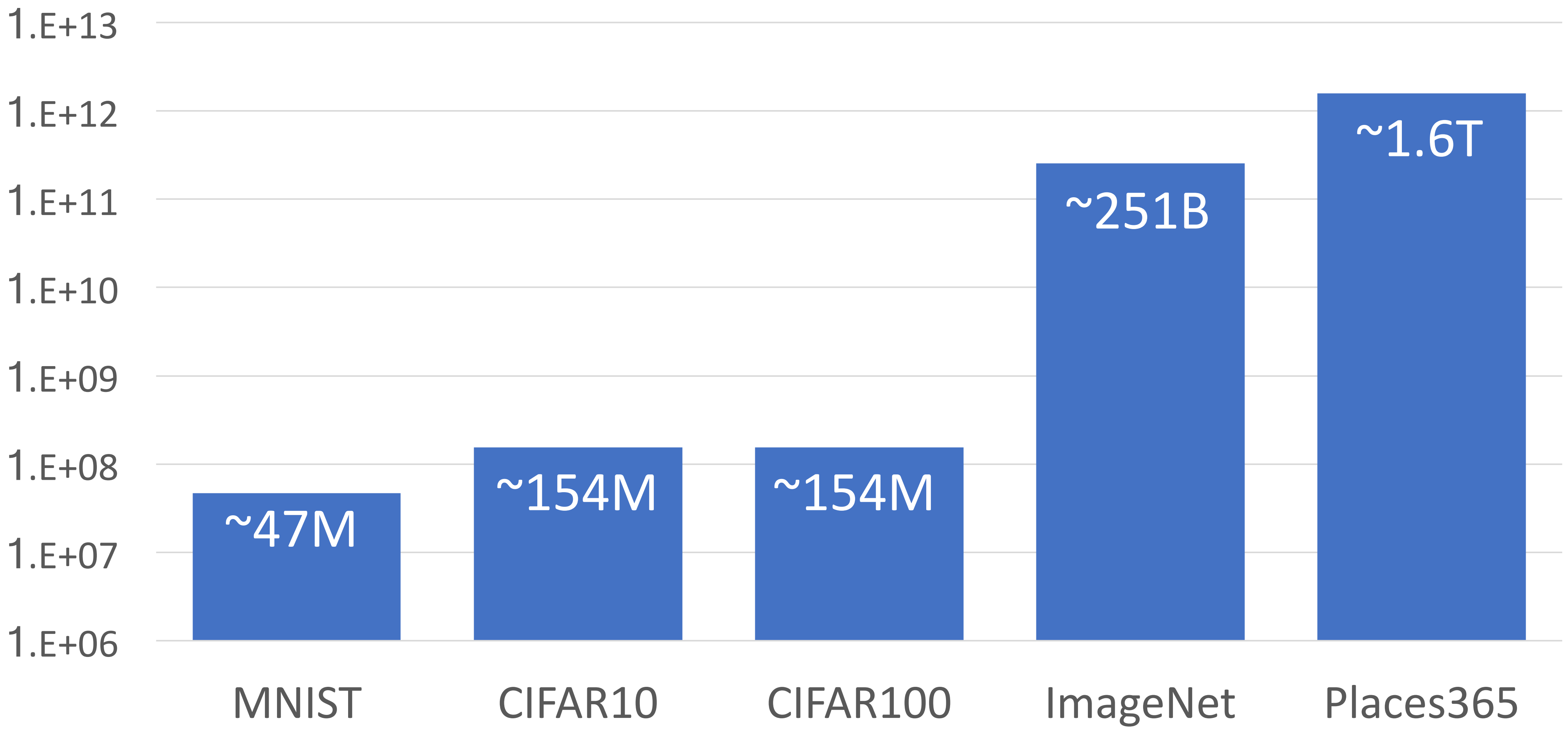


Image Classification Datasets: **Omniglot**

1623 categories: characters from 50 different alphabets

20 images per category

Meant to test **few shot learning**

Lake et al, “Human-level concept learning through probabilistic program induction”, Science, 2015

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

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

$\xrightarrow{\text{add}}$ 456

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

```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

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!

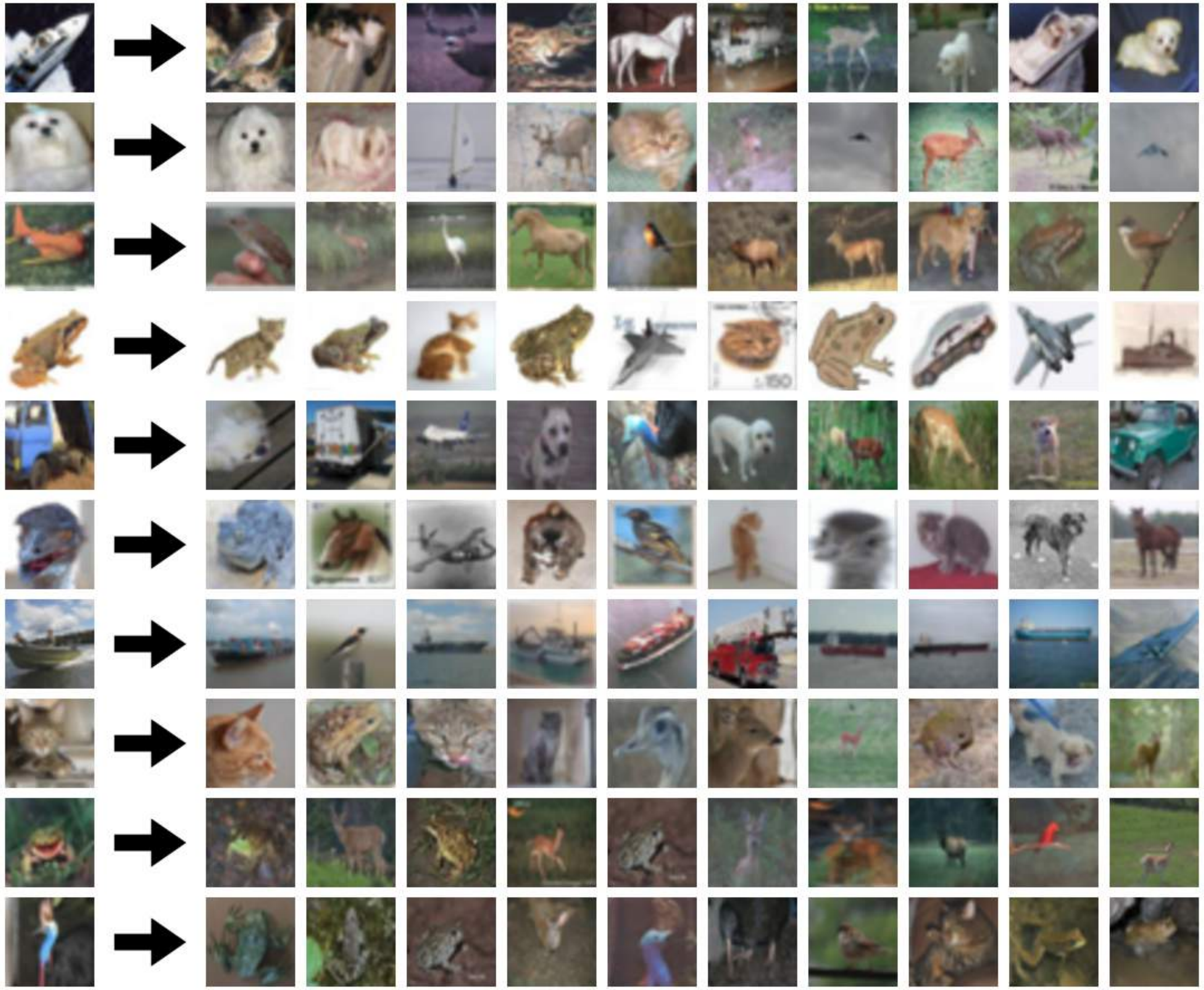
```python
import numpy as np

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

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

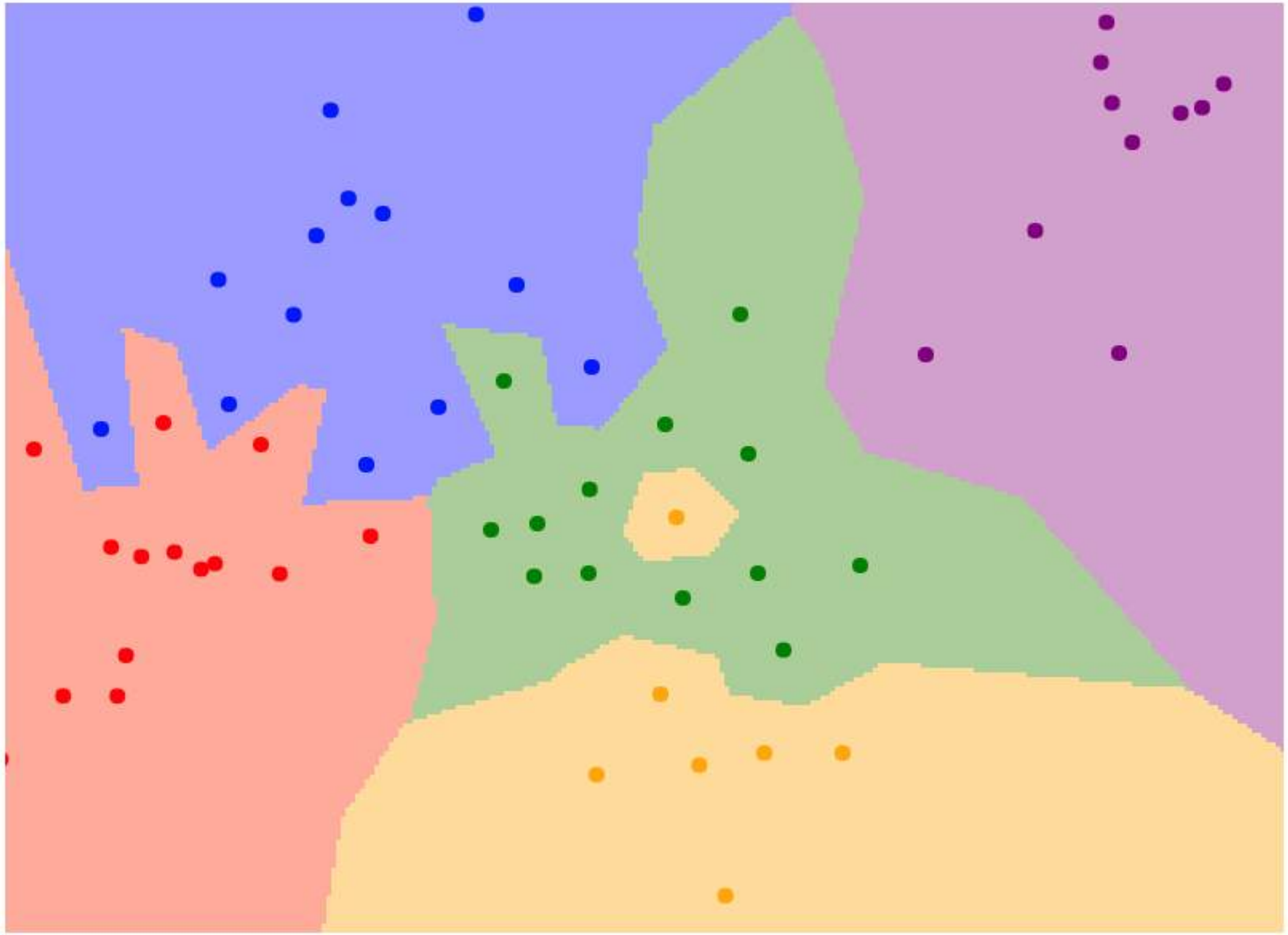
https://github.com/facebookresearch/faiss

What does this look like?



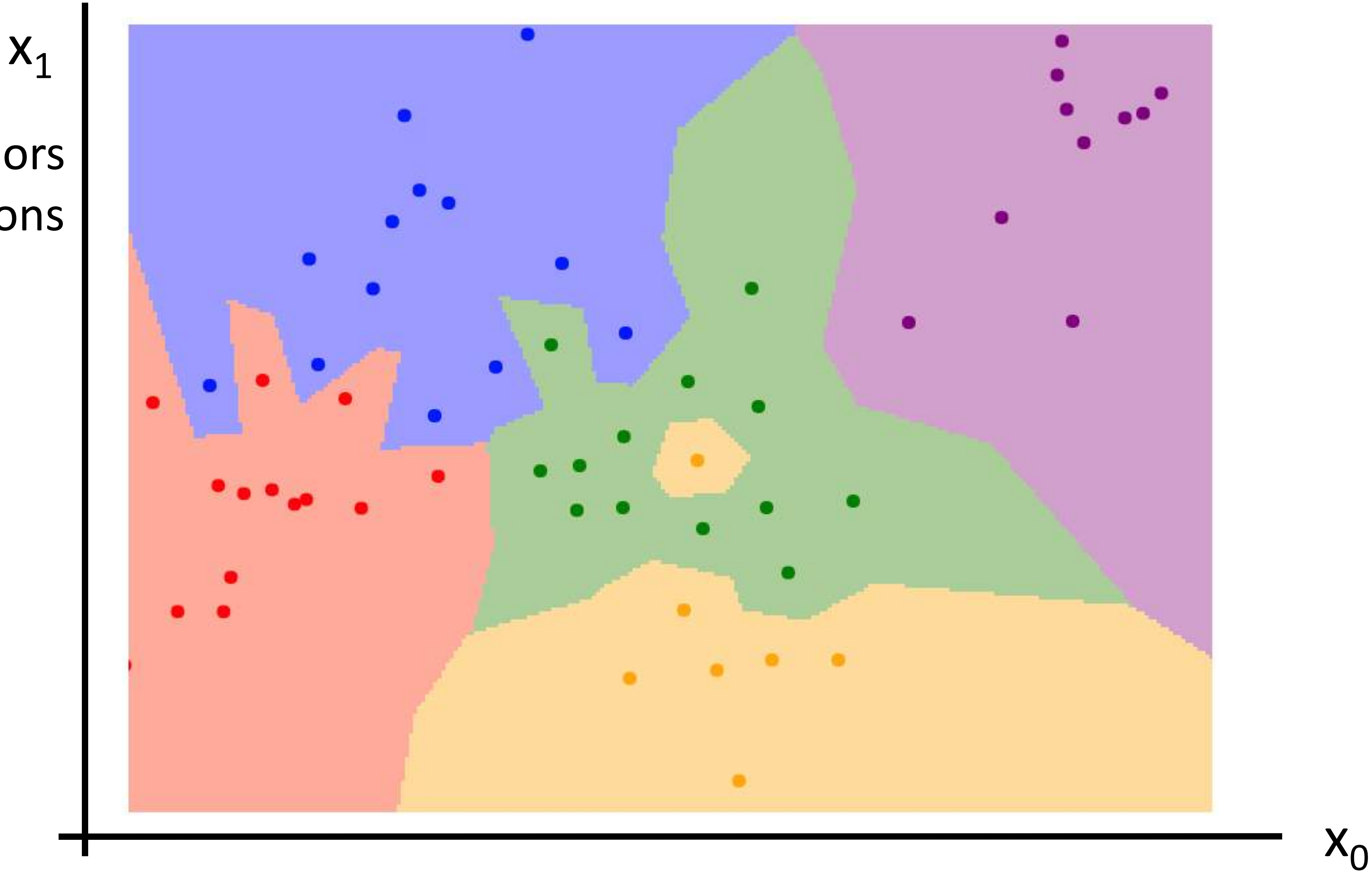
What does this look like?

Nearest Neighbor **Decision Boundaries**



Nearest Neighbor **Decision Boundaries**

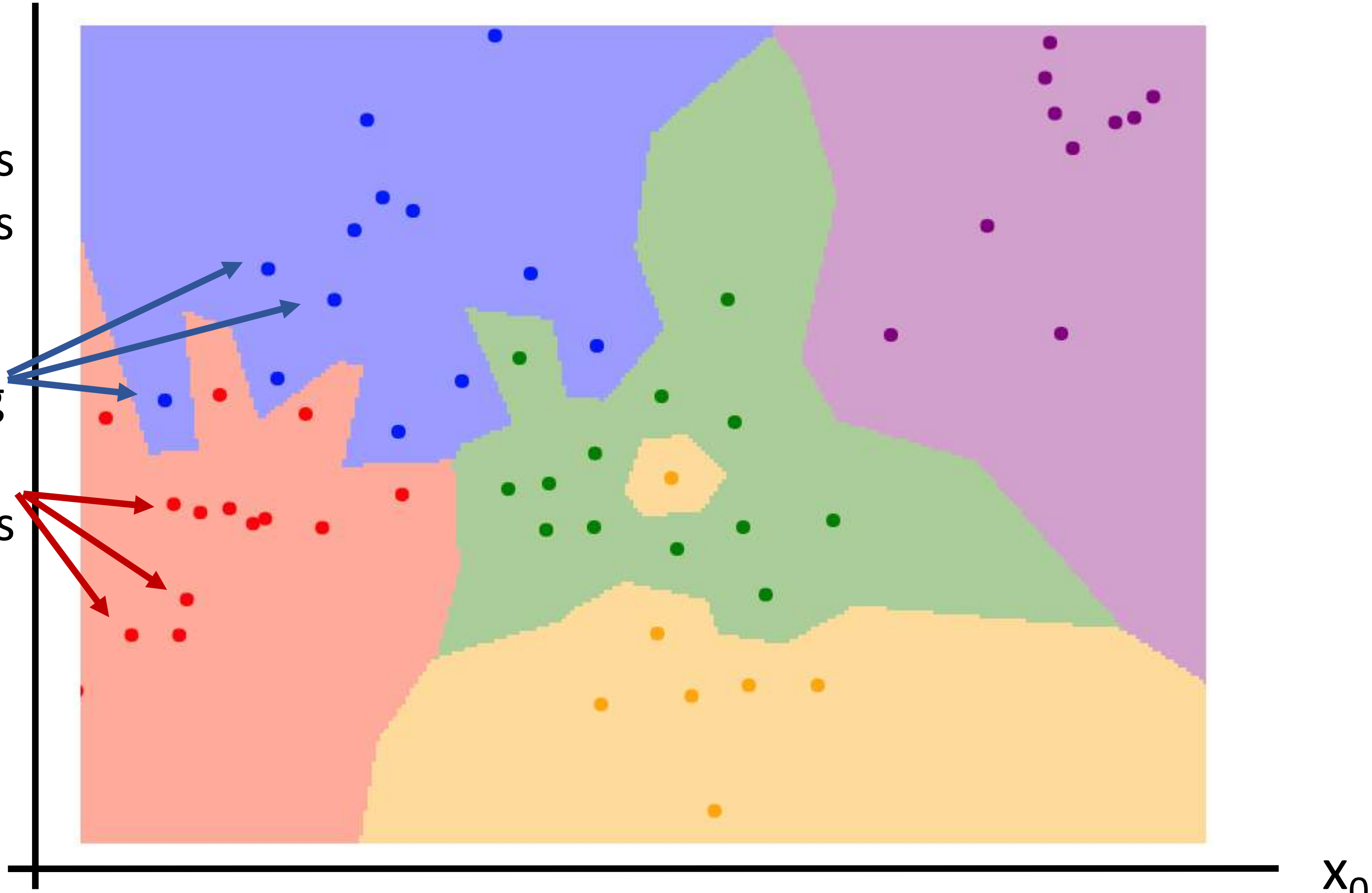
Nearest neighbors in two dimensions



Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

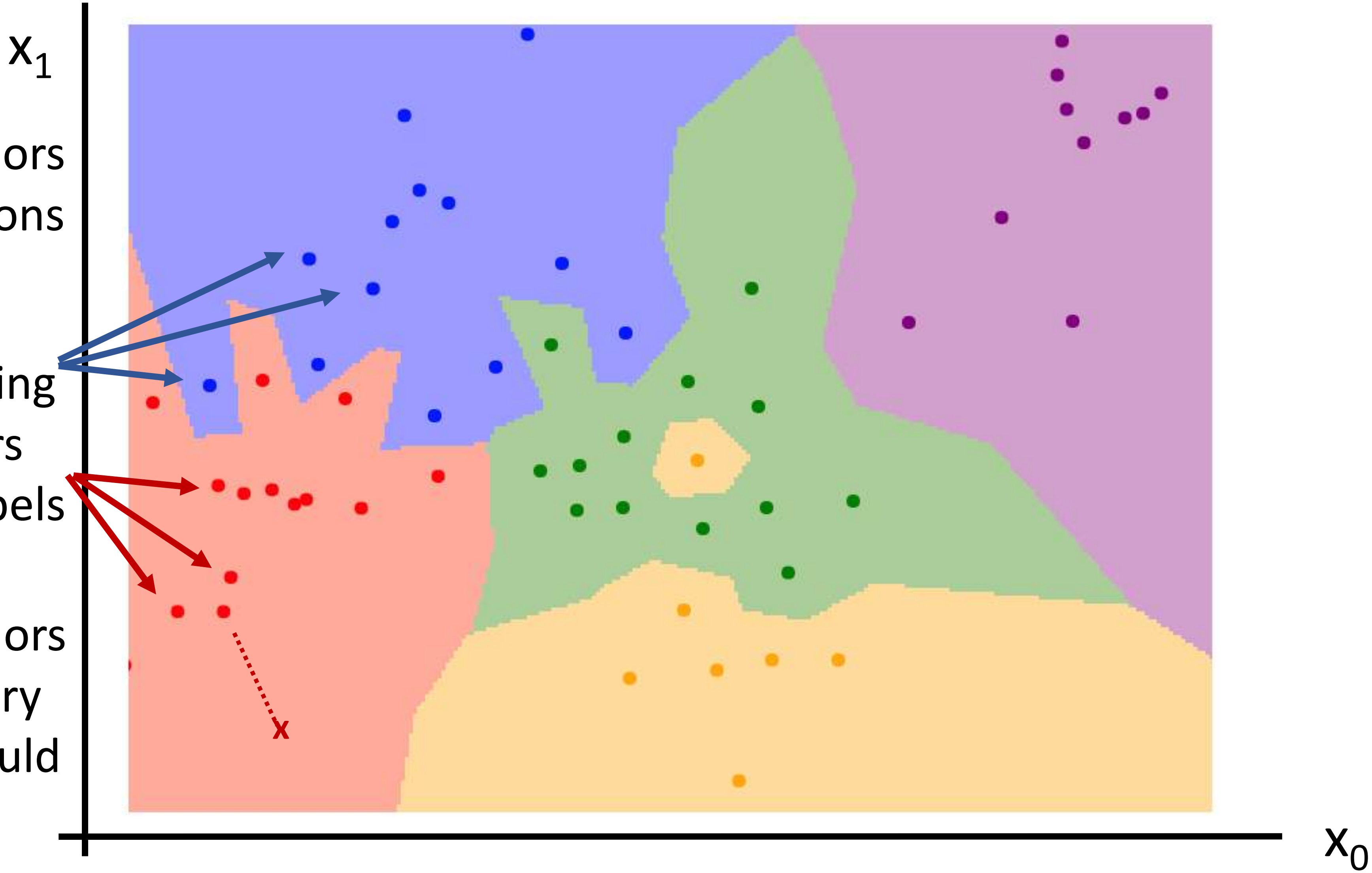


Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned

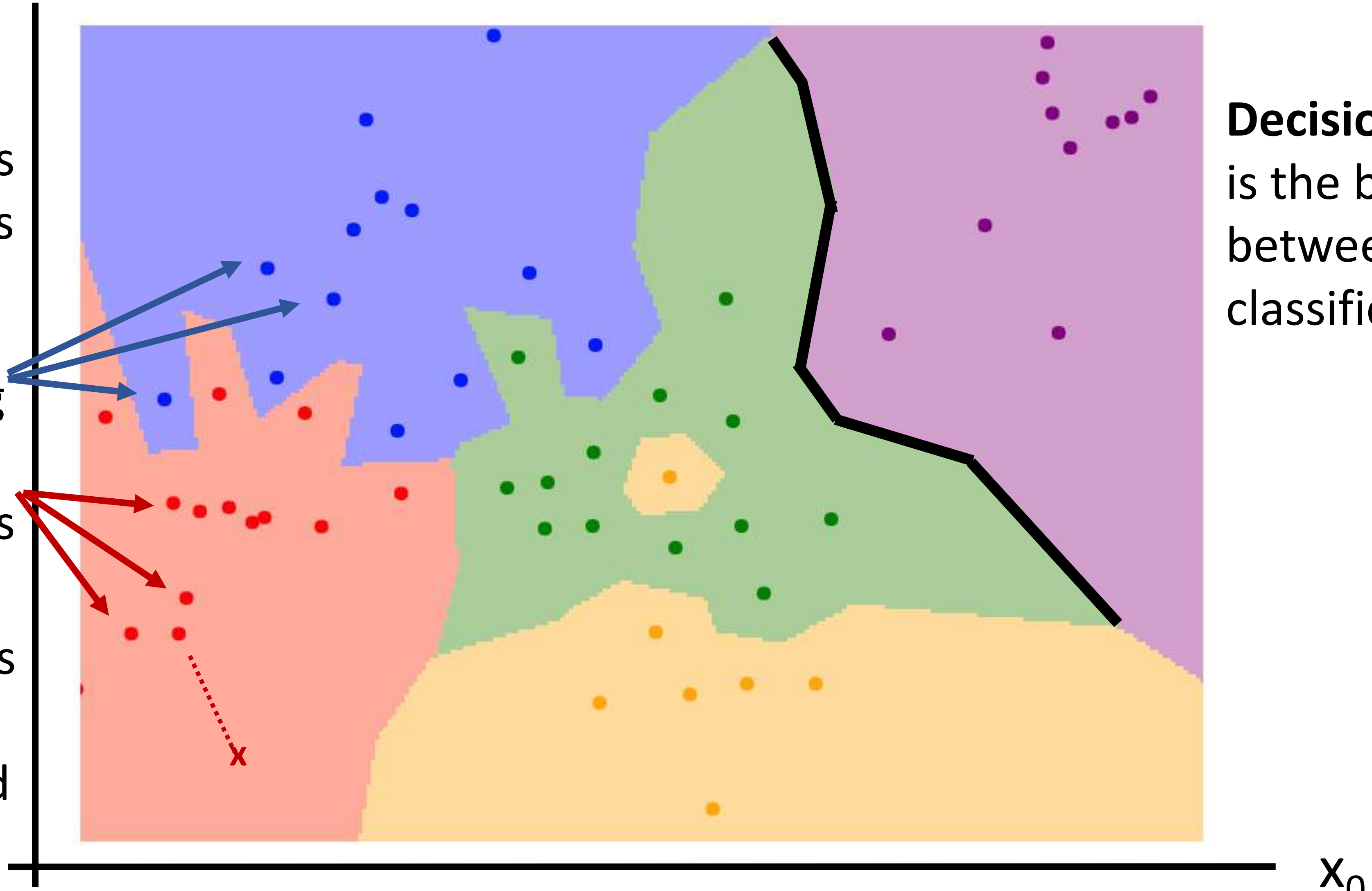


Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



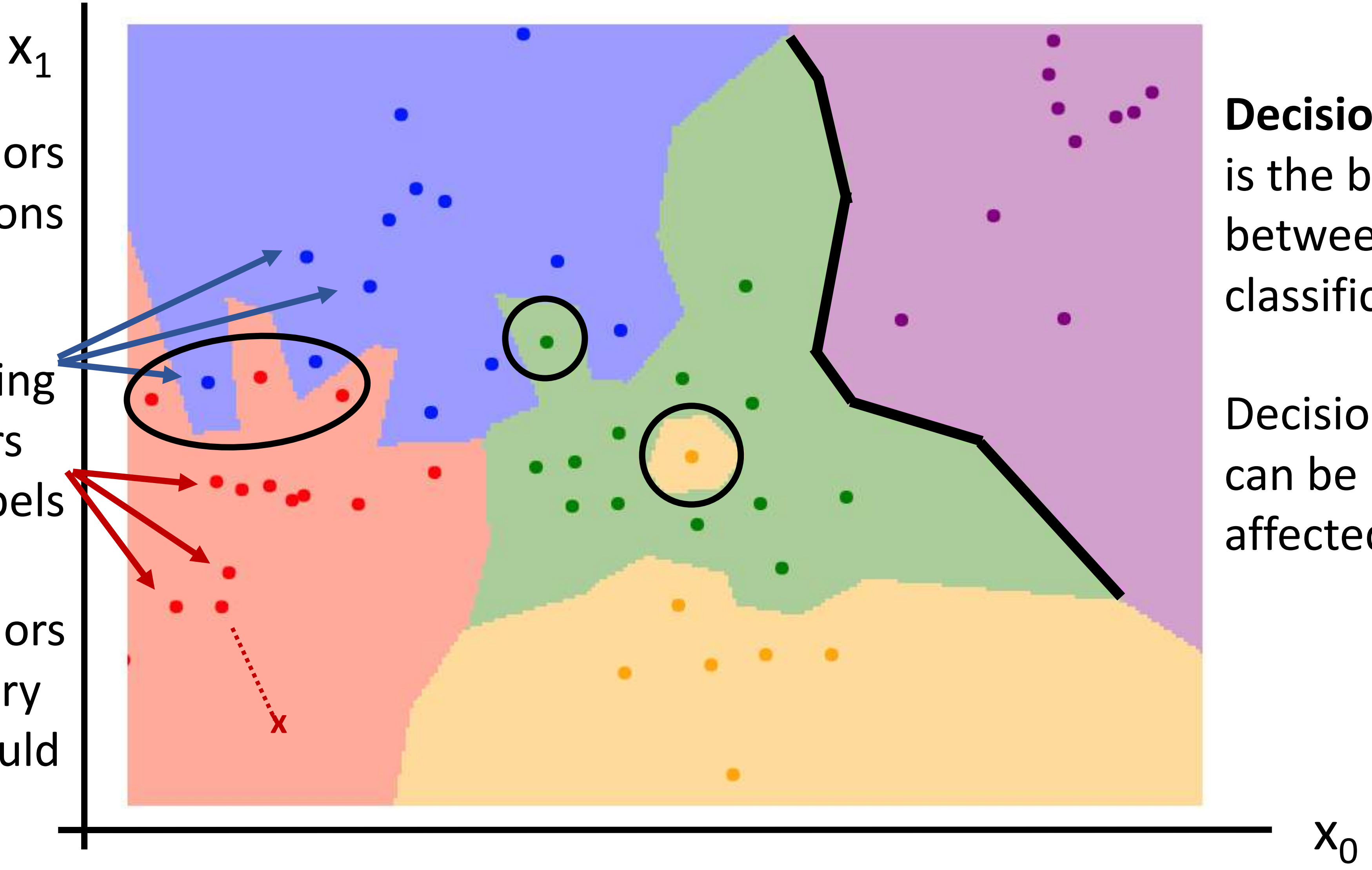
Decision boundary is the boundary between two classification regions

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

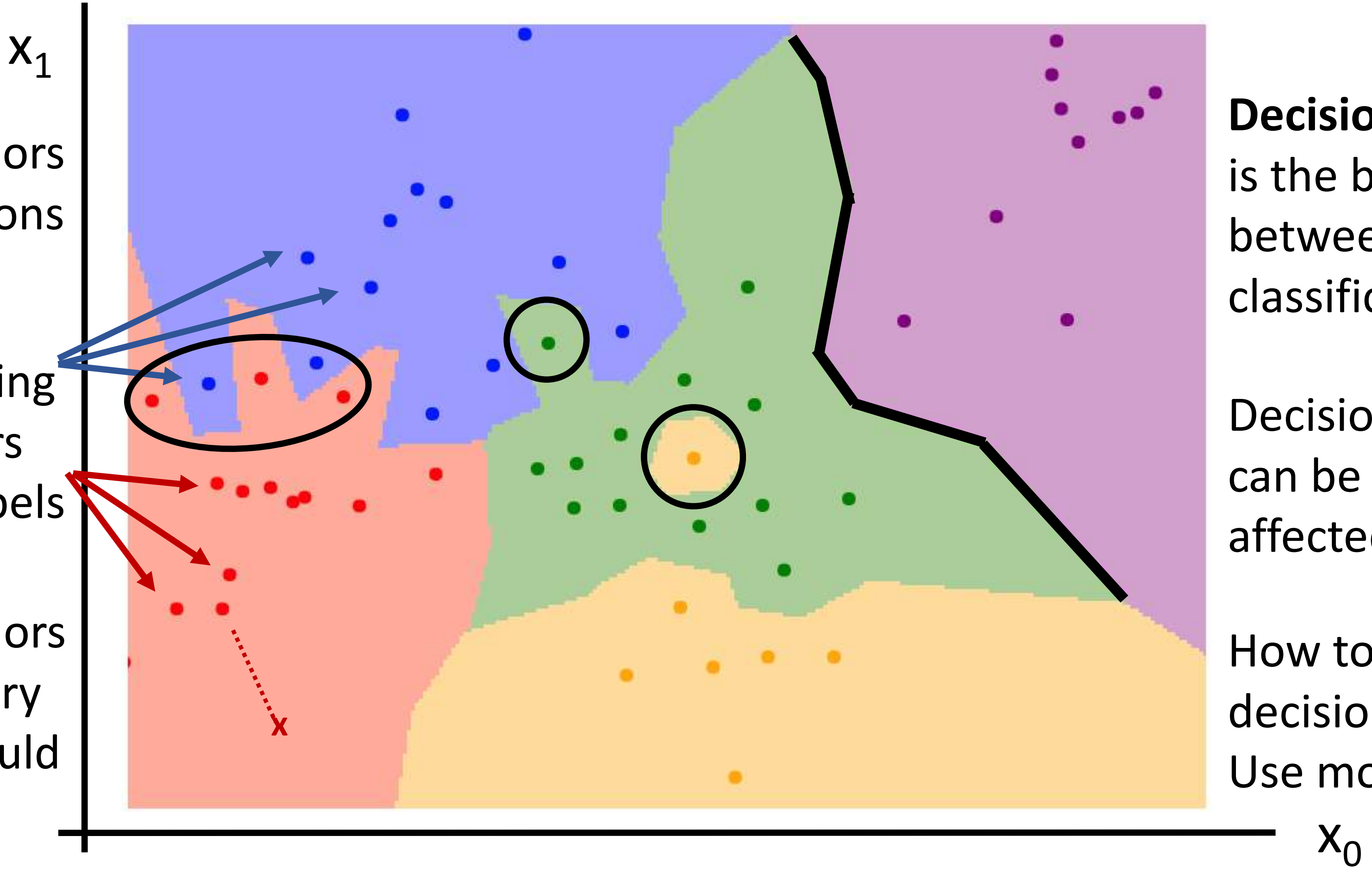
Decision boundaries can be noisy; affected by outliers

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

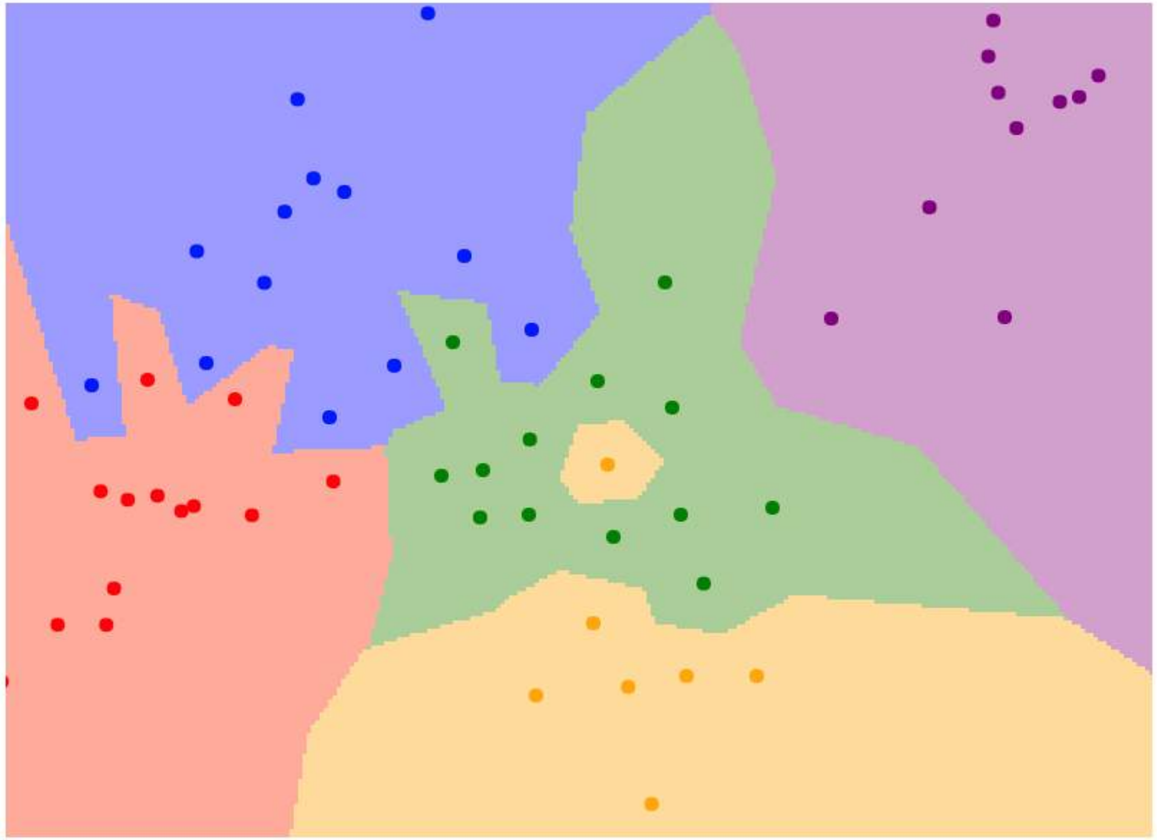
Decision boundaries can be noisy; affected by outliers

How to smooth out decision boundaries? Use more neighbors!

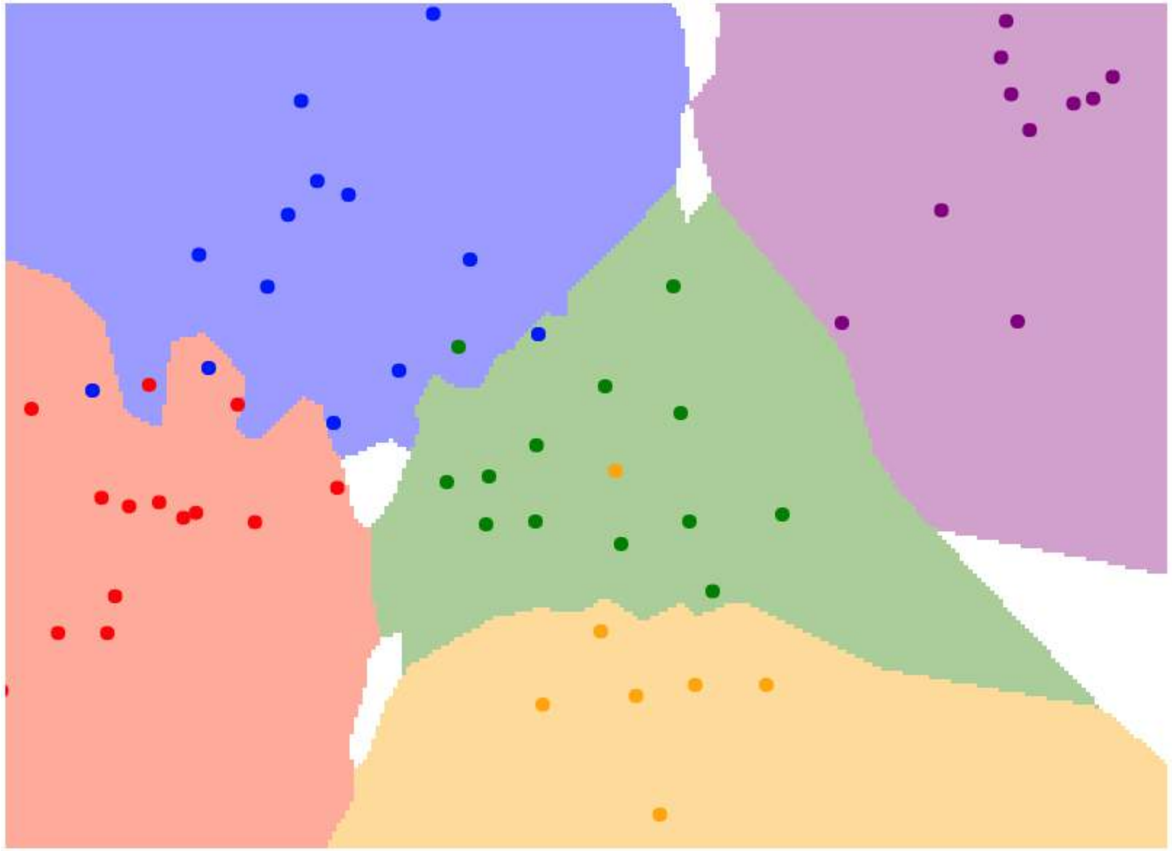
K-Nearest Neighbors

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

K = 1

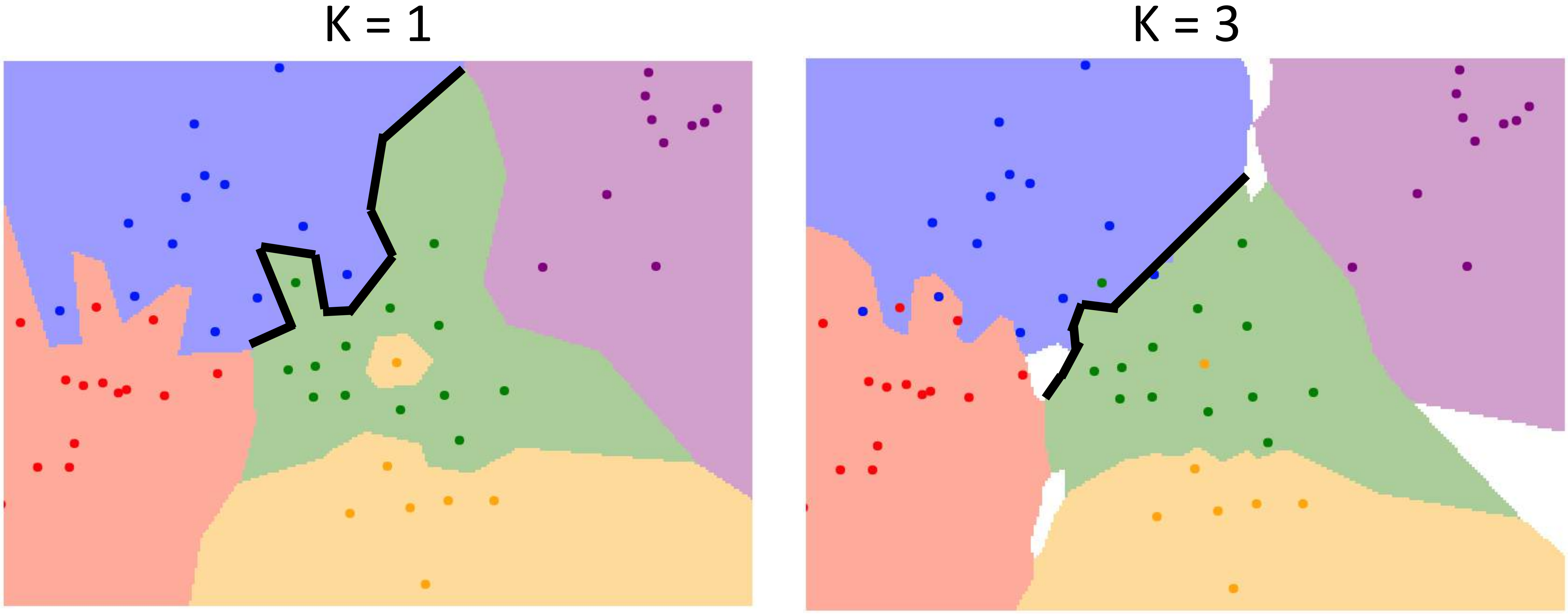


K = 3



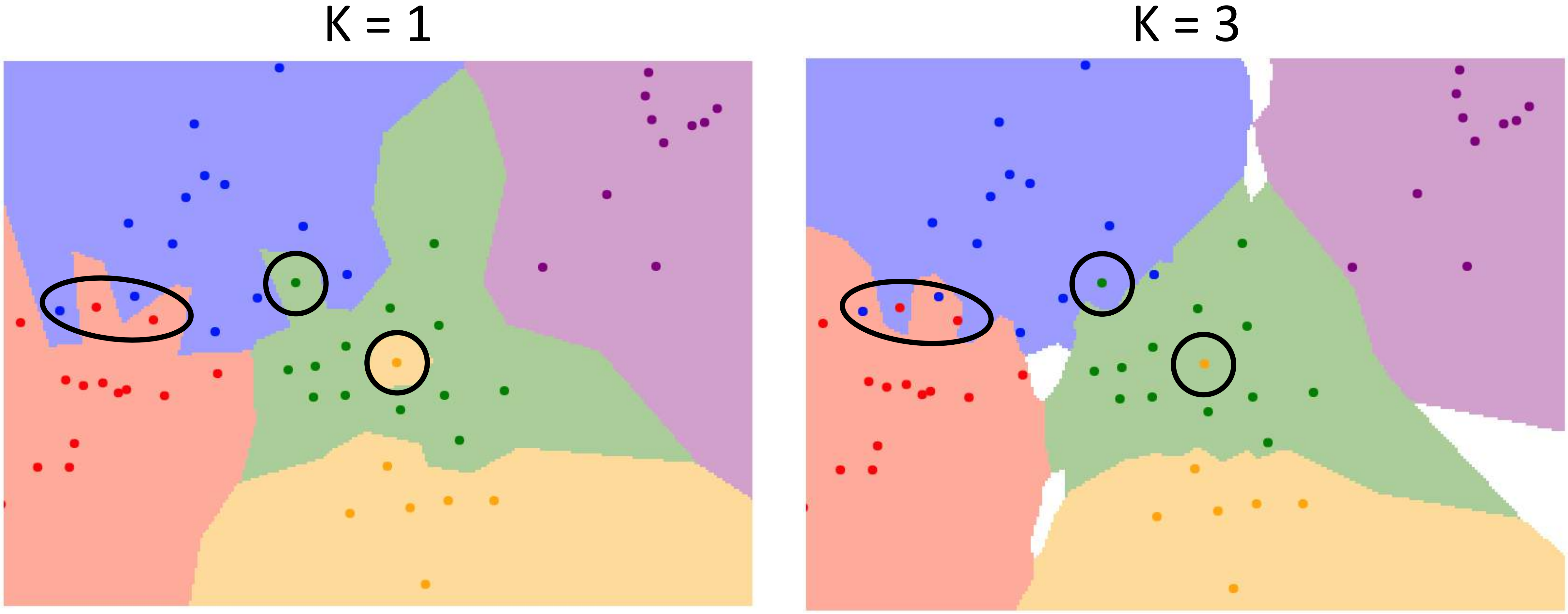
K-Nearest Neighbors

Using more neighbors helps smooth out rough decision boundaries



K-Nearest Neighbors

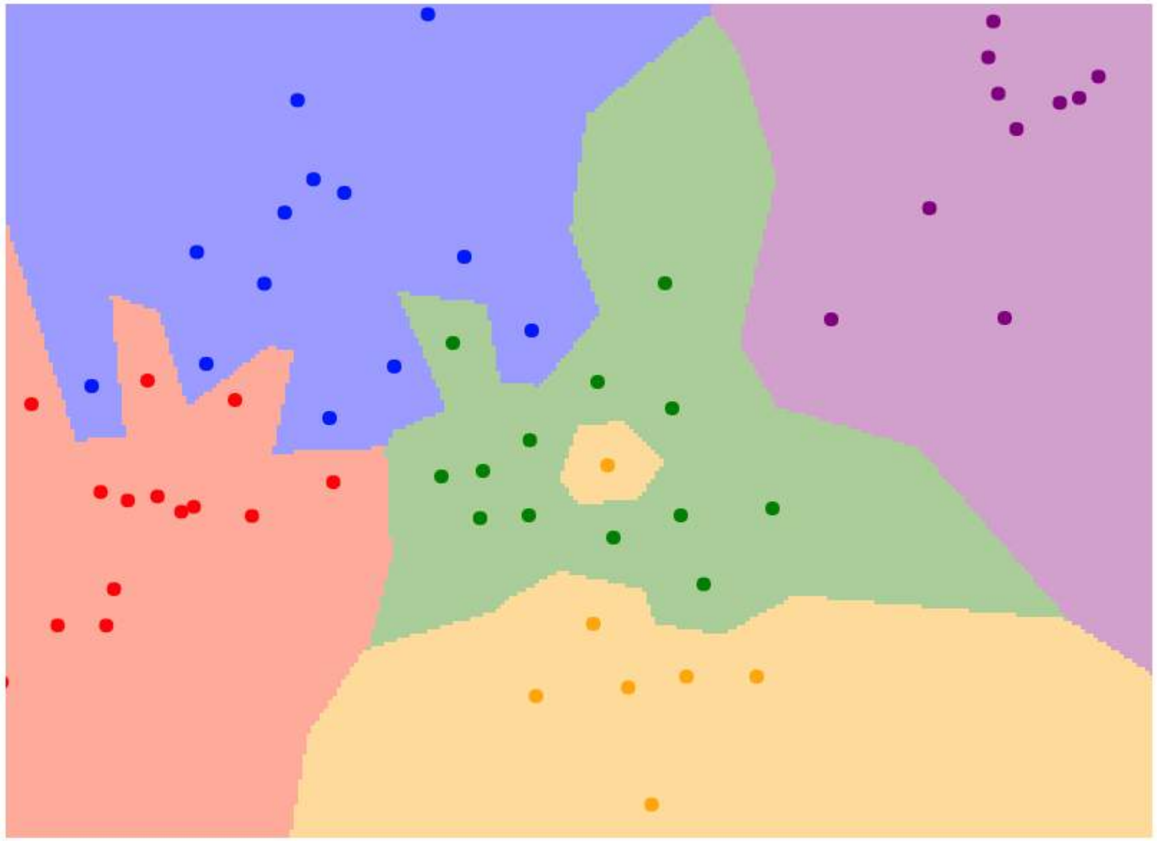
Using more neighbors helps reduce the effect of outliers



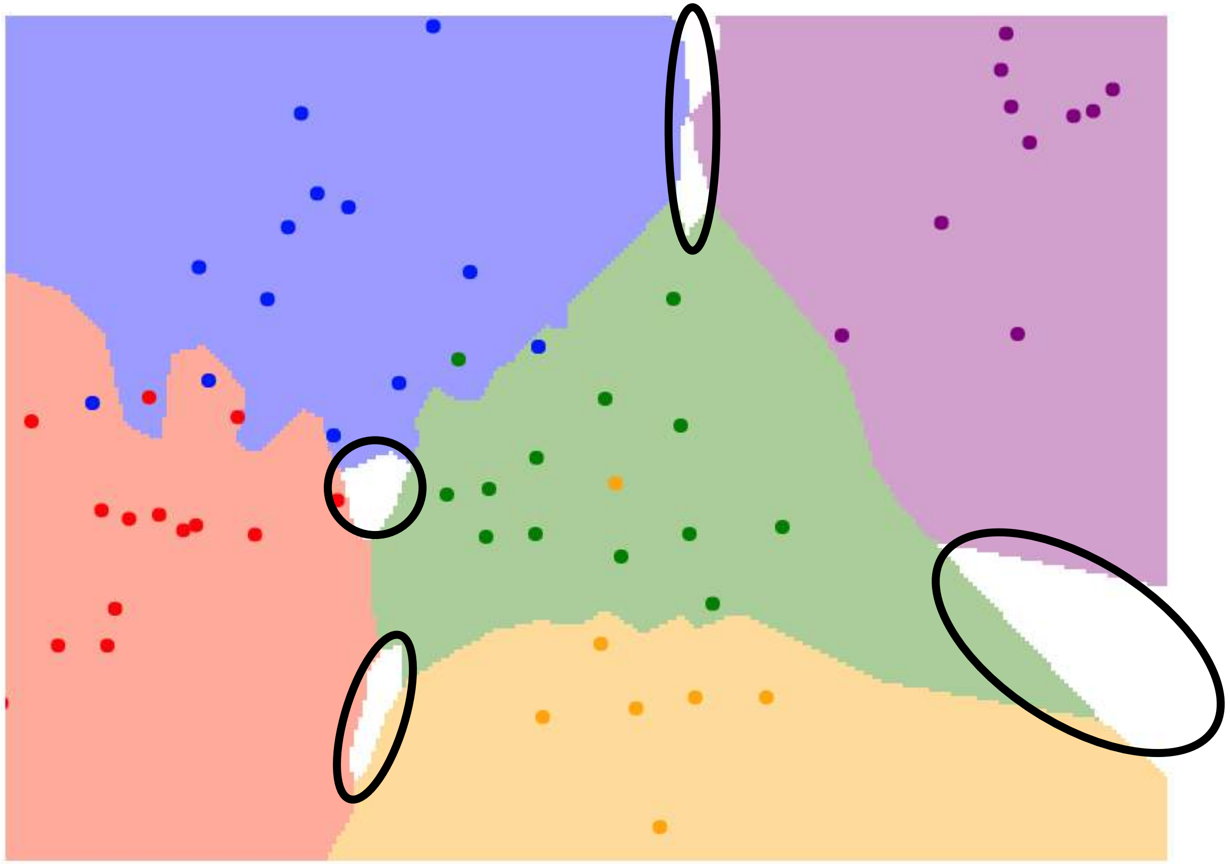
K-Nearest Neighbors

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

K = 1



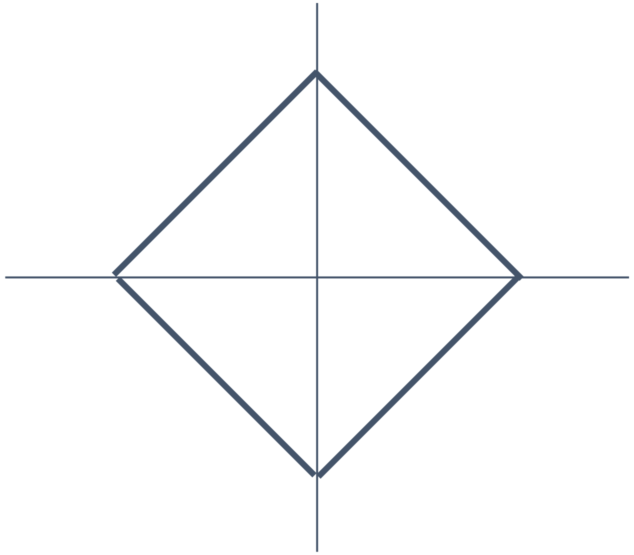
K = 3



K-Nearest Neighbors: **Distance Metric**

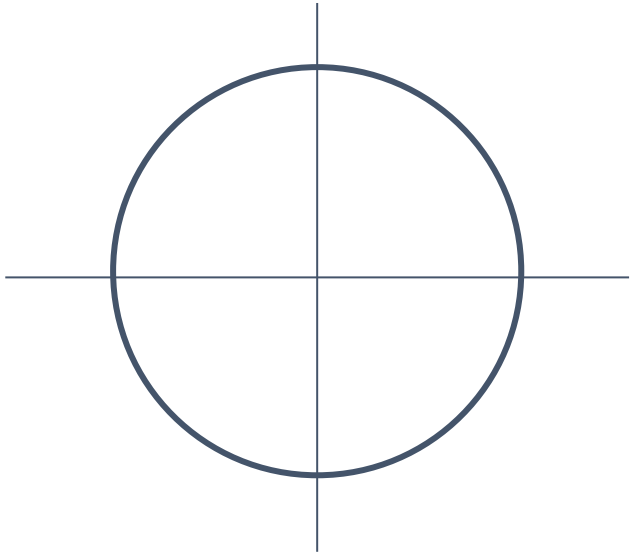
L1 (Manhattan) distance

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



L2 (Euclidean) distance

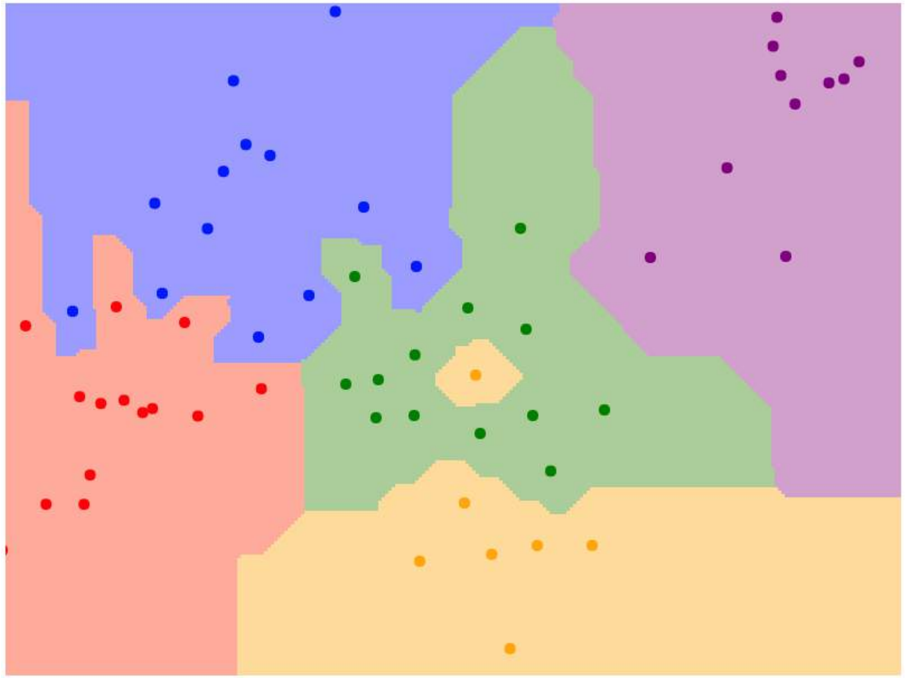
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



K-Nearest Neighbors: **Distance Metric**

L1 (Manhattan) distance

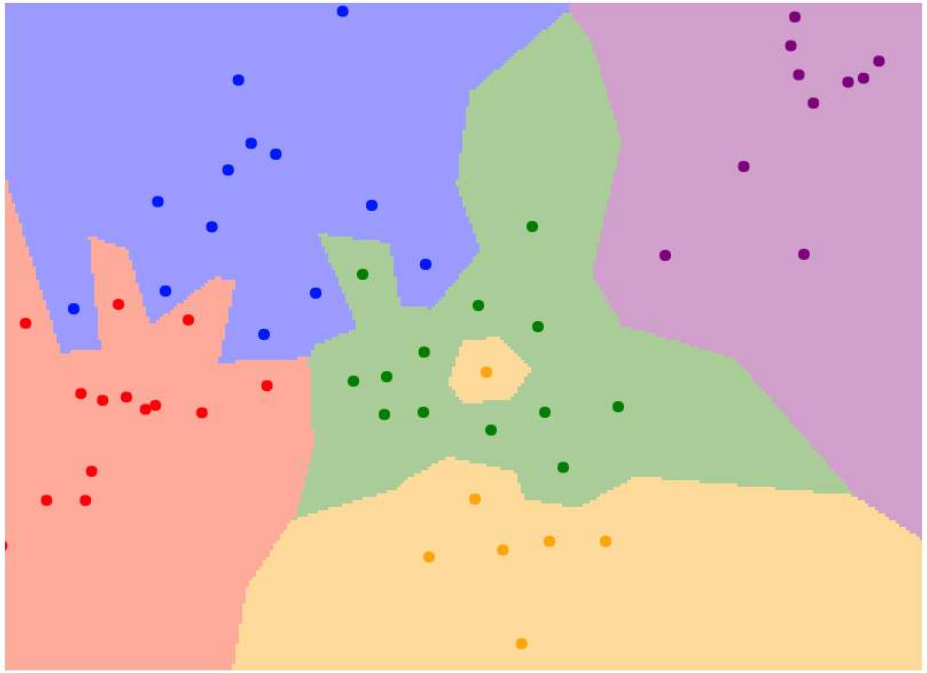
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

L2 (Euclidean) distance

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



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: **Distance Metric**

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

Example: Compare research papers using tf-idf similarity

Mesh R-CNN 1906.02739v1 pdf

Georgia Gkioxari, Jitendra Malik, Justin Johnson show similar discuss

6/6/2019 cs.CV

Rapid advances in 2D perception have led to systems that accurately detect objects in real-world images. However, these systems make predictions in 2D, ignoring the 3D structure of the world. Concurrently, advances in 3D shape prediction have mostly focused on synthetic benchmarks and isolated objects. We unify advances in these two areas. We propose a system that detects objects in real-world images and produces a triangle mesh giving the full 3D shape of each detected object. Our system, called Mesh R-CNN, augments Mask R-CNN with a mesh prediction branch that outputs meshes with varying topological structure by first predicting coarse voxel representations which are converted to meshes and refined with a graph convolution network operating over the mesh's vertices and edges. We validate our mesh prediction branch on ShapeNet, where we outperform prior work on single-image shape prediction. We then deploy our full Mesh R-CNN system on Pix3D, where we jointly detect objects and predict their 3D shapes.

http://www.arxiv-sanity.com/search?q=mesh+r-cnn

K-Nearest Neighbors: **Distance Metric**

Most similar papers:

Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era 1906.06543v2 pdf
Xian-Feng Han, Hamid Laga, Mohammed Bennamoun
6/18/2019 (v1: 6/15/2019) cs.CV | cs.CG | cs.GR | cs.LG
show similar | discuss

3D reconstruction is a longstanding ill-posed problem, which has been explored for decades by the computer vision, computer graphics, and machine learning communities. Since 2015, image-based 3D reconstruction using convolutional neural networks (CNN) has attracted increasing interest and demonstrated an impressive performance. Given this new era of rapid evolution, this article provides a comprehensive survey of the recent developments in this field. We focus on the works which use deep learning techniques to estimate the 3D shape of generic objects either from a single or multiple RGB images. We organize the literature based on the shape representations, the network architectures, and the training mechanisms they use. While this survey is intended for methods which reconstruct generic objects, we also review some of the recent works which focus on specific object classes such as human body shapes and faces. We provide an analysis and comparison of the performance of some key papers, summarize some of the open problems in this field, and discuss promising directions for future research.

Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images 1804.01654v2 pdf
Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, Yu-Gang Jiang
8/3/2018 (v1: 4/5/2018) cs.CV
show similar | discuss

We propose an end-to-end deep learning architecture that produces a 3D shape in triangular mesh from a single color image. Limited by the nature of deep neural network, previous methods usually represent a 3D shape in volume or point cloud, and it is non-trivial to convert them to the more ready-to-use mesh model. Unlike the existing methods, our network represents 3D mesh in a graph-based convolutional neural network and produces correct geometry by progressively deforming an ellipsoid, leveraging perceptual features extracted from the input image. We adopt a coarse-to-fine strategy to make the whole deformation procedure stable, and define various of mesh related losses to capture properties of different levels to guarantee visually appealing and physically accurate 3D geometry. Extensive experiments show that our method not only qualitatively produces mesh model with better details, but also achieves higher 3D shape estimation accuracy compared to the state-of-the-art.

Pixel2Mesh++: Multi-View 3D Mesh Generation via Deformation 1908.01491v2 pdf
Chao Wen, Yinda Zhang, Zhuwen Li, Yanwei Fu
8/16/2019 (v1: 8/5/2019) cs.CV
Accepted by ICCV 2019
show similar | discuss

We study the problem of shape generation in 3D mesh representation from a few color images with known camera poses. While many previous works learn to hallucinate the shape directly from priors, we resort to further improving the shape quality by leveraging cross-view information with a graph convolutional network. Instead of building a direct mapping function from images to 3D shape, our model learns to predict series of deformations to improve a coarse shape iteratively. Inspired by traditional multiple view geometry methods, our network samples nearby area around the initial mesh's vertex locations and reasons an optimal deformation using perceptual feature statistics built from multiple input images. Extensive experiments show that our model produces accurate 3D shape that are not only visually plausible from the input perspectives, but also well aligned to arbitrary viewpoints. With the help of physically driven architecture, our model also exhibits generalization capability across different semantic categories, number of input images, and quality of mesh initialization.

GEOMetrics: Exploiting Geometric Structure for Graph-Encoded Objects 1901.11461v1 pdf
Edward J. Smith, Scott Fujimoto, Adriana Romero, David Meger
1/31/2019 cs.CV
18 pages
show similar | discuss

Mesh models are a promising approach for encoding the structure of 3D objects. Current mesh reconstruction systems predict uniformly distributed vertex locations of a predetermined graph through a series of graph convolutions, leading to compromises with respect to performance or resolution. In this paper, we argue that the graph representation of geometric objects allows for additional structure, which should be leveraged for enhanced reconstruction. Thus, we propose a system which properly benefits from the advantages of the geometric structure of graph encoded objects by introducing (1) a graph convolutional update preserving vertex information; (2) an adaptive splitting heuristic allowing detail to emerge; and (3) a training objective operating both on the local surfaces defined by vertices as well as the global structure defined by the mesh. Our proposed method is evaluated on the task of 3D object reconstruction from images with the ShapeNet dataset, where we demonstrate state of the art performance, both visually and numerically, while having far smaller space requirements by generating adaptive meshes

http://www.arxiv-sanity.com/1906.02739v1

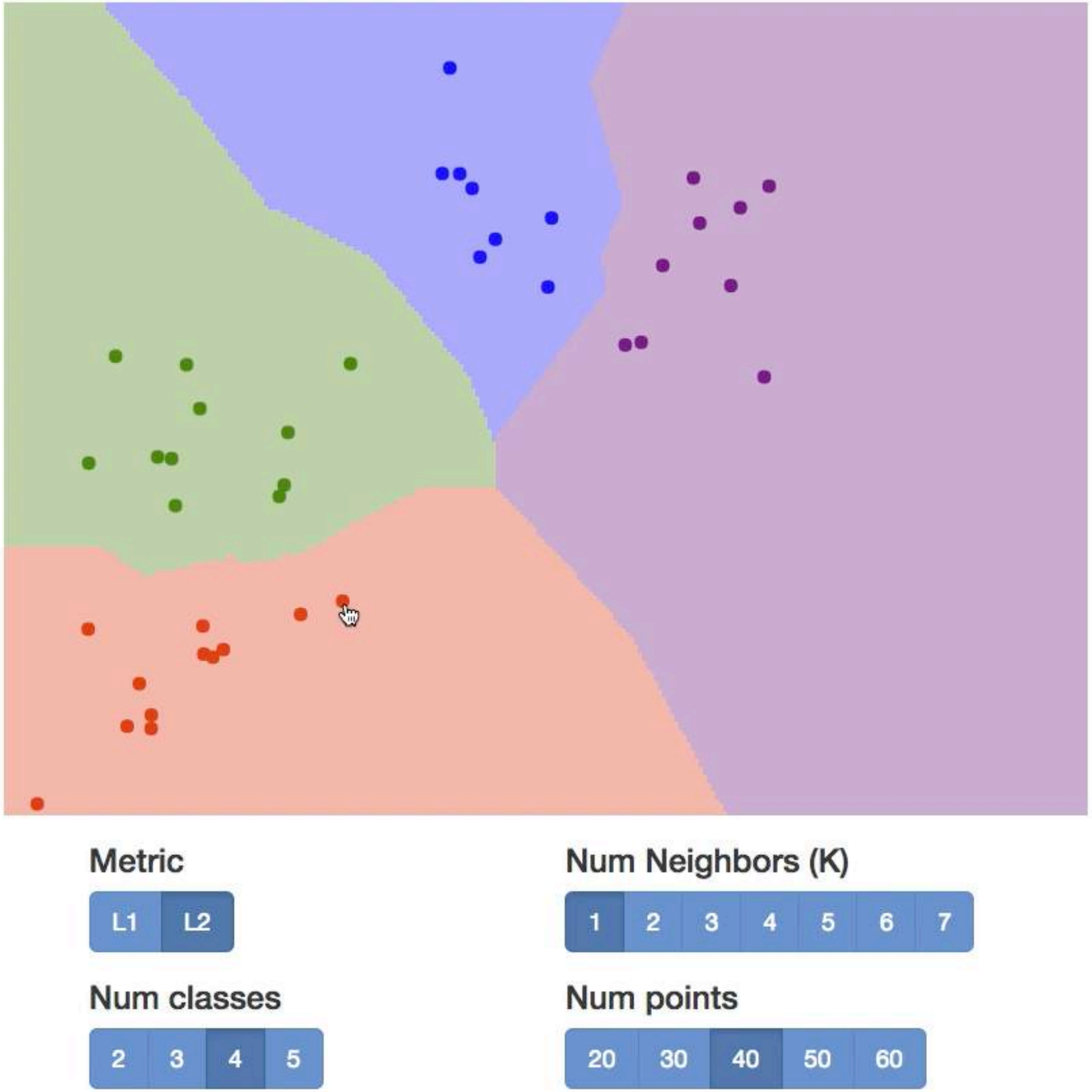
K-Nearest Neighbors: Web Demo

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

Your Dataset

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

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

train	test

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

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

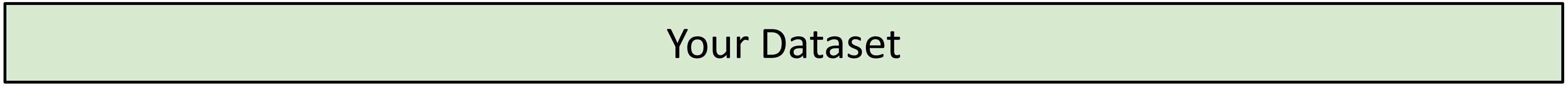
BAD: No idea how algorithm will perform on new data

train	test

Setting Hyperparameters

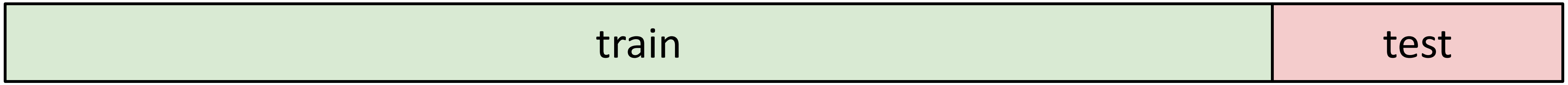
Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data



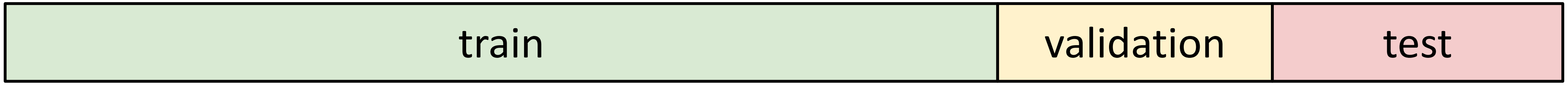
Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

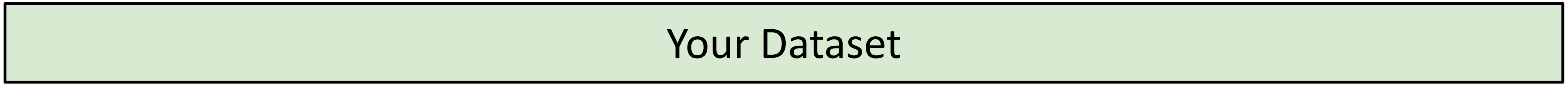


Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

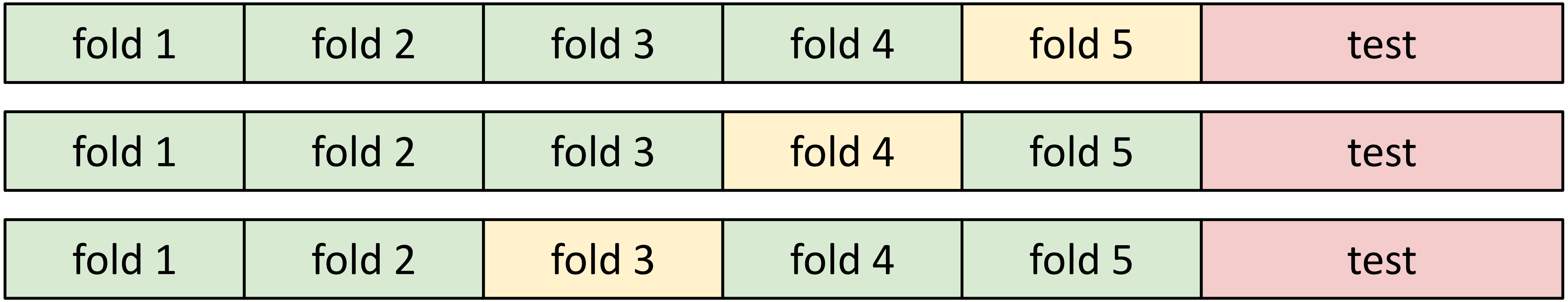
Better!



Setting Hyperparameters

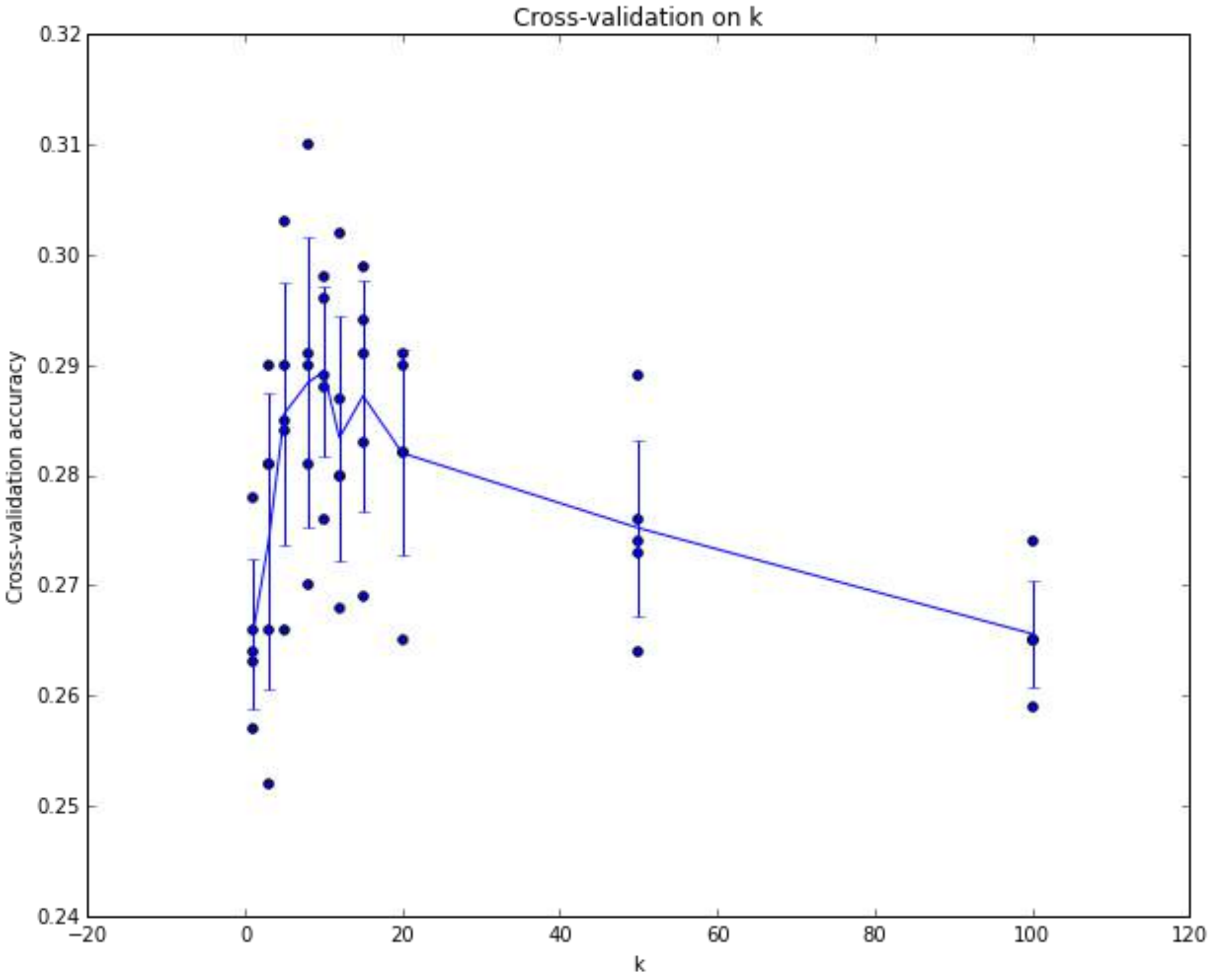


Idea #4: **Cross-Validation**: Split data into **folds**, try each fold as validation and average the results



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)

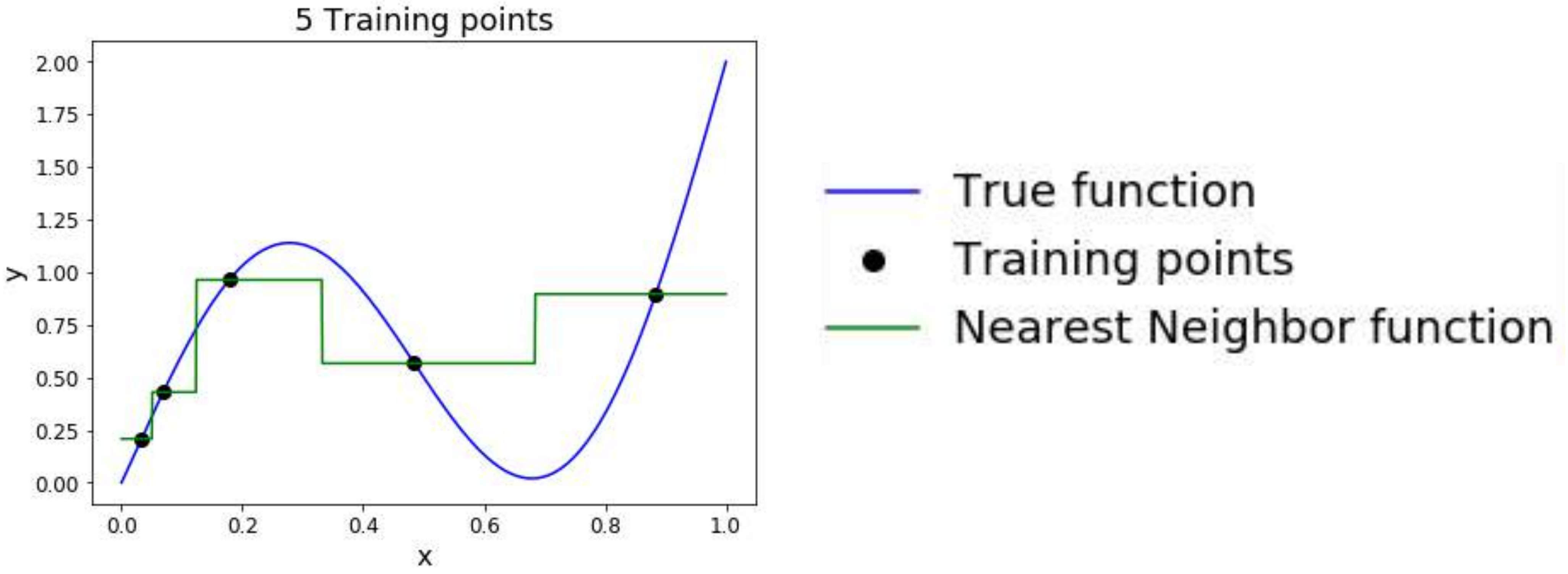
K-Nearest Neighbor: **Universal Approximation**

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!

(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

K-Nearest Neighbor: **Universal Approximation**

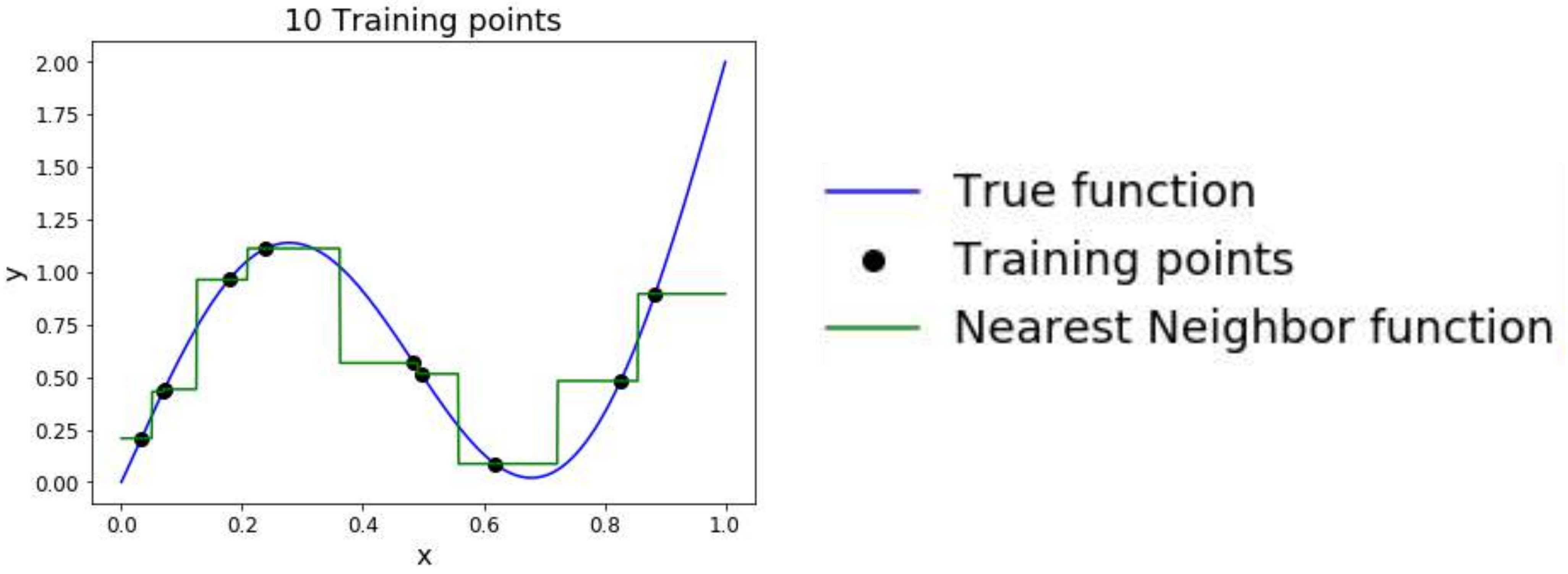
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

K-Nearest Neighbor: **Universal Approximation**

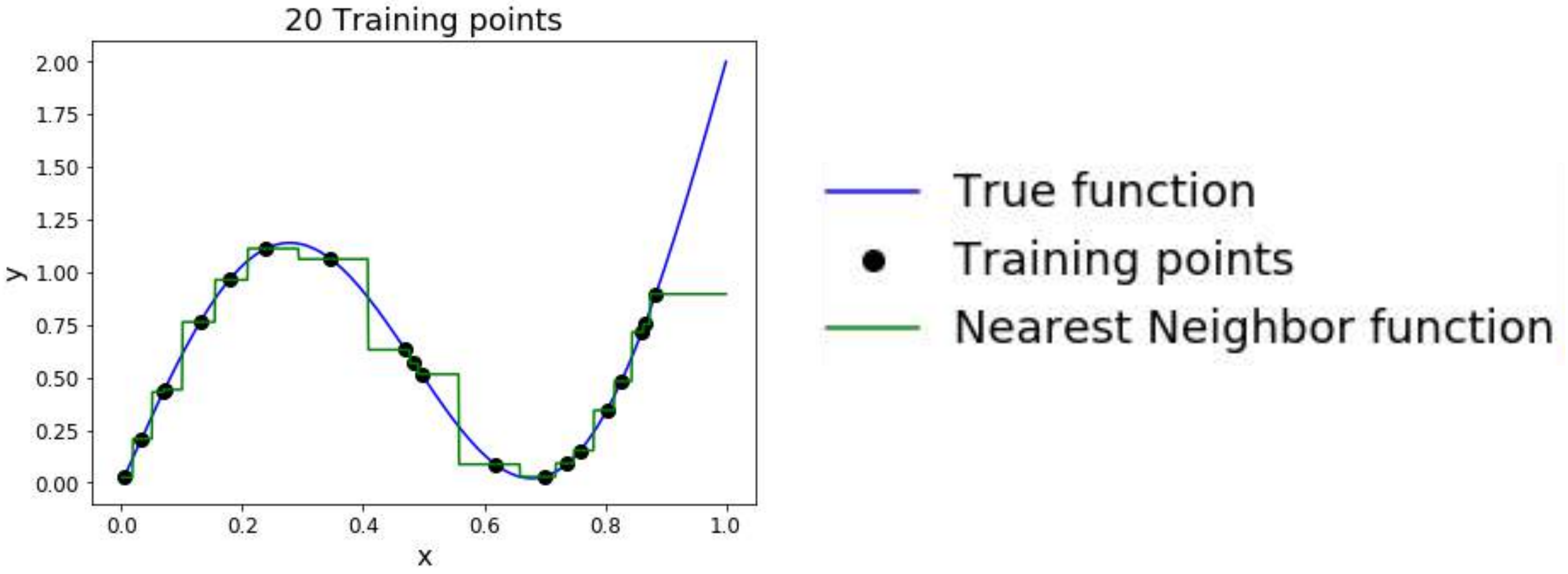
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

K-Nearest Neighbor: **Universal Approximation**

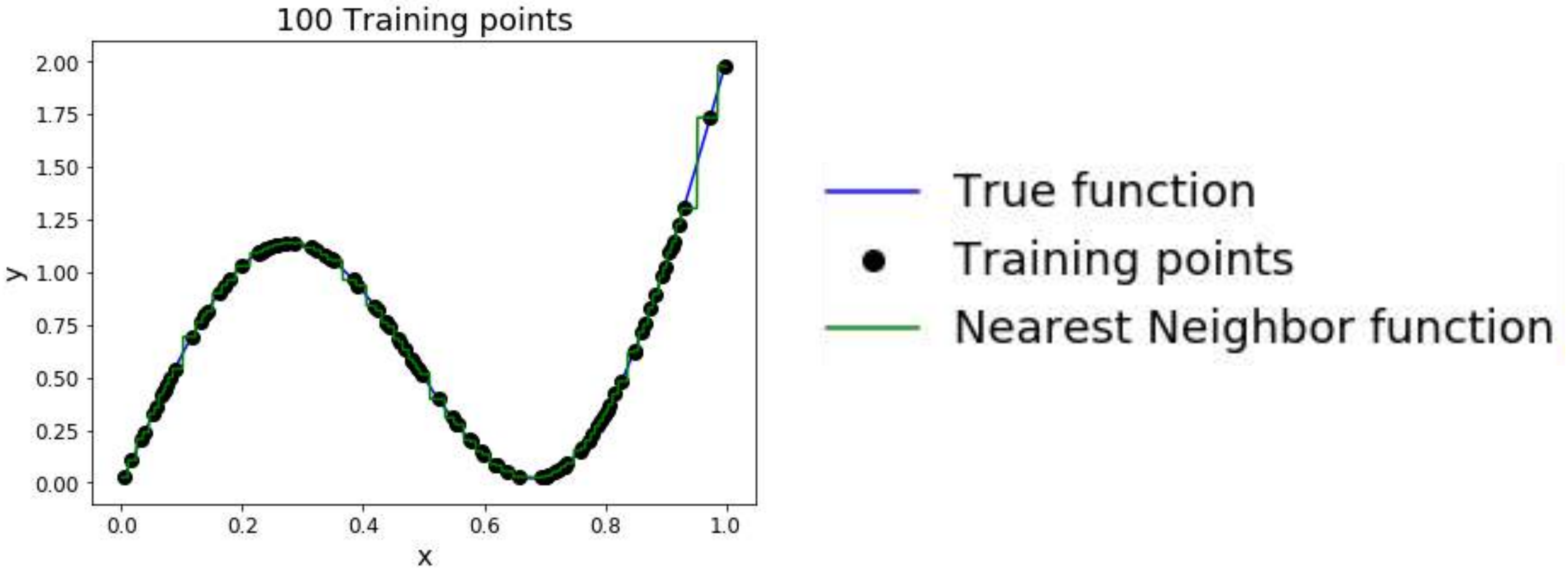
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

K-Nearest Neighbor: **Universal Approximation**

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



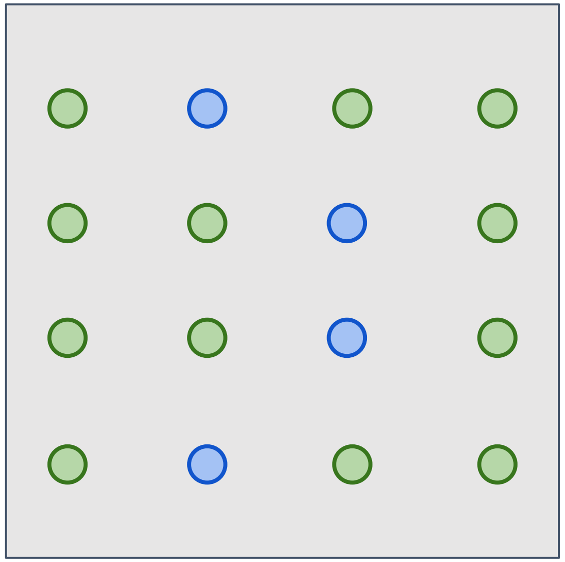
(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

Problem: **Curse of Dimensionality**

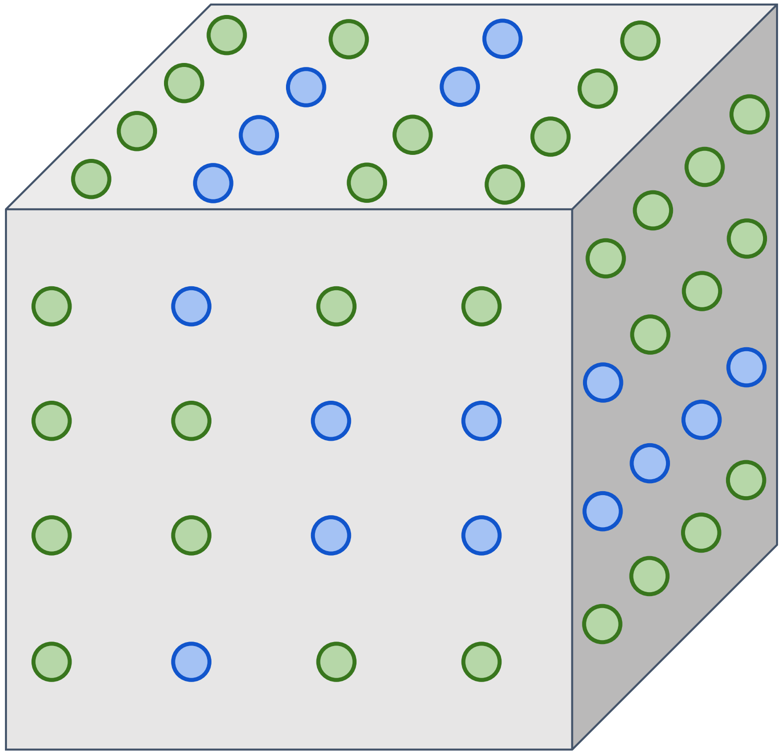
Curse of dimensionality: For uniform coverage of space, number of training points needed grows exponentially with dimension

Dimensions = 1
Points = 4

Dimensions = 2
Points = 4^2



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:

$$2^{32\text{x}32} \approx 10^{308}$$

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:

$$2^{32\text{x}32} \approx 10^{308}$$

Number of elementary particles in the visible universe: (source)

$$\approx 10^{97}$$

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)

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

Summary

In **Image classification** we start with a **training set** of images and labels, and must predict labels on the **test set**

Image classification is challenging due to the semantic gap: we need invariance to occlusion, deformation, lighting, intraclass variation, etc

Image classification is a **building block** for other vision tasks

The **K-Nearest Neighbors** classifier predicts labels based on nearest training examples

Distance metric and K are **hyperparameters**

Choose hyperparameters using the **validation set**; only run on the test set once at the very end!

Next time: Linear Classifiers