Deep Learning

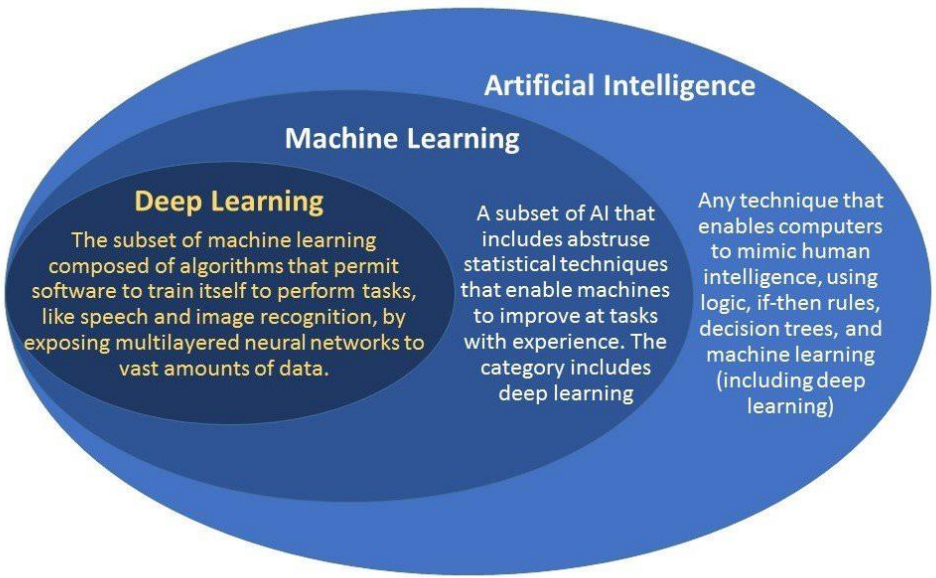
Wojciech Palubicki

Course Goals

- Part 1
  - Introduction into **Computer Vision**
  - Introduction into **Deep Learning**
  - Development of **Deep Neural Networks** for vision tasks
- Part 2
  - Introduction into **Natural Language Processing**
  - Development of **Deep Neural Networks** for NLP

Computer Vision

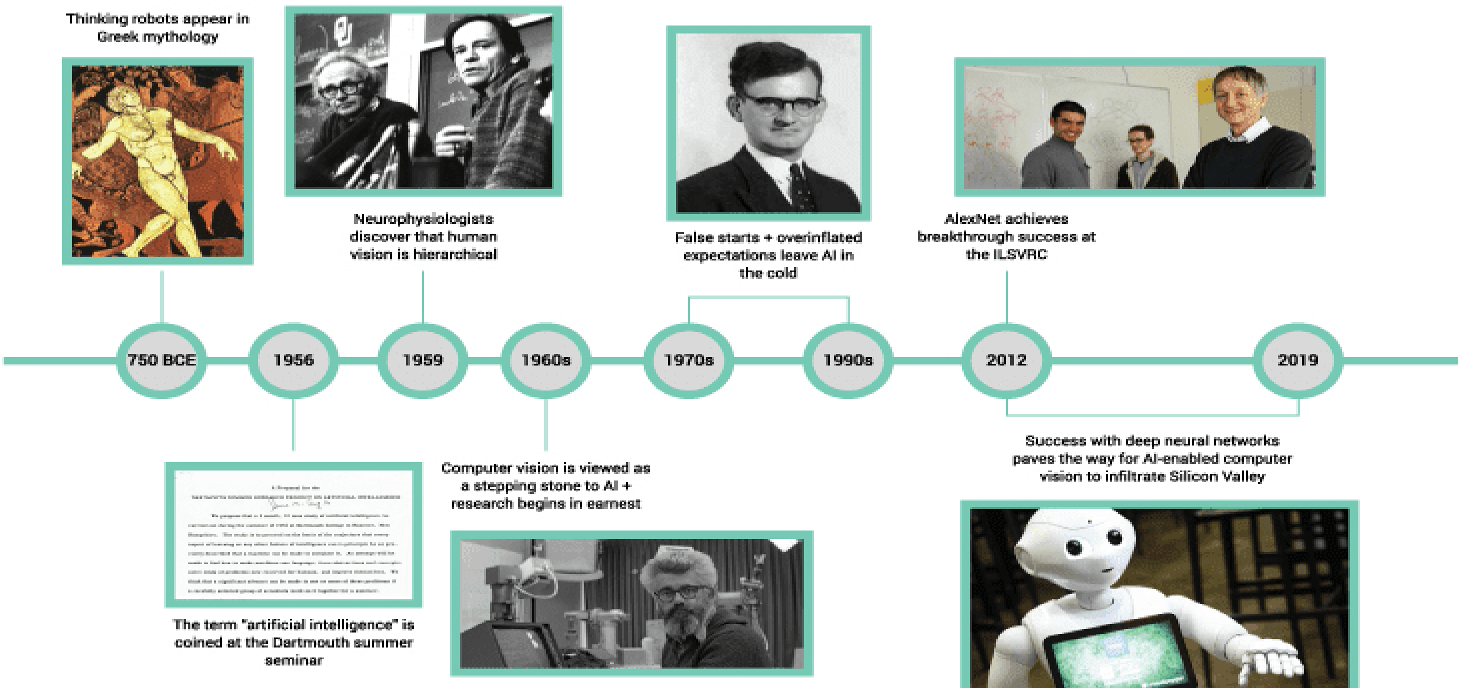
AI and Computer Vision



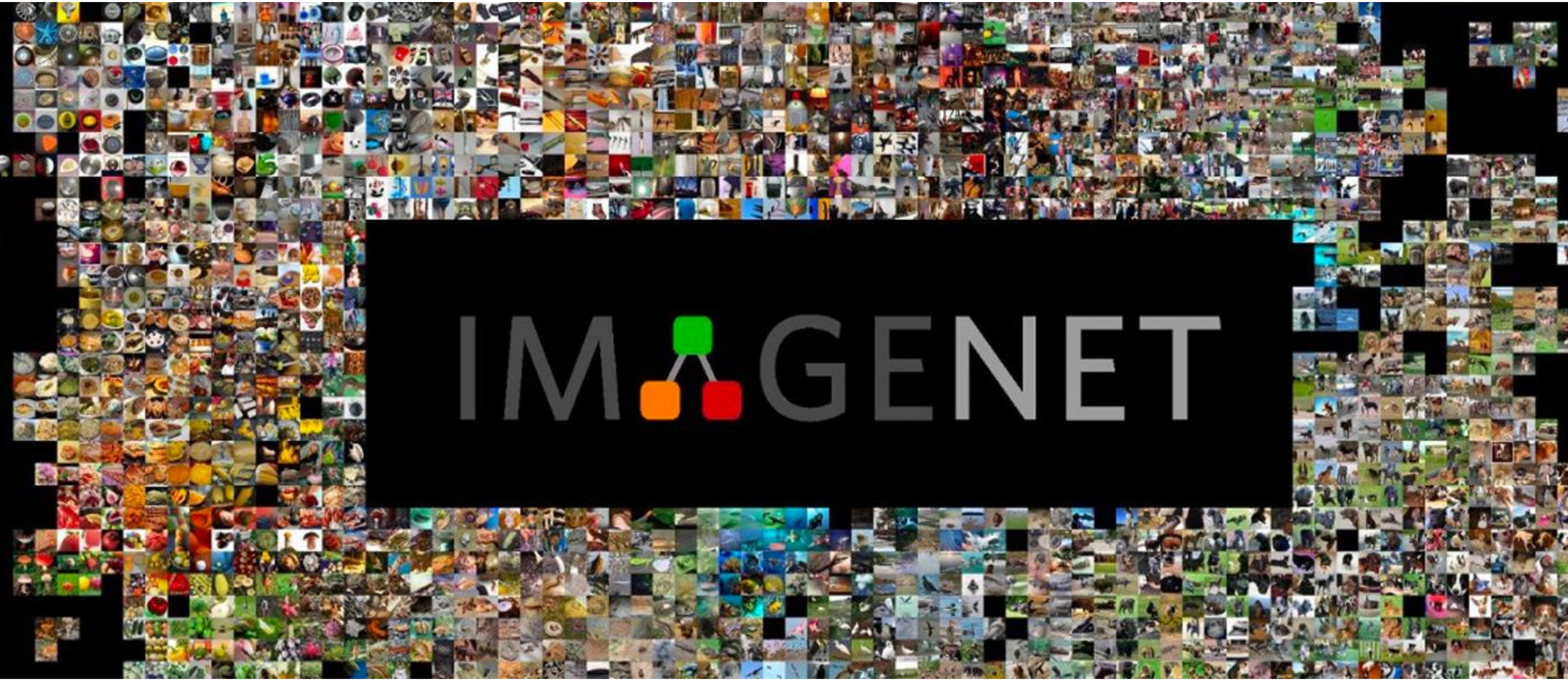
Computer Vision

- Object detection
- Object classification
- Scene understanding
- Semantic scene segmentation
- 3D reconstruction
- Object tracking
- Human pose estimation
- Activity recognition
- VQA
- ....

Computer Vision History



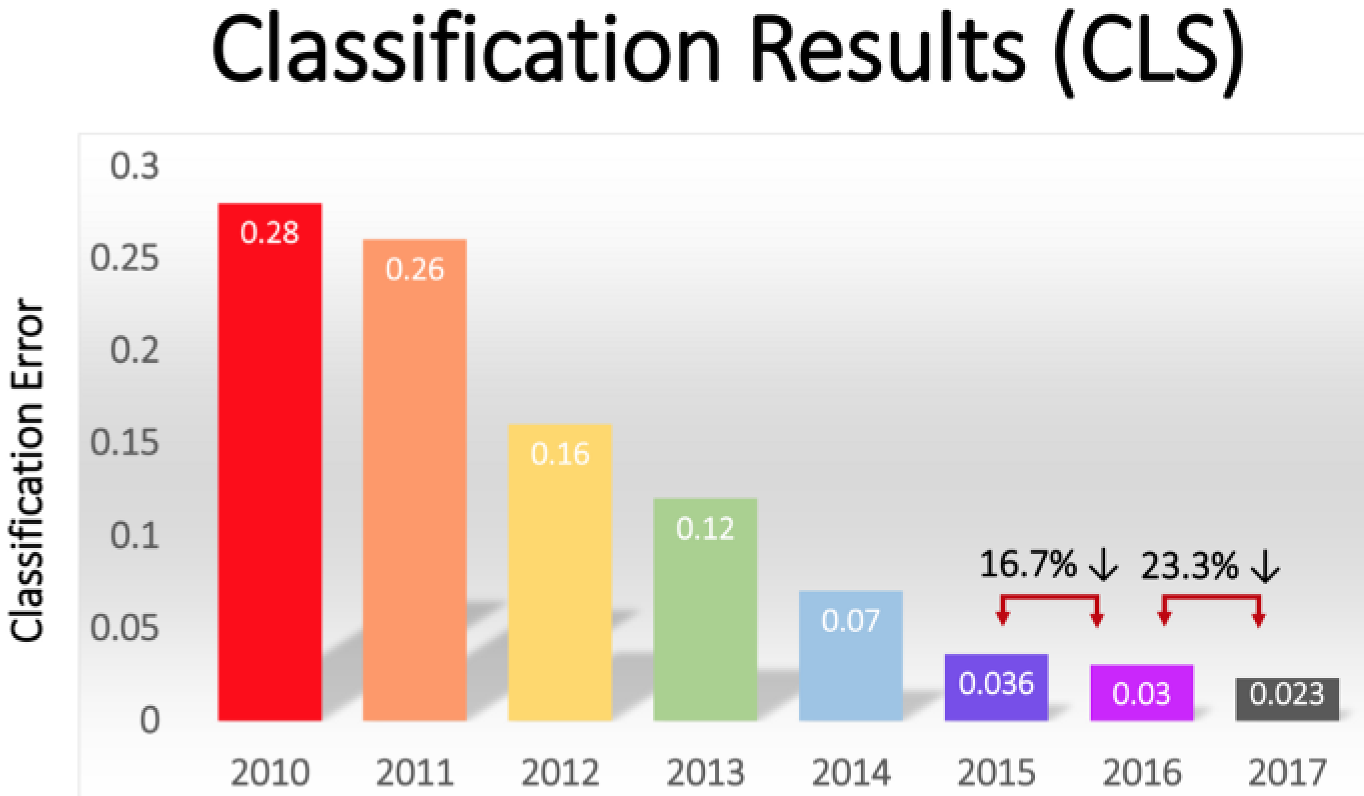
Large Datasets 1M+



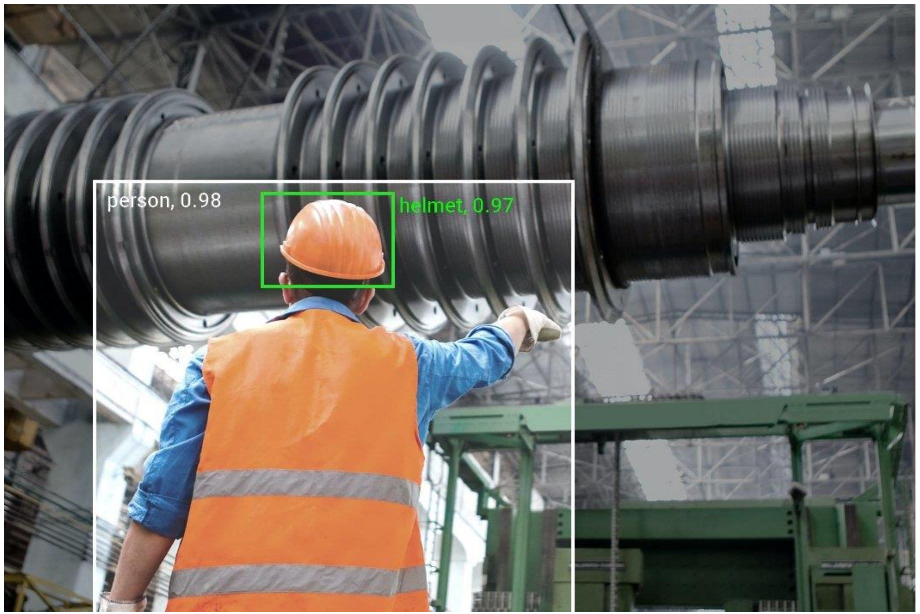
Detailed Labels

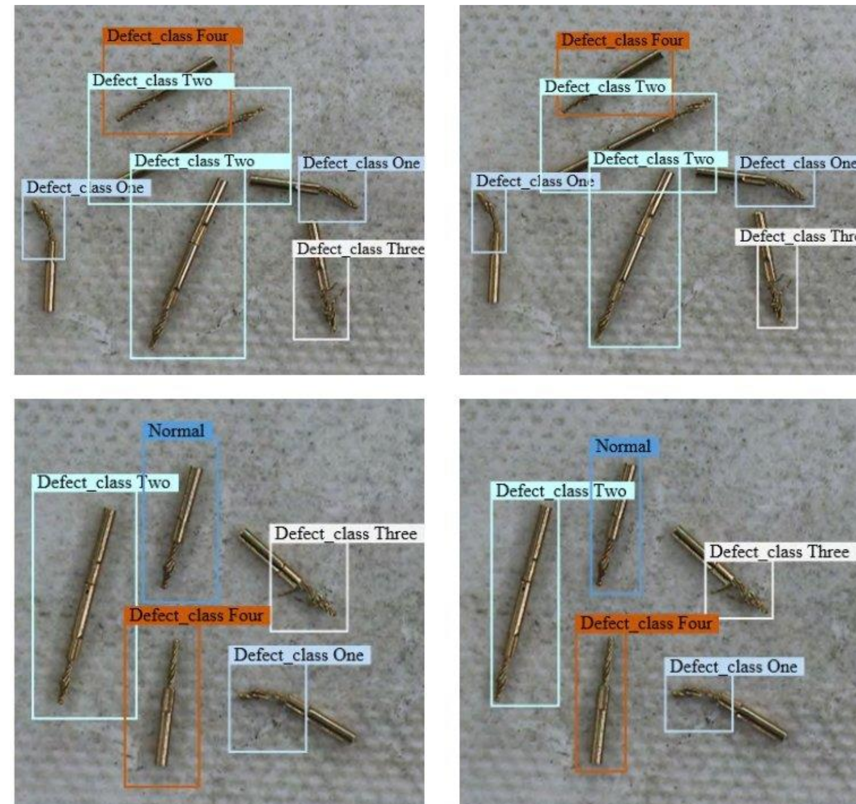
The Image Classification Challenge:
1,000 object classes
1,431,167 images

Imagenet Large Scale Visual Recognition Challenge

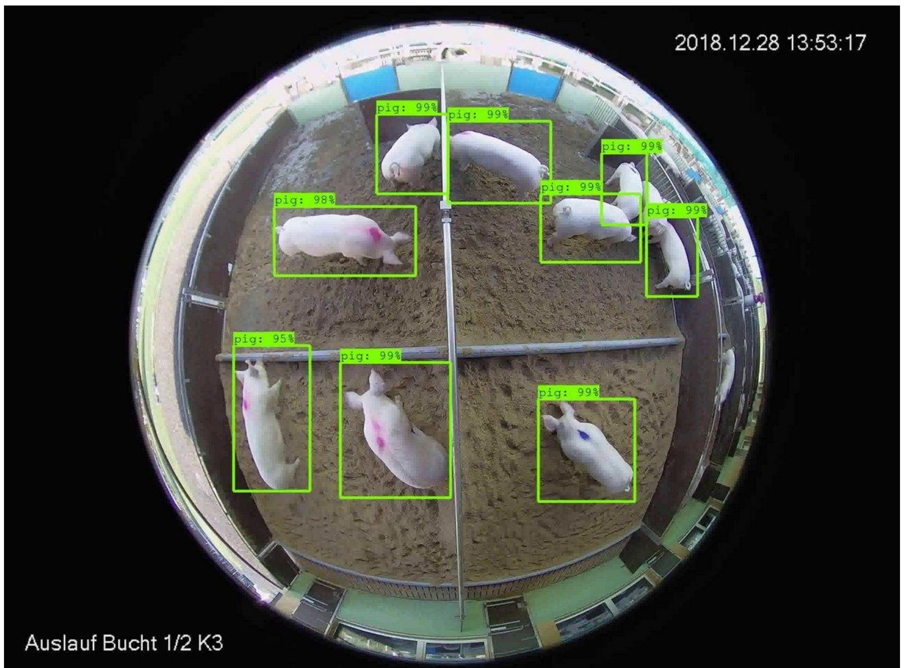


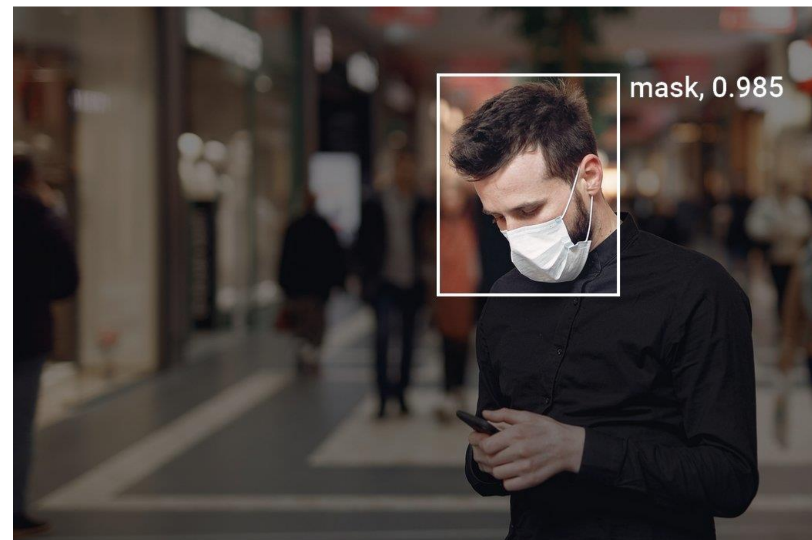
Applications



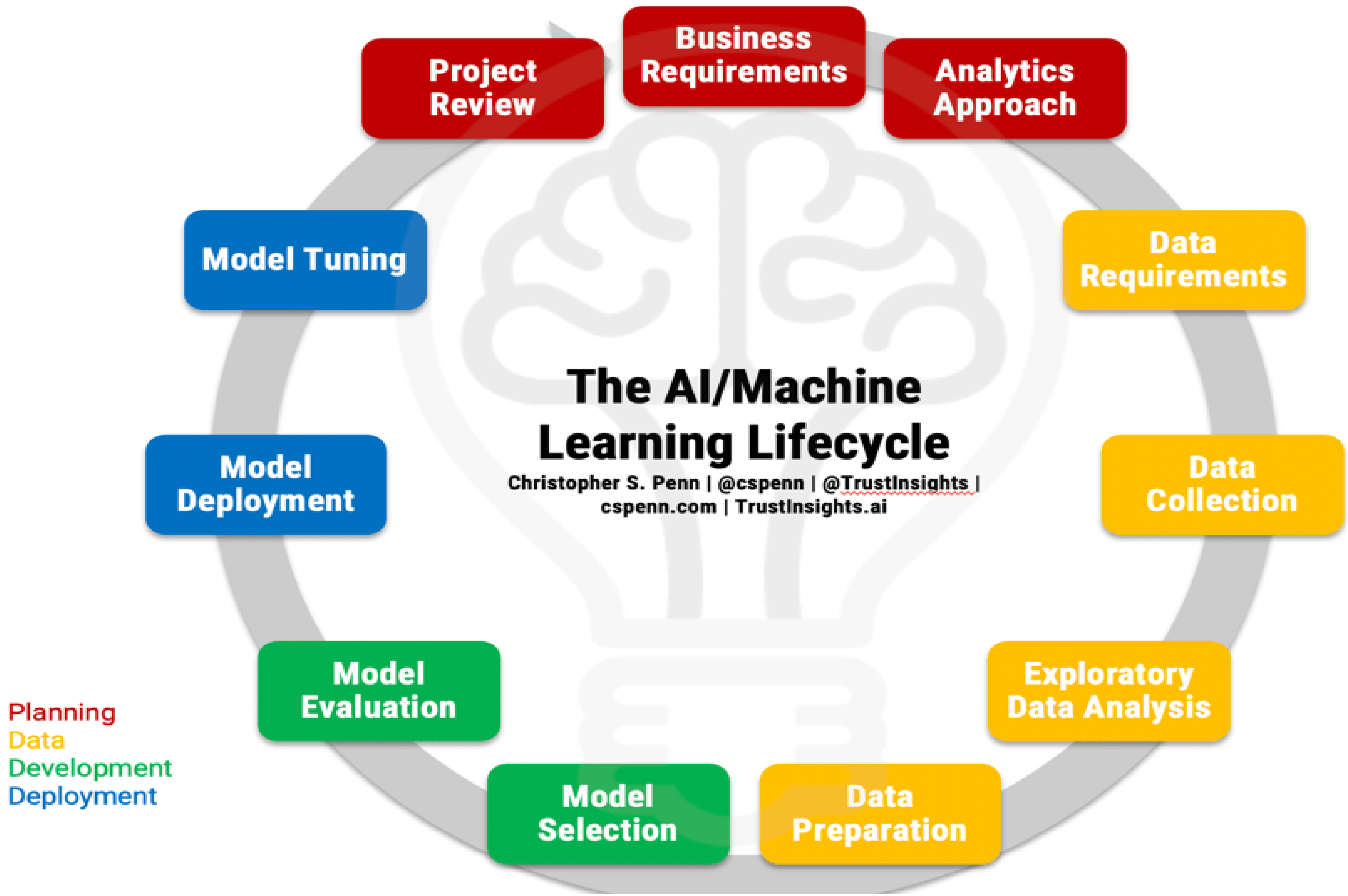


Applications



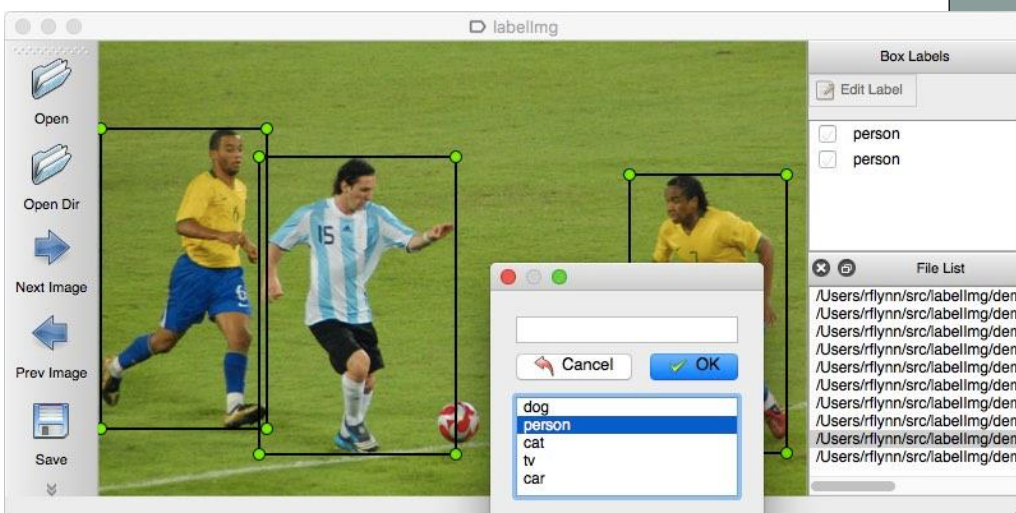


Applications



Business Requirements
Analytics Approach
Data Requirements
Data Collection
Exploratory Data Analysis
Data Preparation
Model Selection
Model Evaluation
Model Deployment
Model Tuning
Project Review
The AI/Machine Learning Lifecycle
Christopher S. Penn | @cspenn | @TrustInsights |
cspenn.com | TrustInsights.ai
Planning
Data
Development
Deployment

Data Collection



Grading

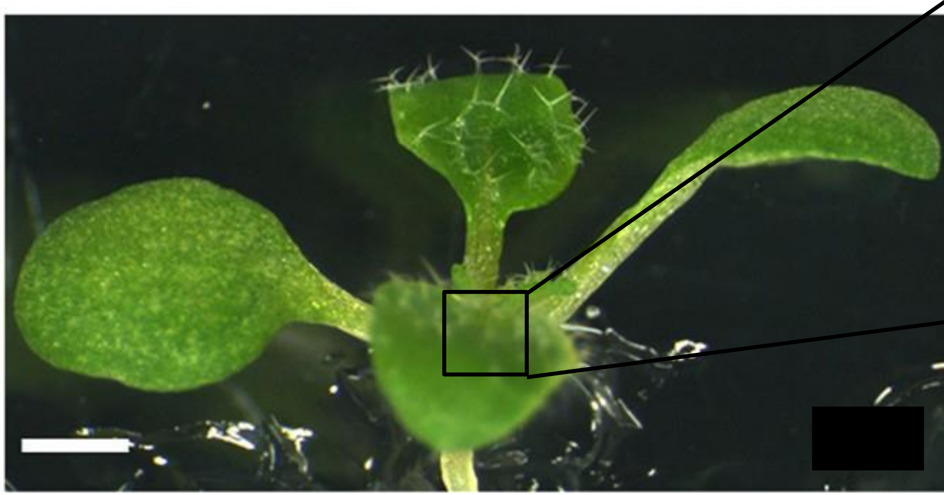
- **Final Exam:** written test about lecture material on computer vision
- Multiple Choice Test
- End of April
- 50% of final lecture grade
- Requirements: 3+ from labs

Image Classification

→ plant

Select correct class from a given set of classes

Image Classification



```
[[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]]
```

Computational representation

An image is a tensor of integers between [0, 255]:

e.g. 1920 x 1080 x 3 (RGB)

Challenges: Different Viewpoints

Pixel values change when the camera moves.

Challenges: Different Backgrounds

Challenges: Different Illumination

Challenges: Occlusion

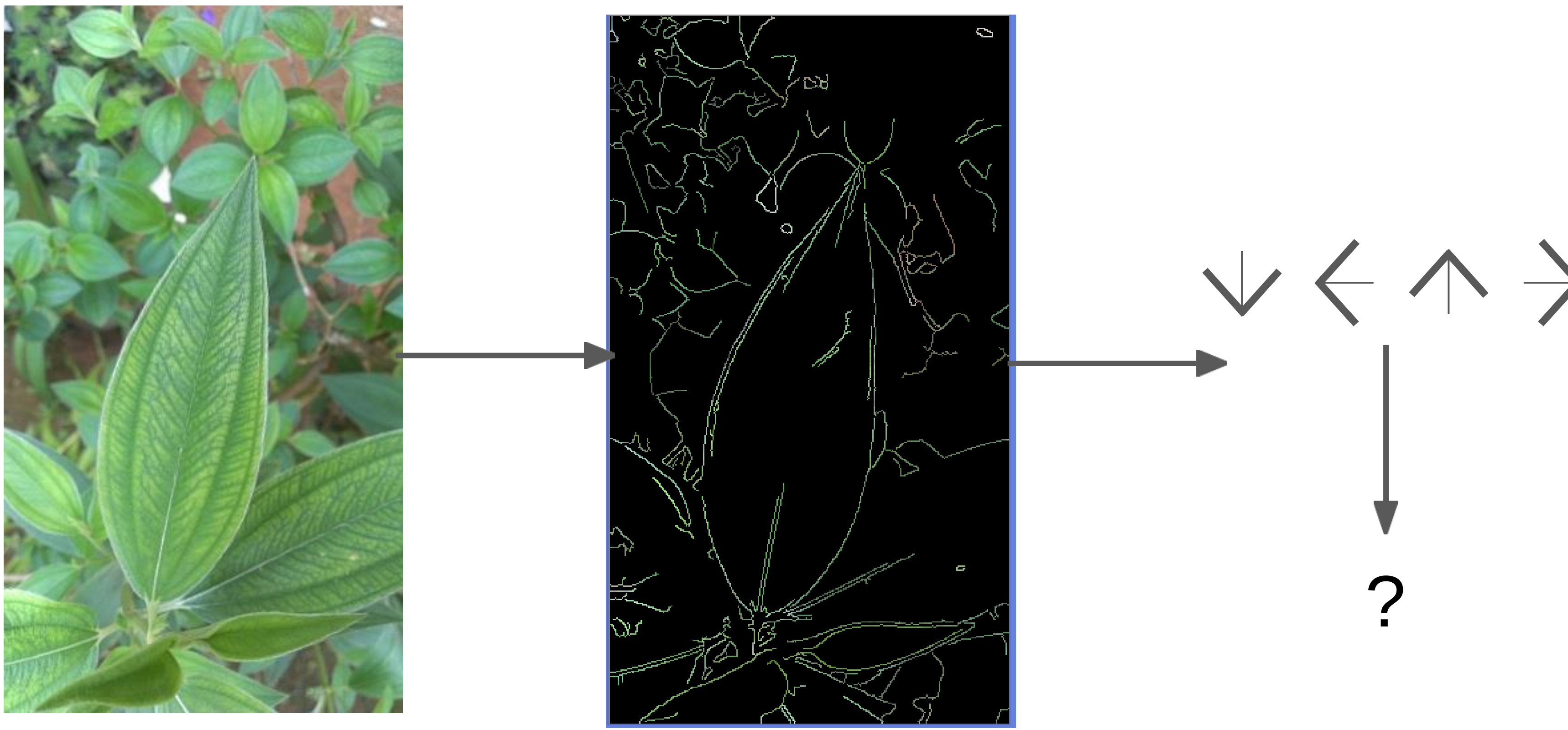
Challenges: Variation

Image Classifier

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

There is no deterministic, trivial way of selecting correct classes given just an input image

Rule-based Methods



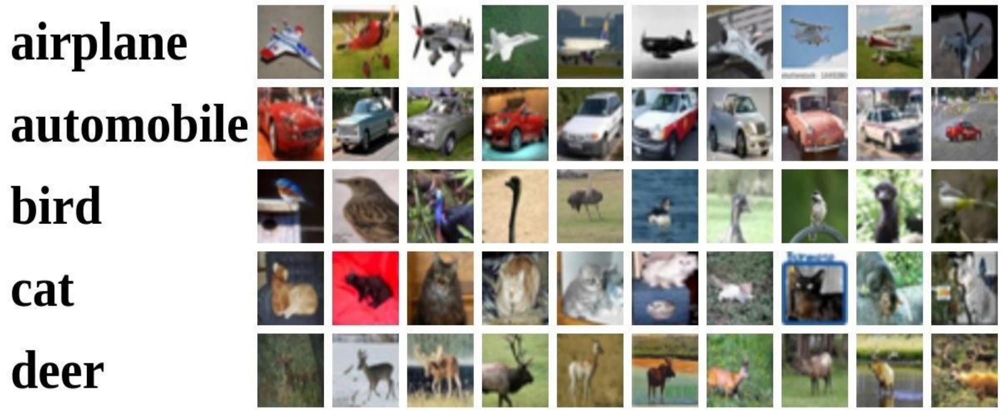
Machine Learning: Data-Driven Approach

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

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

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

Example training set



Nearest Neighbor Classifier

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
```

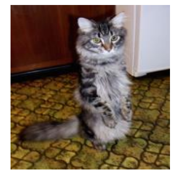
→ Predict the label of the most similar training image

First classifier: **Nearest Neighbor**

deer

bird

plane

cat

car


Training data with labels

?

query data

Distance Metric | ,  | $\to \mathbb{R}$

Distance Metric to compare images

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

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
```

```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

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

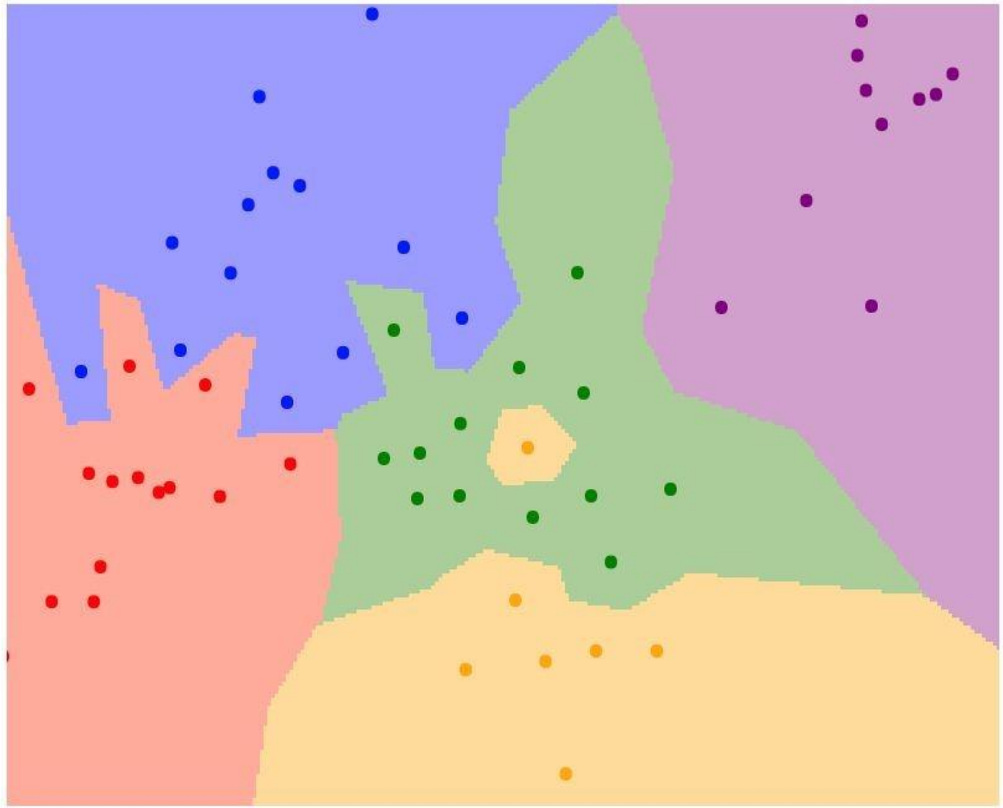
For each test image:
  Find closest train image
  Predict label of nearest image

```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

Train O(1)
Predict O(N)

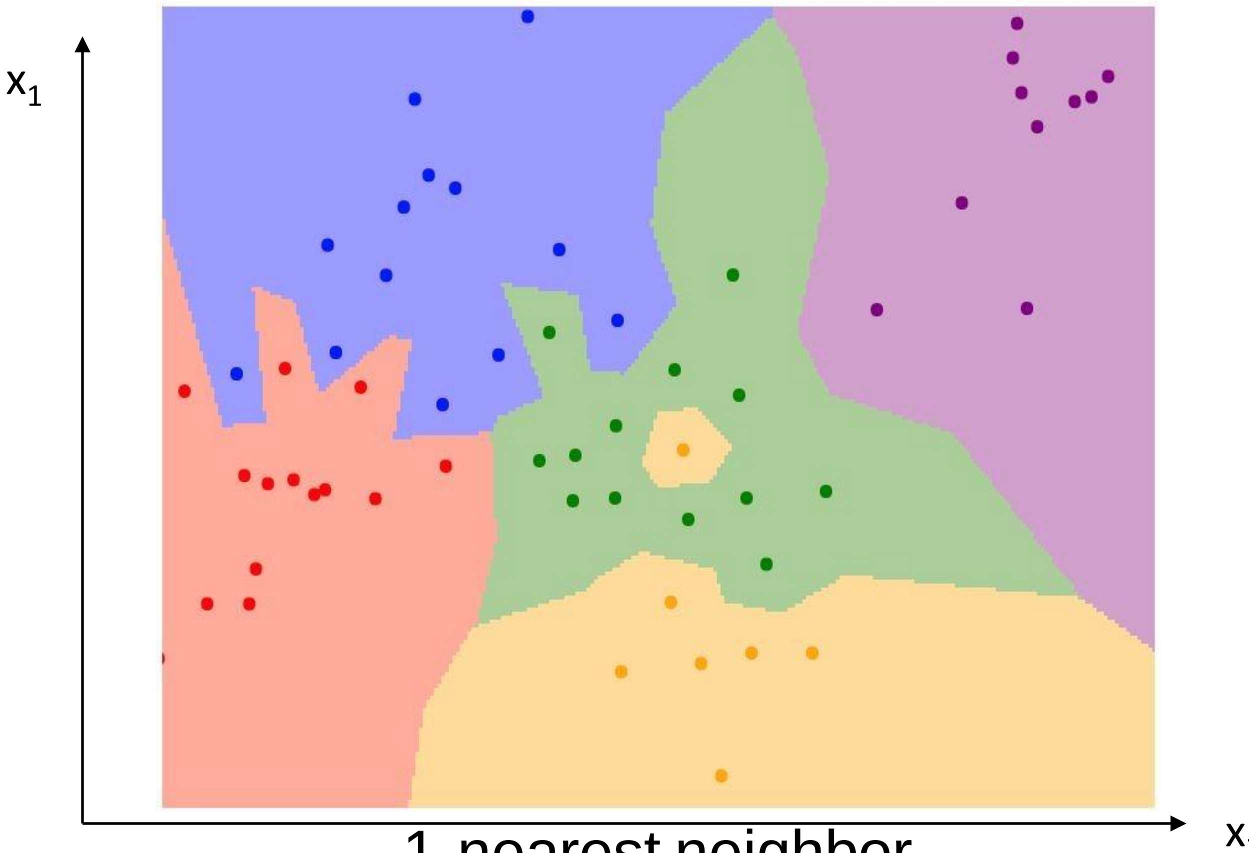
Example



1-nearest neighbor

https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py

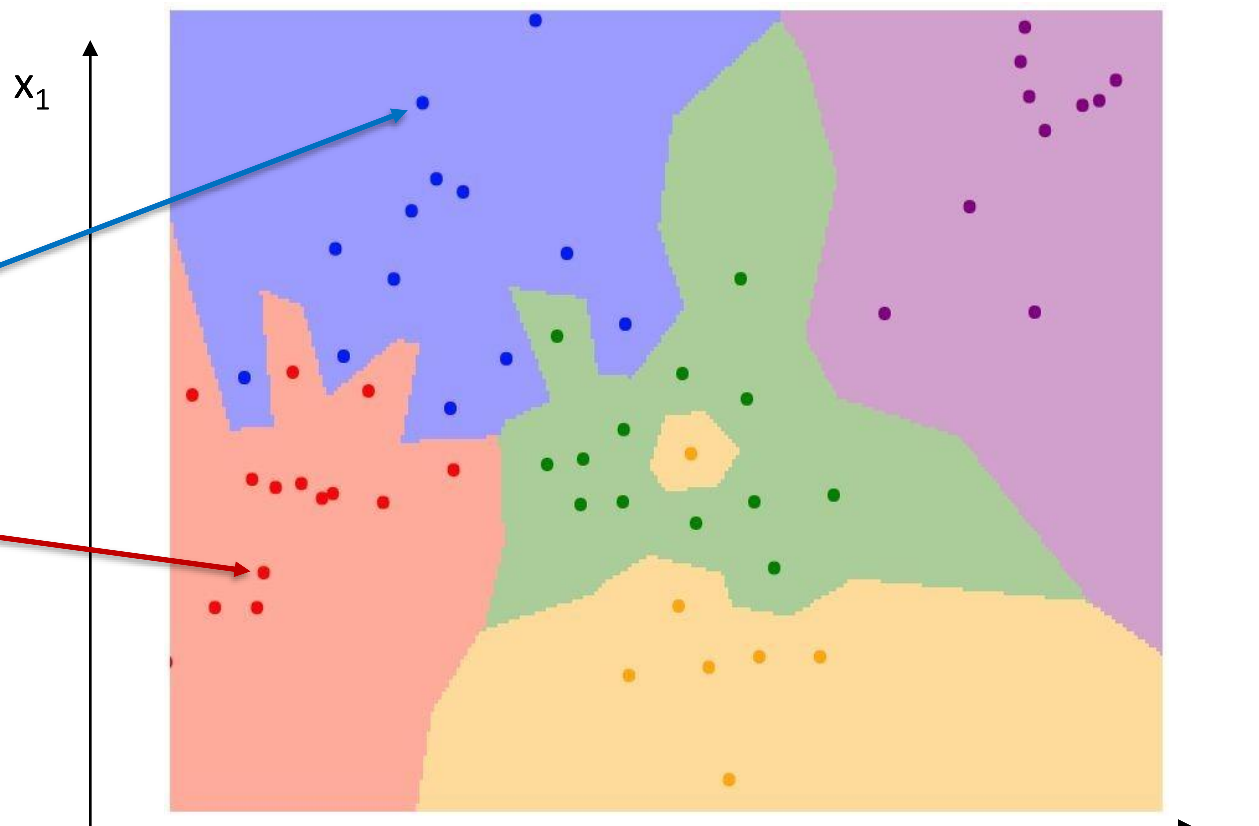
Example



1-nearest neighbor

Example

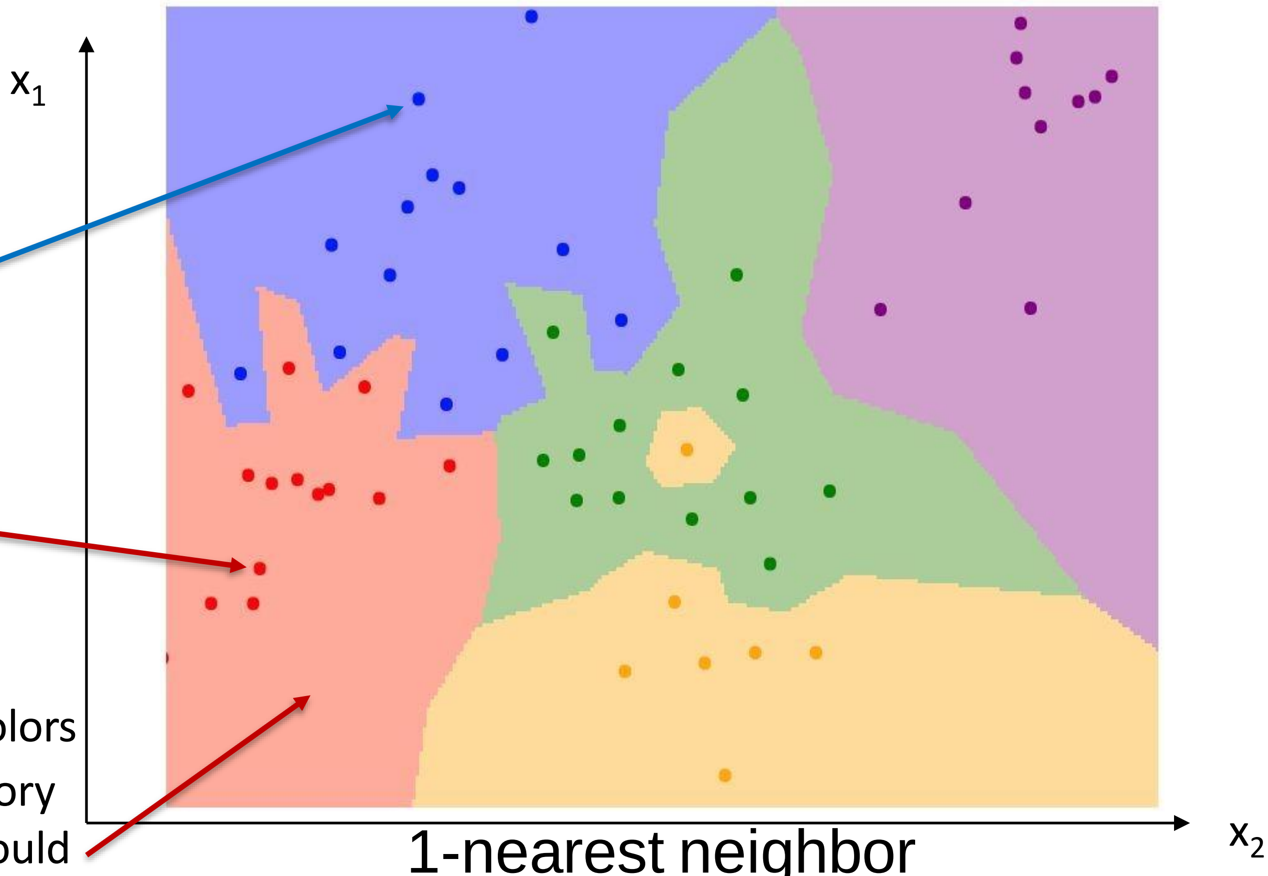
Points are training examples; colors give training labels



1-nearest neighbor

Example

Points are training examples; colors give training labels

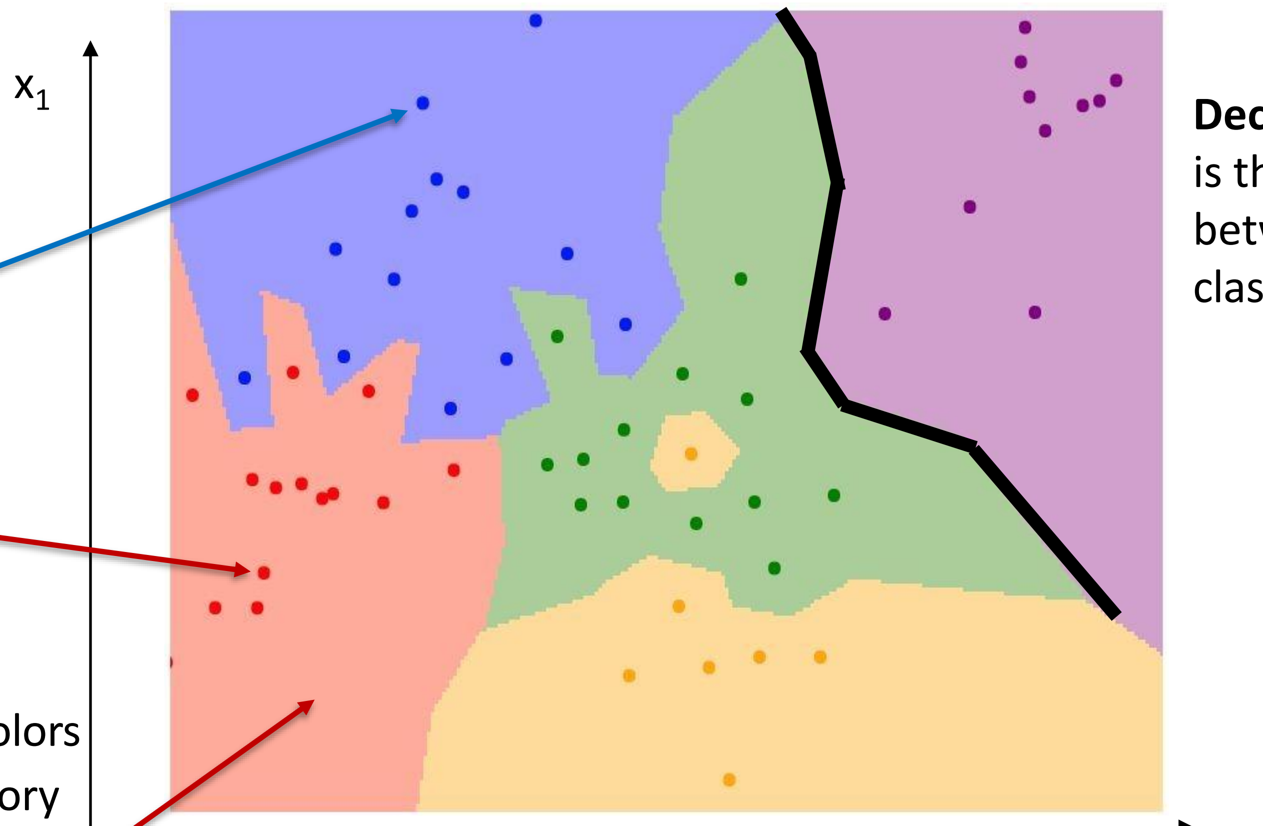


Background colors give the category a test point would be assigned

Example

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



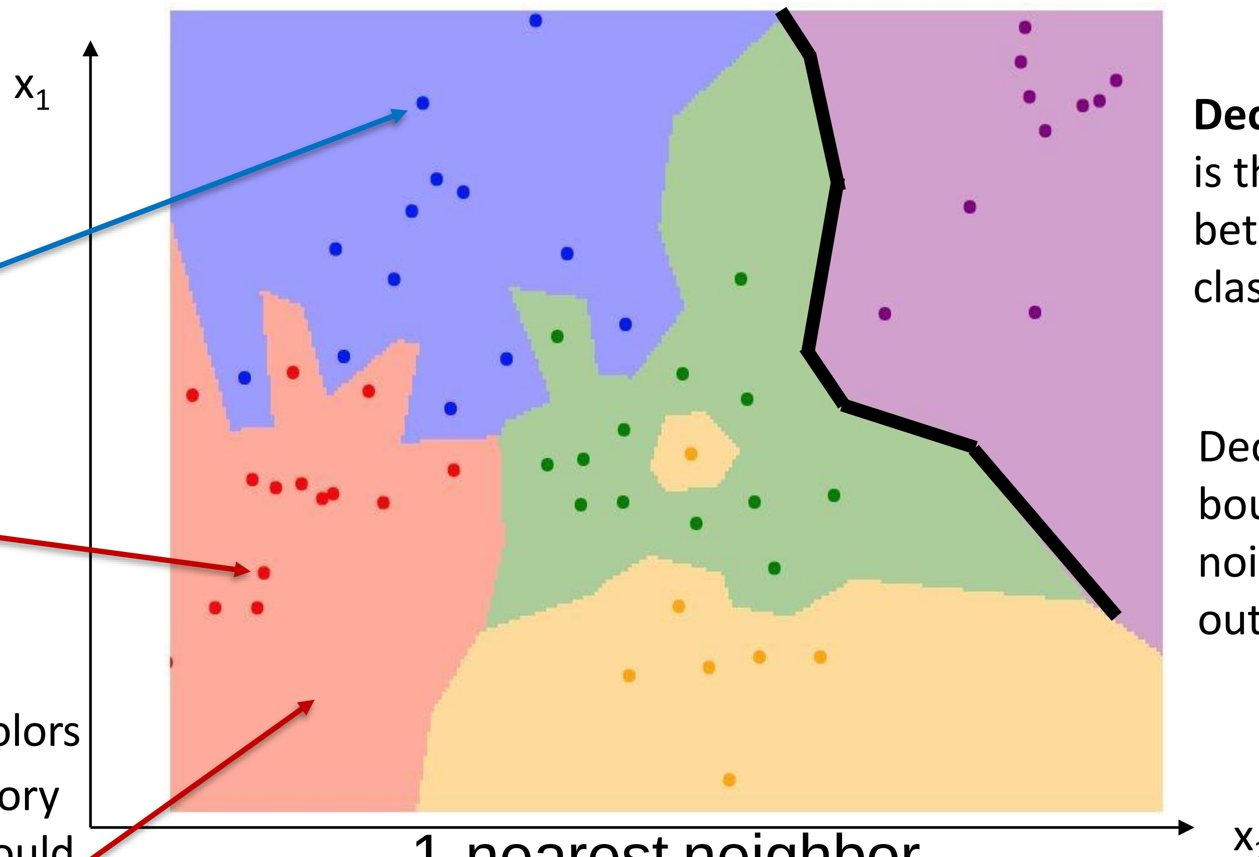
1-nearest neighbor

Decision boundary is the boundary between two classification regions

Example

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



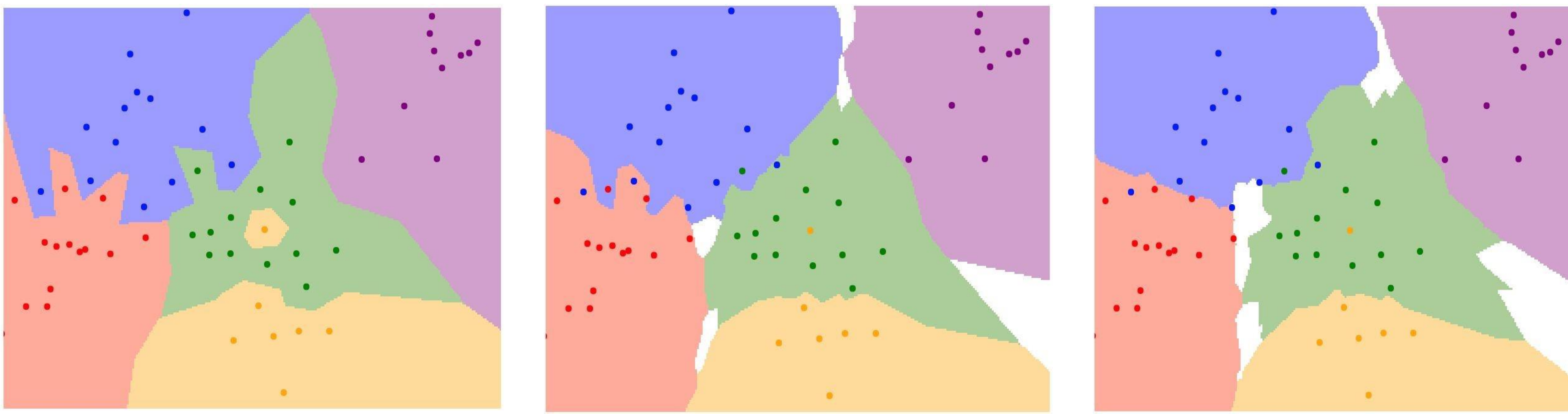
1-nearest neighbor

Decision boundary is the boundary between two classification regions

Decision boundaries can be noisy; affected by outliers

K-Nearest Neighbors

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

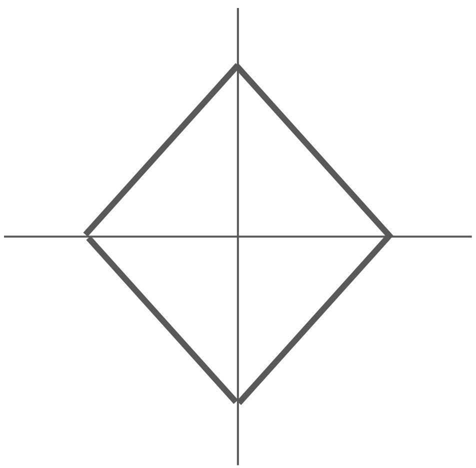


K = 1 K = 3 K = 5

K-Nearest Neighbors: Distance Metric

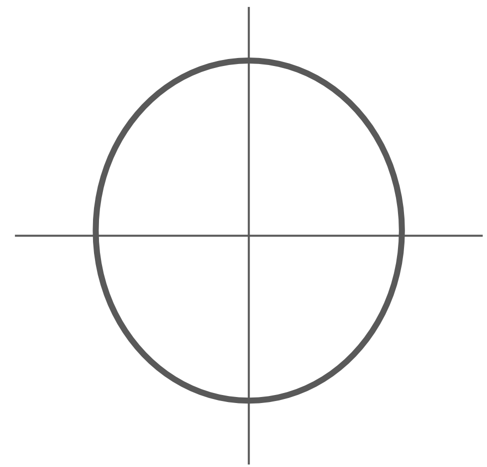
L1 (Manhattan) distance

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



L2 (Euclidean) distance

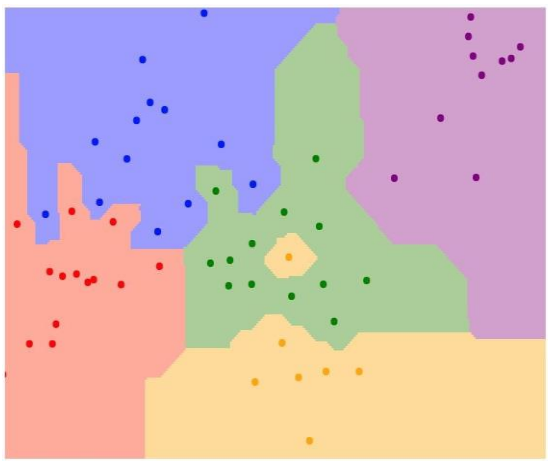
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p \right)^2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

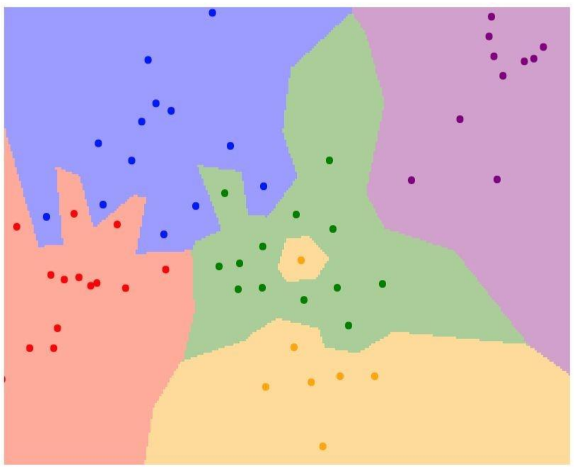
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



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

Hyperparameters

What is the optimal value of **k** to use?
What is the optimal **distance metric** to use?

Hyperparameters are choices about the algorithms themselves we can’t learn.

Hyperparameters

What is the optimal value of **k** to use?
What is the optimal **distance metric** to use?

Hyperparameters are choices about the algorithms themselves we can't learn.

Problem-dependent: try different configuration settings

Setting Hyperparameters

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

Setting Hyperparameters

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

train

Setting Hyperparameters

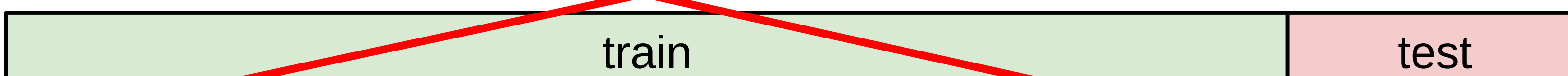
~~**Idea #1**: Choose hyperparameters that work best on the **training data**~~

Idea #2: choose hyperparameters that work best on **test** data

Setting Hyperparameters

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

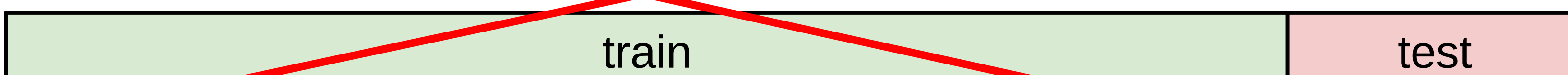
~~**Idea #2**: choose hyperparameters that work best on **test** data~~



Setting Hyperparameters

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

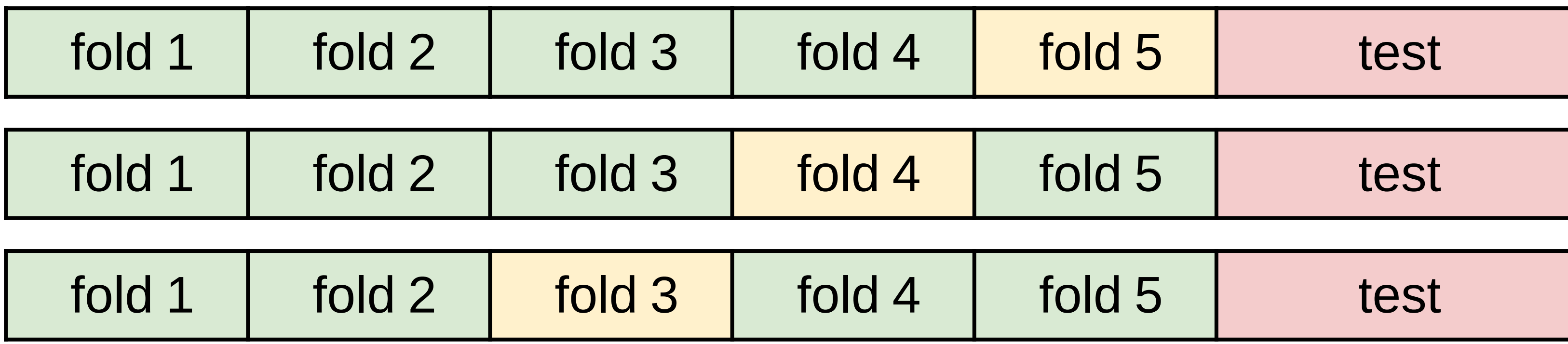
~~**Idea #2**: choose hyperparameters that work best on **test** data~~



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

Setting Hyperparameters

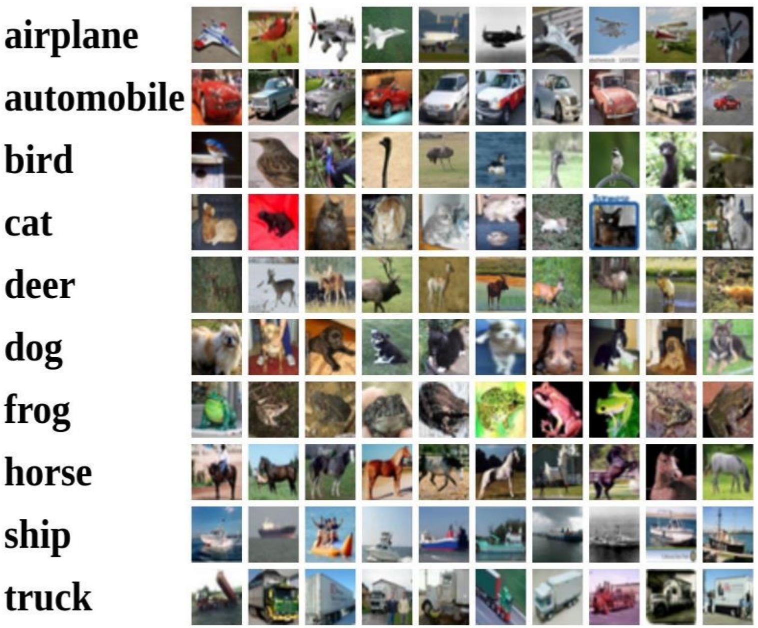
Idea #4: **Cross-Validation**: Split data into **folds**, try each fold as validation and average the results



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

Example Dataset: **CIFAR10**

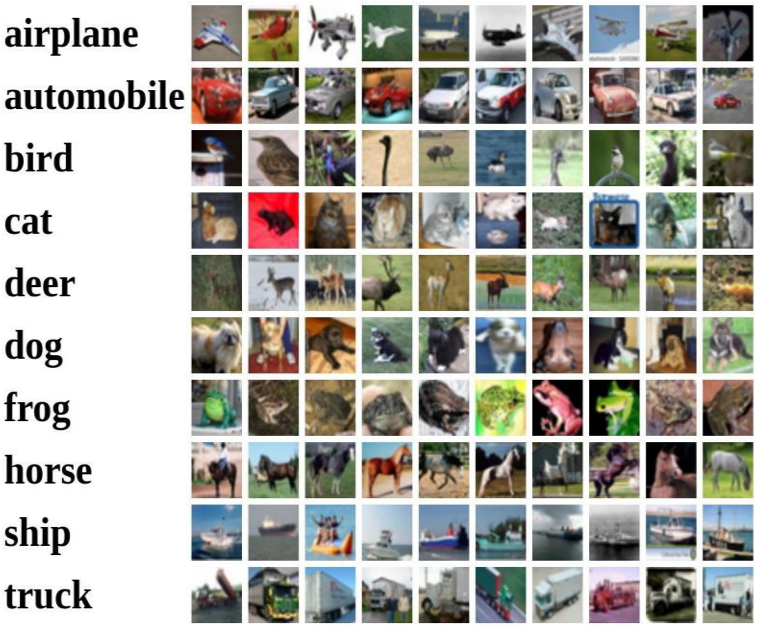
10 classes
50,000 training images
10,000 testing images



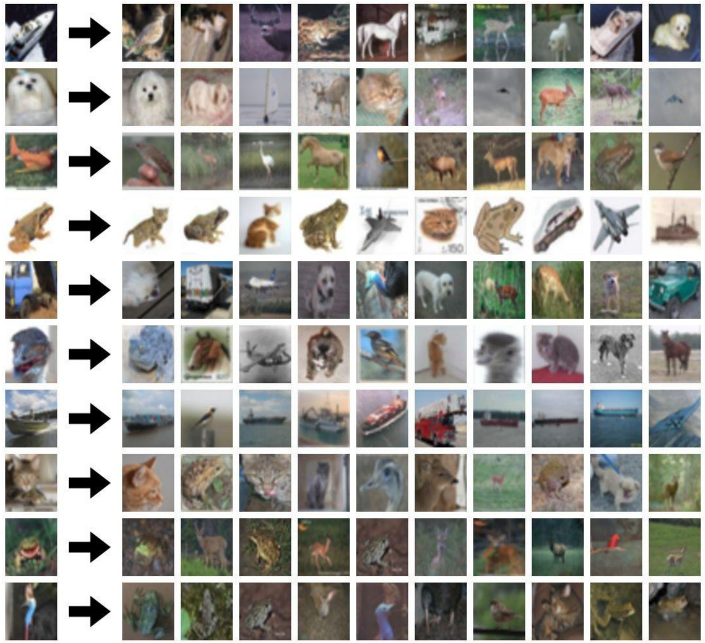
Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

Example Dataset: **CIFAR10**

10 classes
50,000 training images
10,000 testing images

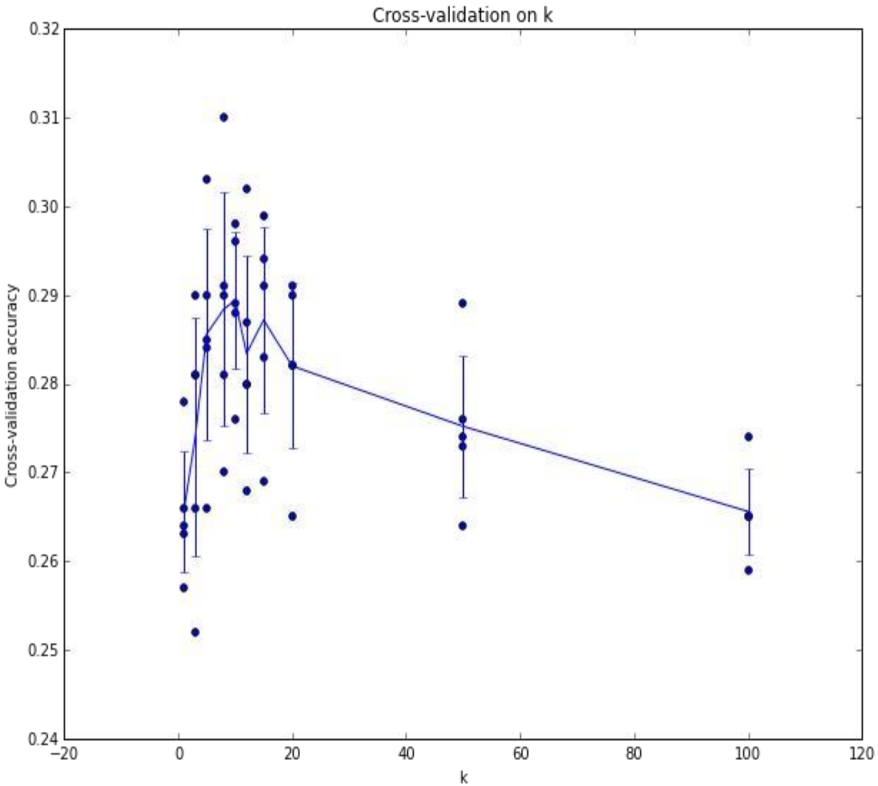


Test images and nearest neighbors



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

Setting Hyperparameters



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

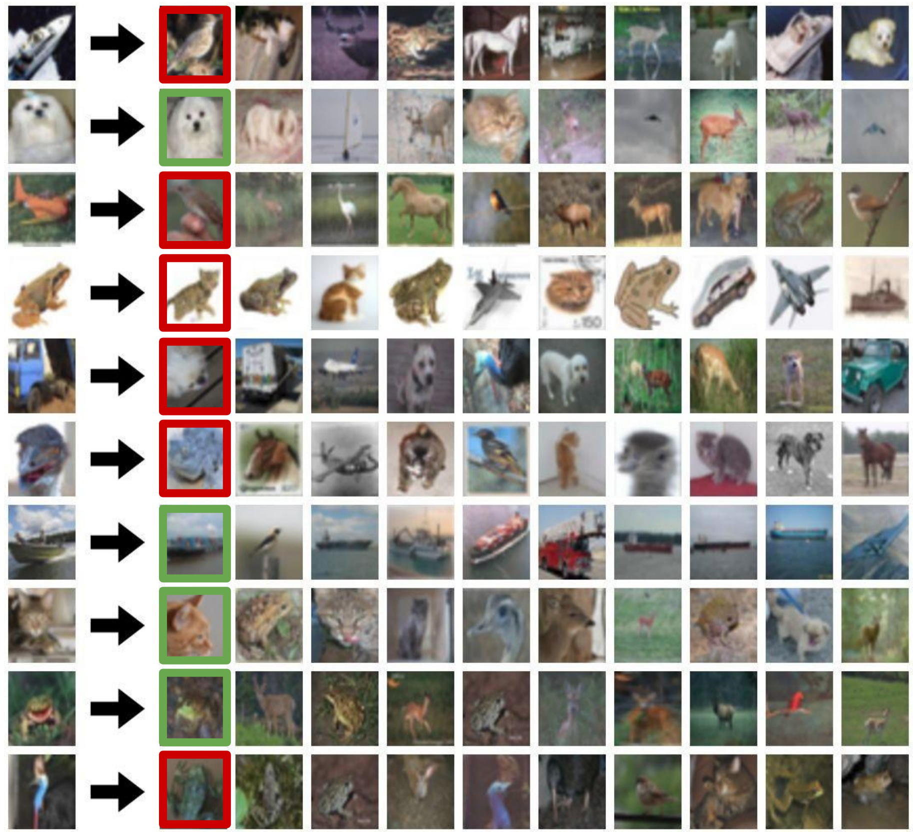
Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

(Seems that k ~= 7 works best
for this data)

kNN Results

kNN Results



K-Nearest Neighbors Summary

Image classification starts with a **training set** of images and labels. It predicts labels on a **test set.**

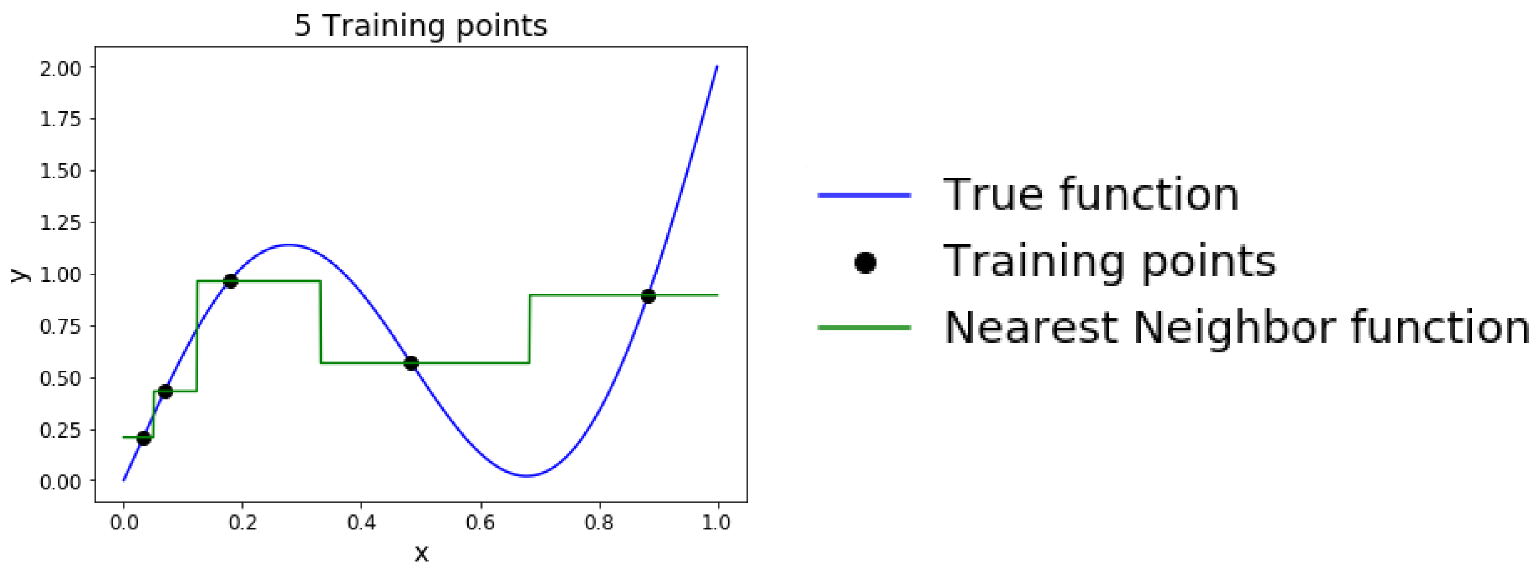
The **k-Nearest Neighbors** classifier predicts labels based on the k nearest training examples

Distance metric and k are **hyperparameters**

Select hyperparameter values using a **validation set**

K-Nearest Neighbor: Universal Approximation

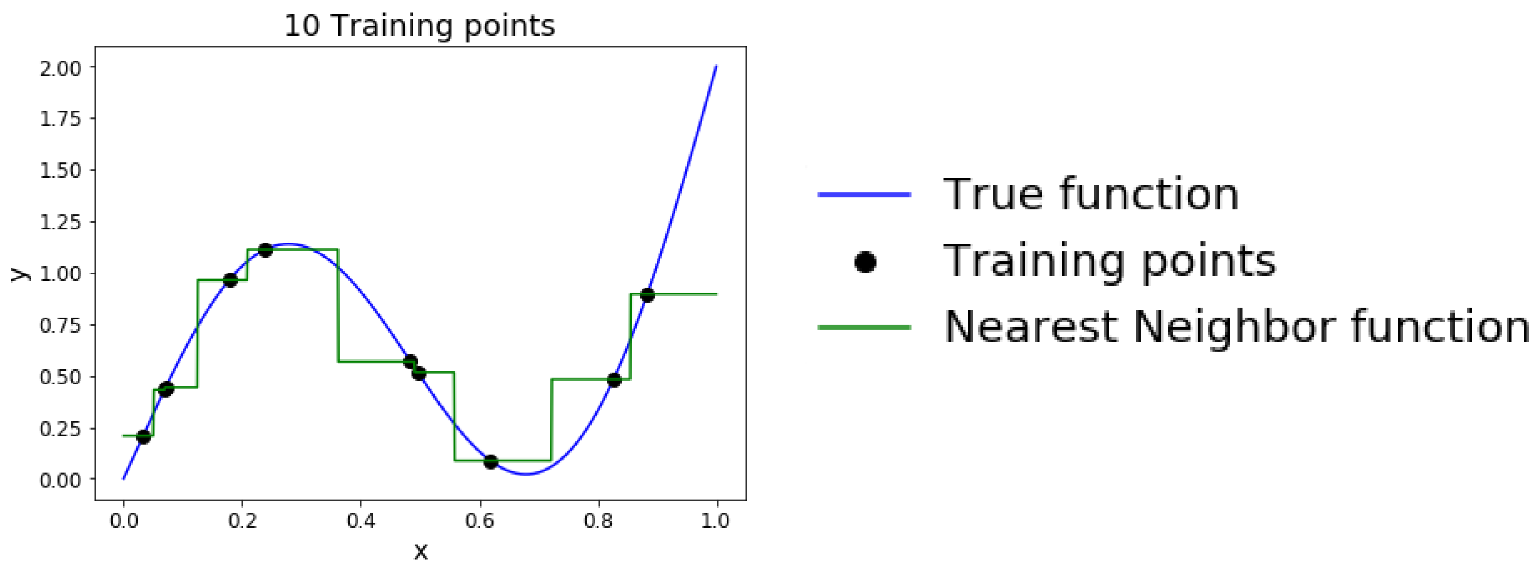
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



(*) 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

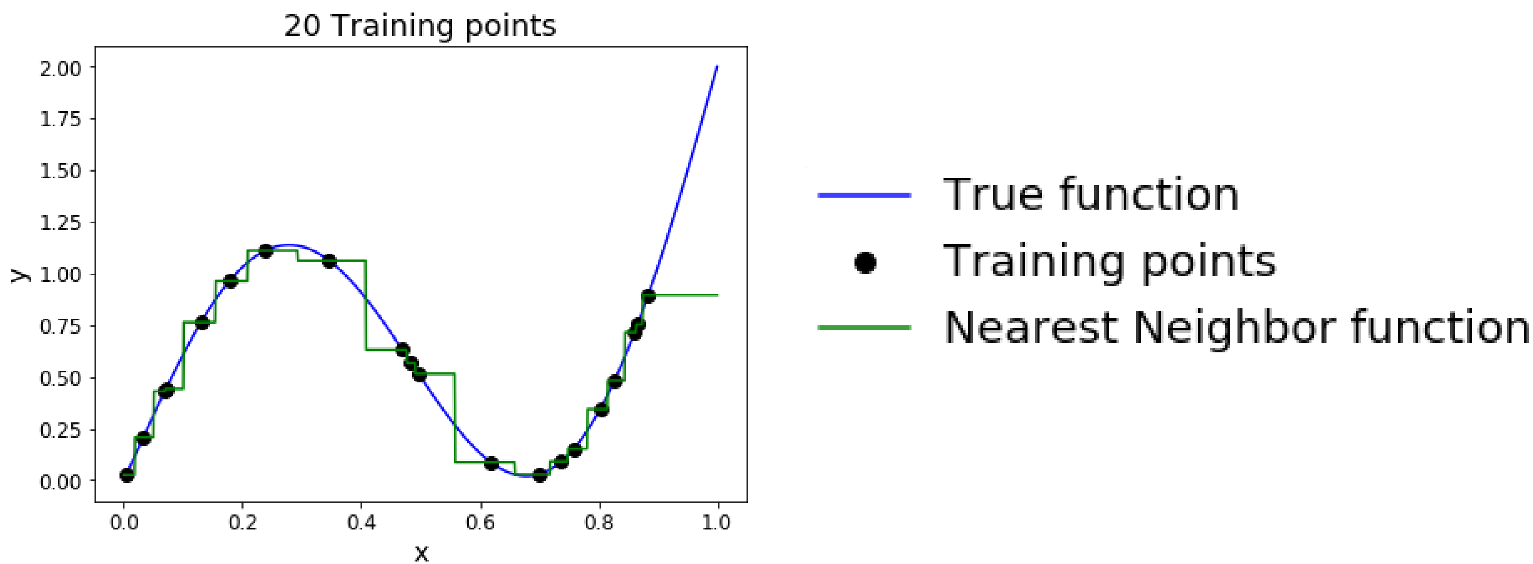
As the number of training samples goes to infinity, nearest neighbor can represent any$^{(*)}$ function!



(*) 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

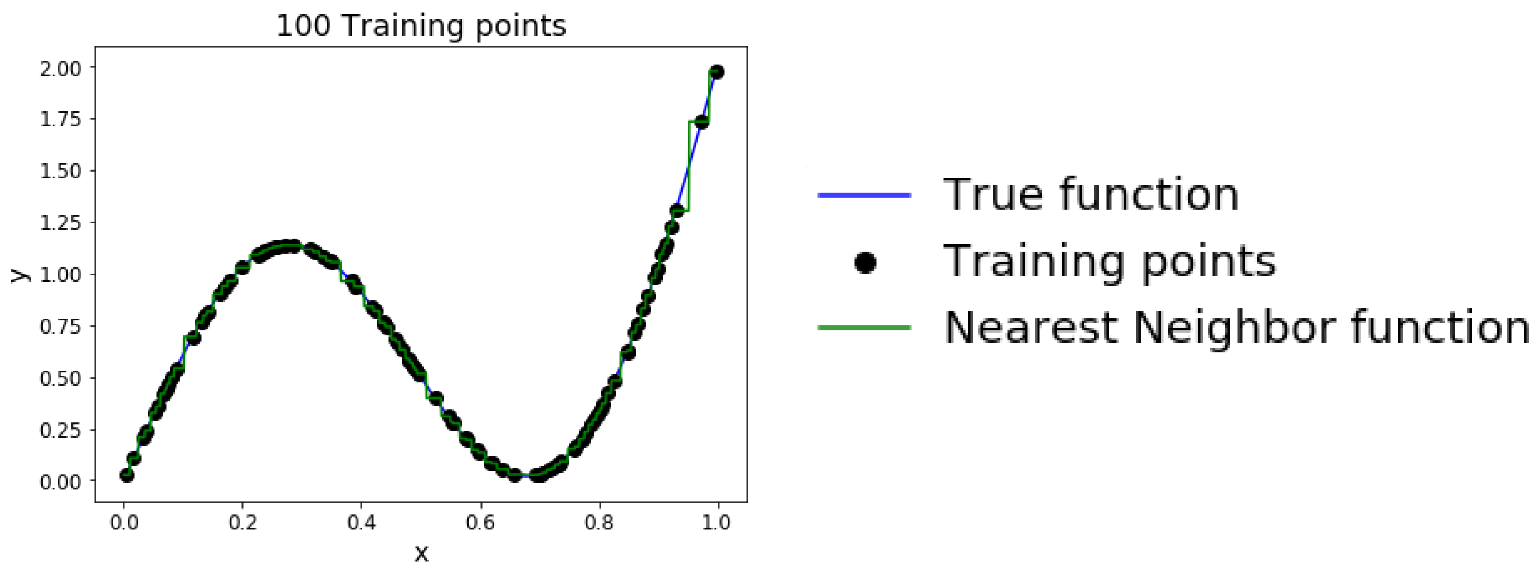
As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!



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

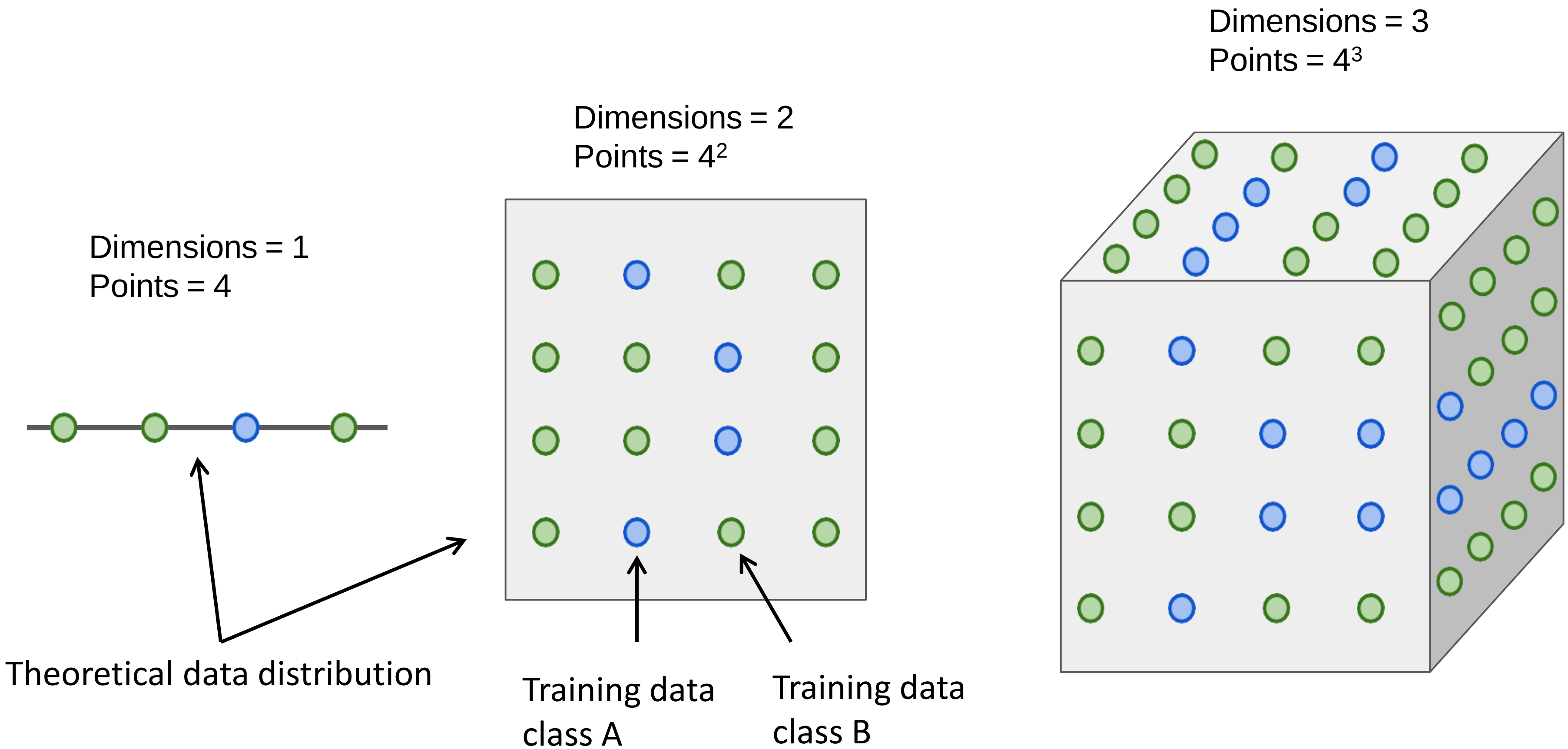
K-Nearest Neighbor: Universal Approximation

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.

Spatial Coverage Needs Increases with Dimension



Spatial Coverage Needs Increases with Dimension

Number of possible 32x32 binary images:

$2^{32x32} \approx 10^{308}$

Number of elementary particles in the visible universe:

$\approx 10^{97}$

k-Nearest Neighbor Drawbacks

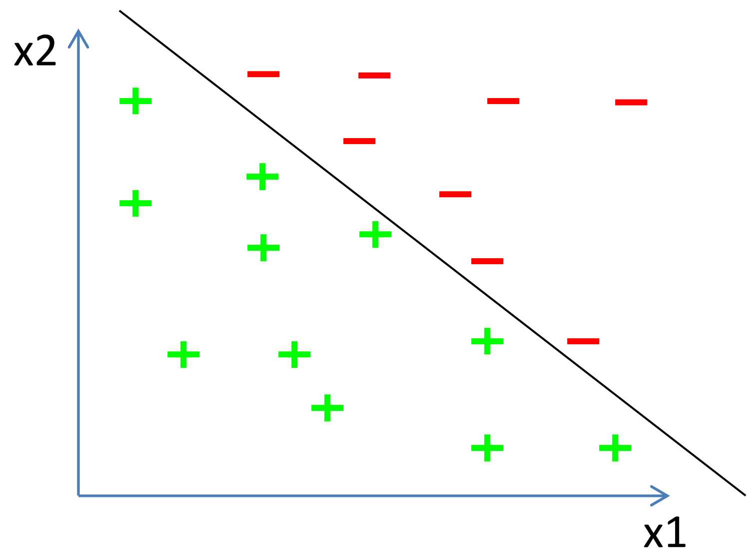
- Distance metrics on pixels are not informative
- Very slow at prediction

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

Linear Classifier

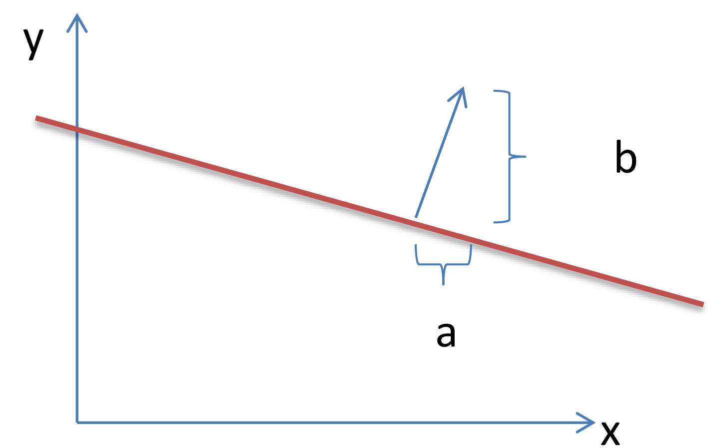
Linear classifiers : Motivation

- kNN produce decision boundaries by calculating them during prediction.
- Can we define a (simple) function during training to define decision boundaries directly?



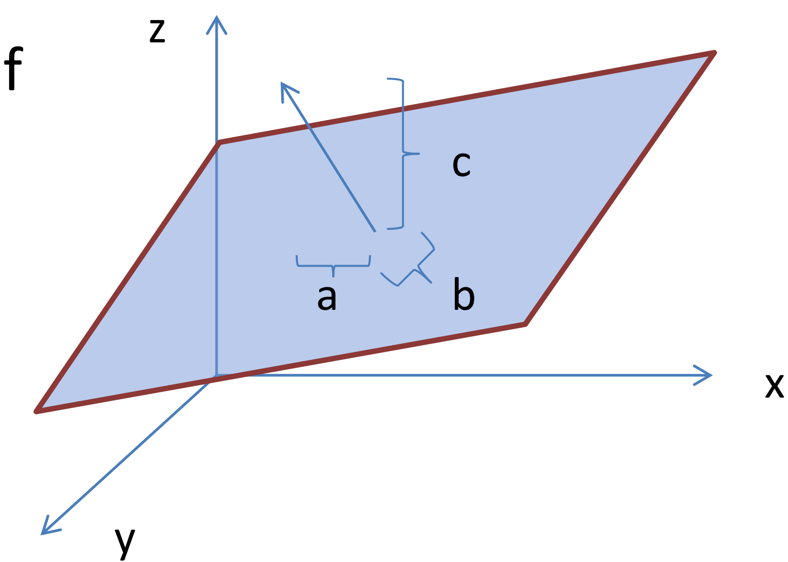
Plane Geometry

- Any line in 2D can be expressed as the set of solutions (x,y) to the equation ax+by+c=0 (an implicit line)
  - ax+by+c > 0 is one side of the line
  - ax+by+c < 0 is the other
  - ax+by+c = 0 is the line itself



Plane Geometry

- In 3D, a (hyper)plane can be expressed as the set of solutions (x,y,z) to the equation ax+by+cz+d=0
  - ax+by+cz+d > 0 is one side of the plane
  - ax+by+cz+d < 0 is the other side
  - ax+by+cz+d = 0 is the plane itself



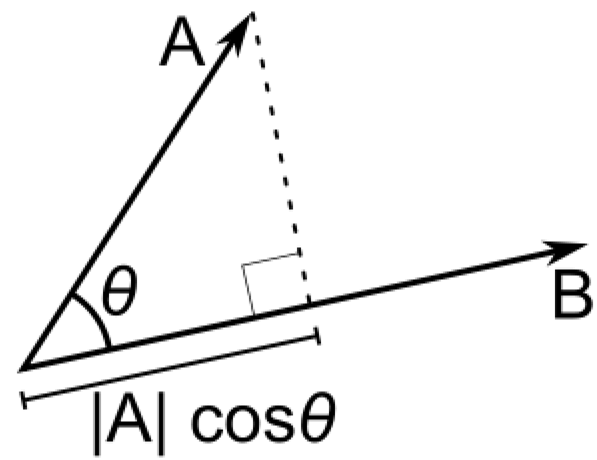
Linear Classifier

- In **d** dimensions,

  $c_0+c_1*x_1+\ldots+c_d*x_d=0$

- Abbreviate with dot product:

  $c_0+\mathbf{c}\cdot\mathbf{x}=c_0+c_1*x_1+\ldots+c_d*x_d=0$



Dot product

Describe relation between image and label

Image

$\xrightarrow{f}$ Label

Describe relation between image and label

Image

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

$$f(\mathbf{x}, \mathbf{W}) \longrightarrow$$

10 numbers defining
class scores

W
parameters
or weights

Parametric Approach: Linear Classifier

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

Image

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

$f(\mathbf{x},\mathbf{W})$

10 numbers defining
class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

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

Shape: (10,1)

Parametric Approach: Linear Classifier

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

Shape: (10,1)

Shape: (3072,1)

Parametric Approach: Linear Classifier

Shape: (10,3072)

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

Shape: (3072,1)

Shape: (10,1)

Parametric Approach: Linear Classifier

$\mathbf{w}_1 \cdot \mathbf{x} = w_{1,1} * x_1 + \ldots + w_{1,3072} * x_{3072}$

Shape: (10,3072)

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

Shape: (3072,1)

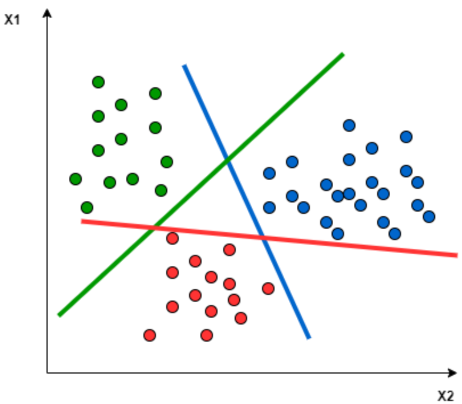
Shape: (10,1)

Parametric Approach: Linear Classifier

$\mathbf{w}_1 \cdot \mathbf{x} = w_{1,1} * x_1 + \ldots + w_{1,3072} * x_{3072}$

Shape: (10,3072)

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



Shape: (10,1)

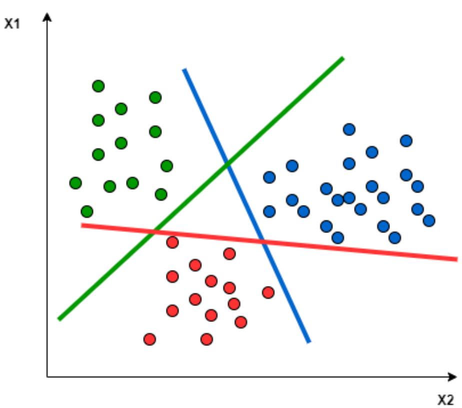
Shape: (3072,1)

Parametric Approach: Linear Classifier

$$\mathbf{w}_1 \cdot \mathbf{x} = w_{1,1} {}^* x_1 + \ldots + w_{1,3072} {}^* x_{3072}$$

Shape: (10,3072)

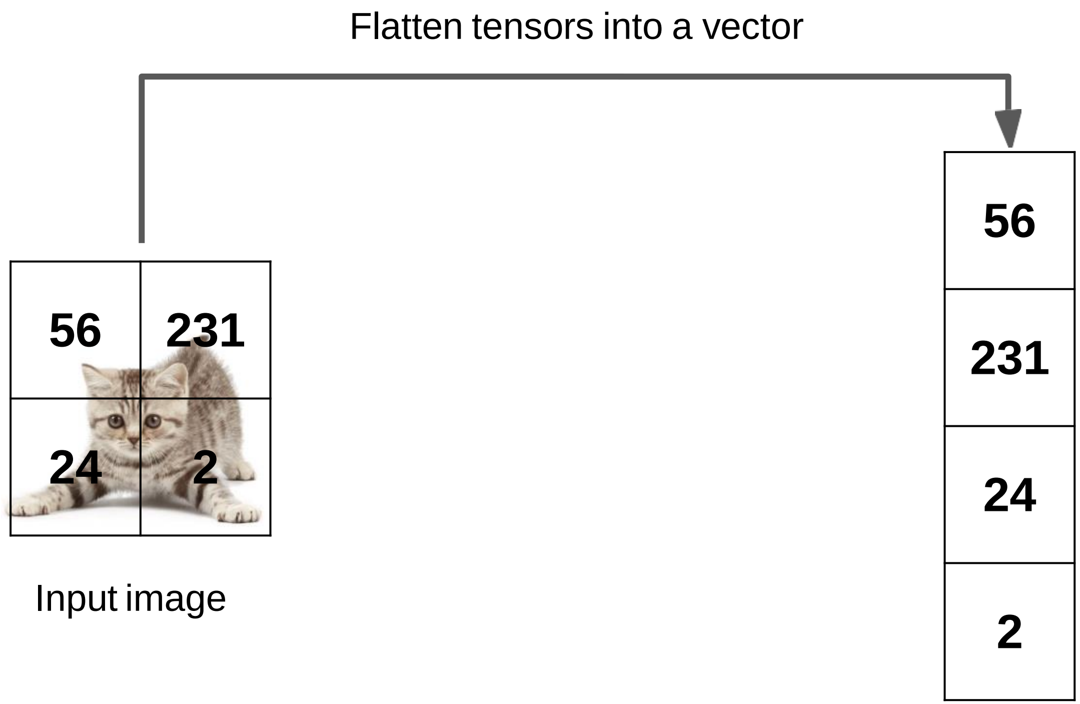
$$f(x,W) = Wx+b$$



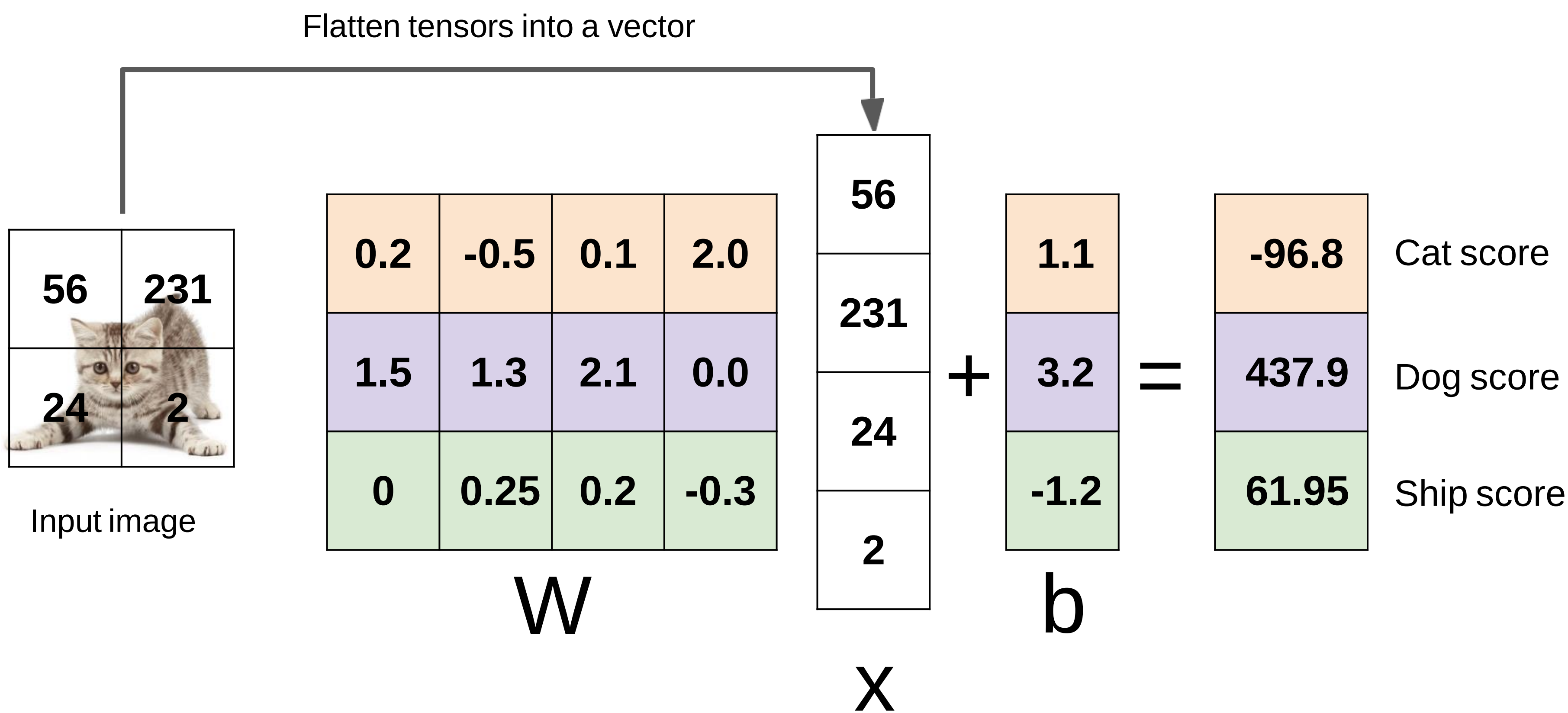
Shape: (10,1)

Shape: (3072,1)

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



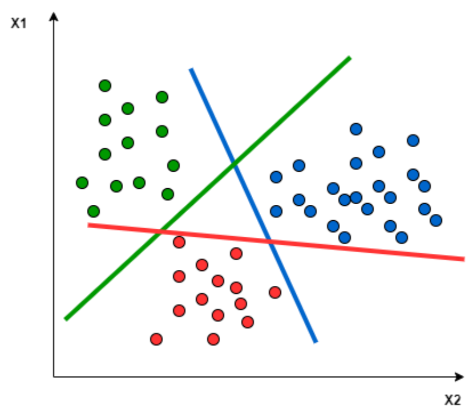
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



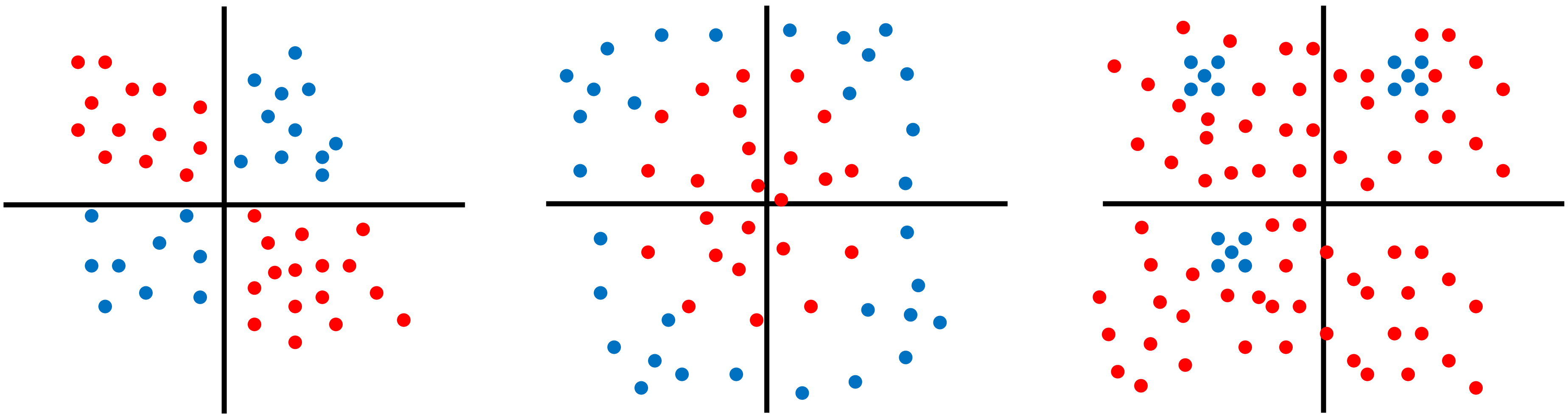
Linear Classifier Predict Efficiently

- Predict fast by generating scores with matrix-vector multiplications

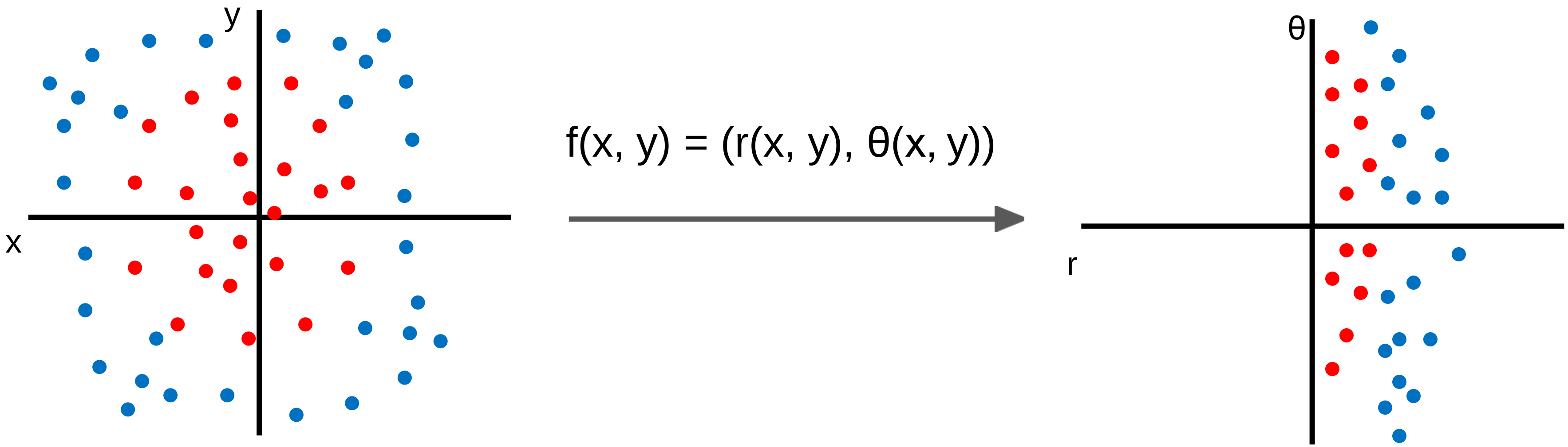
```
scores = W.dot(image) + b
```



Difficult cases for linear classifiers

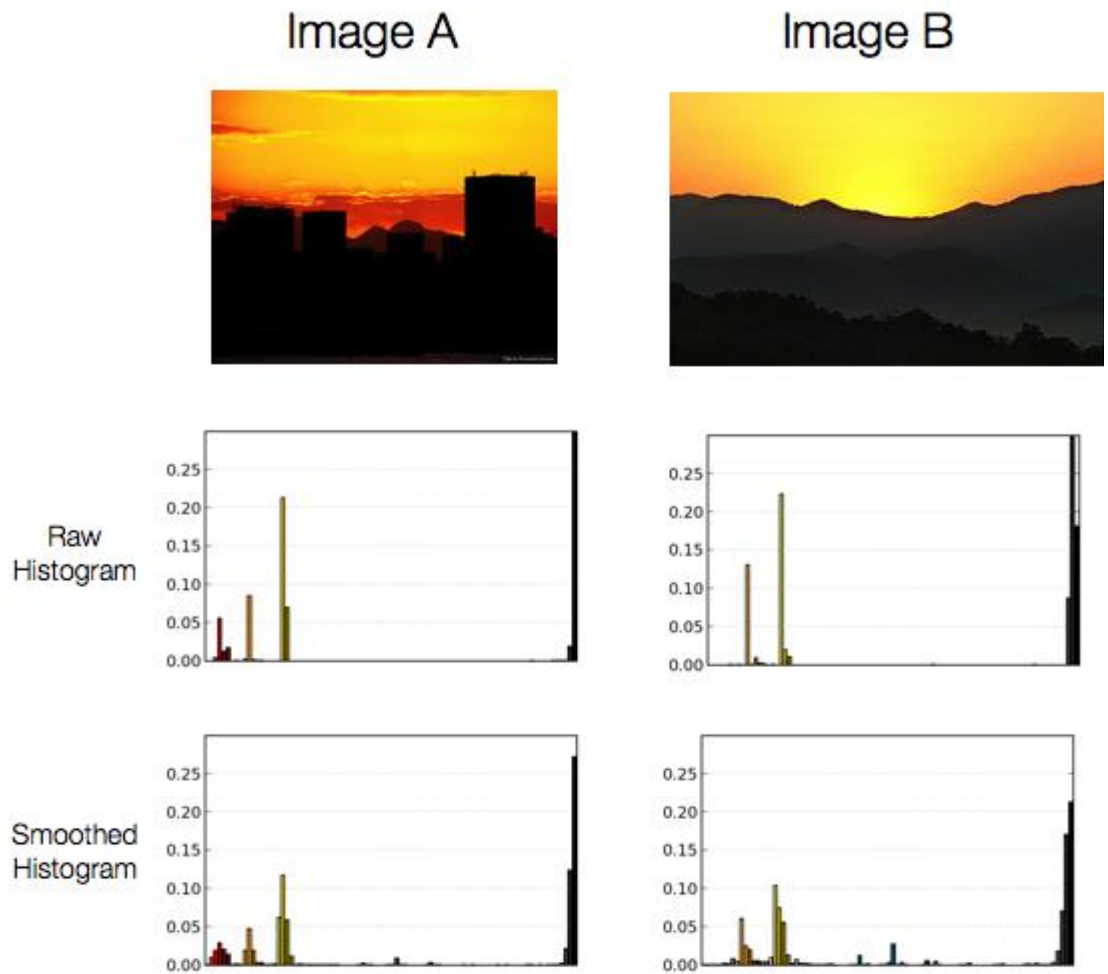


Apply Transformations

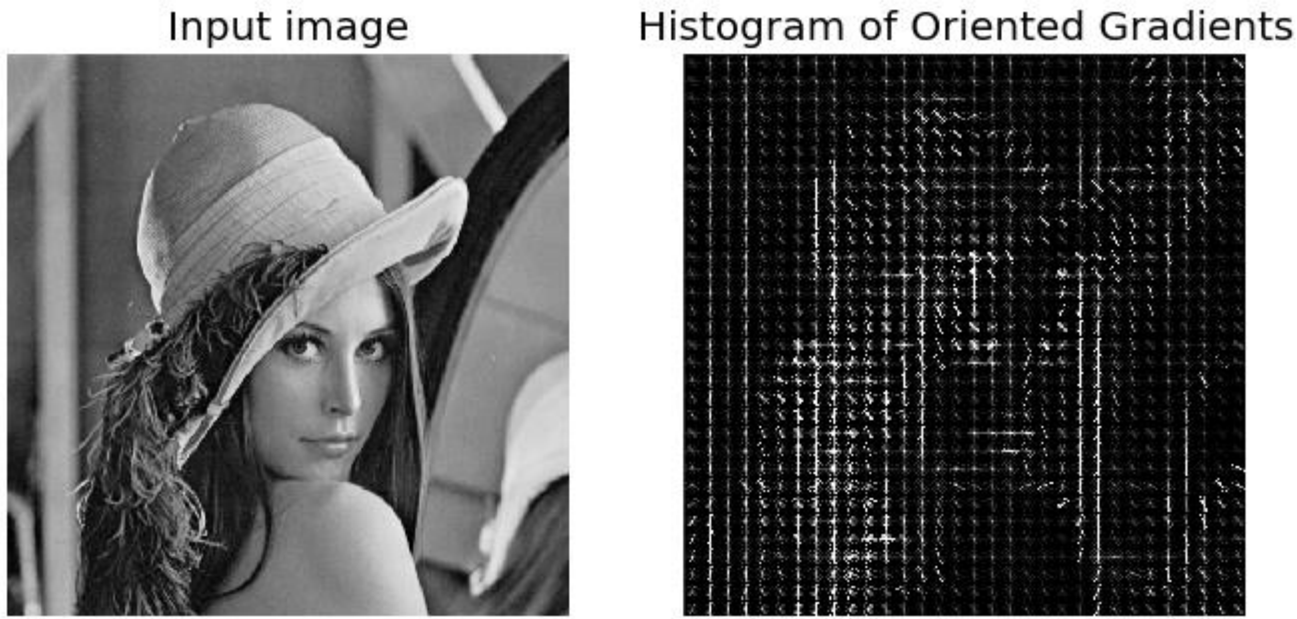


Extract features using transformations

Example: Color Histogram

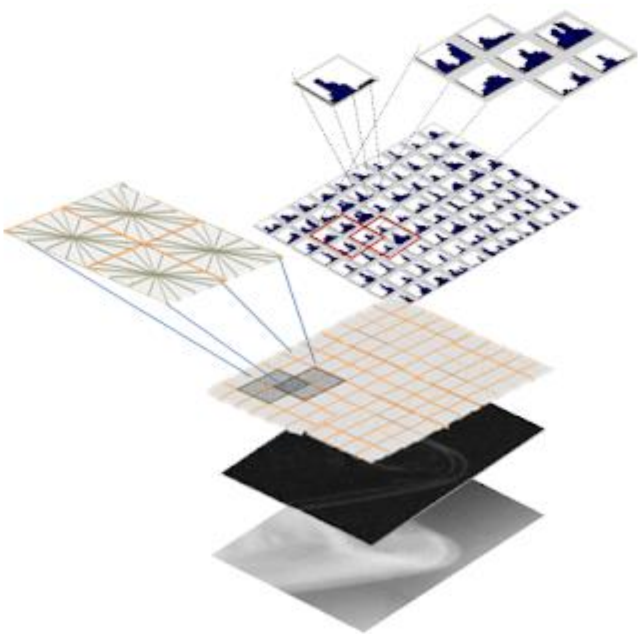


Example: Histogram of Oriented Gradients (HoG)



Lowe, “Object recognition from local scale-invariant features”, ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Example: Histogram of Oriented Gradients (HoG)



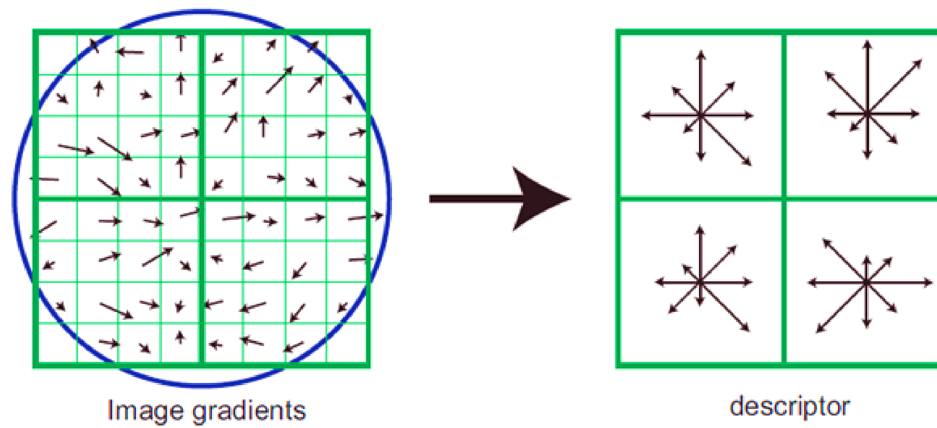
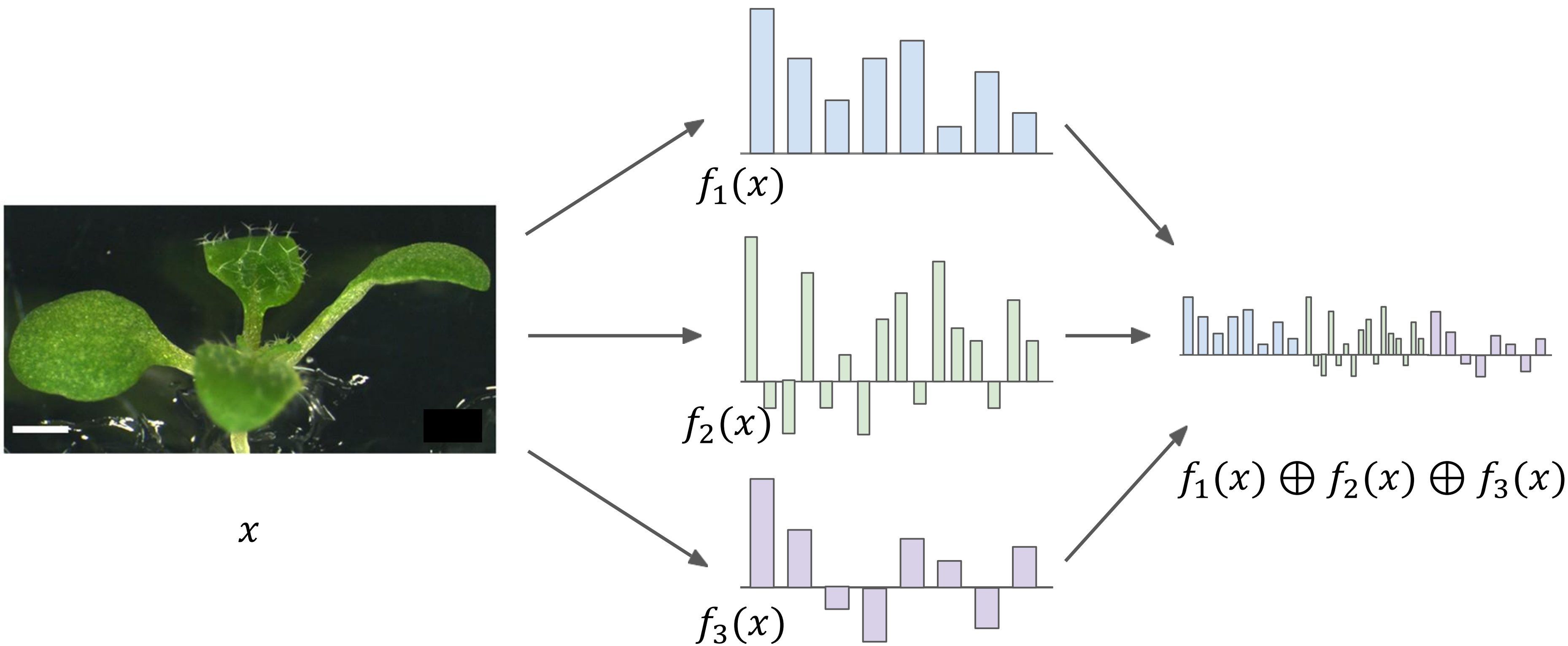
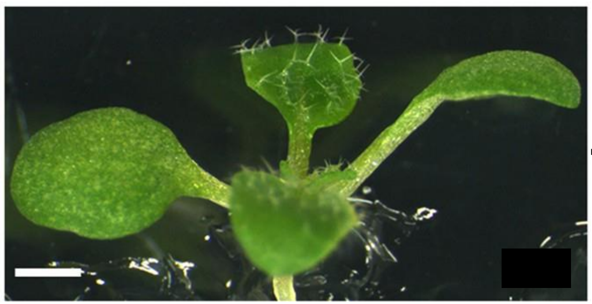


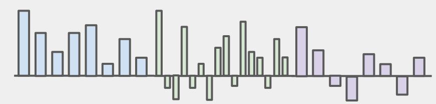
Image Feature Aggregation



Classification on Image Features



Feature Extraction



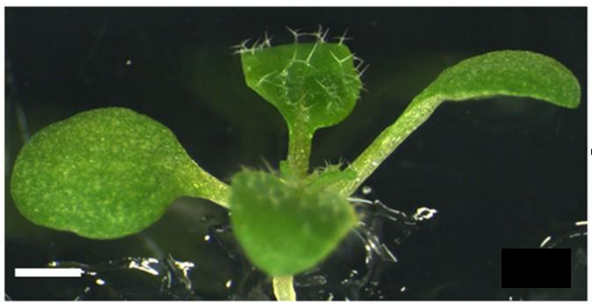
+

Class Label

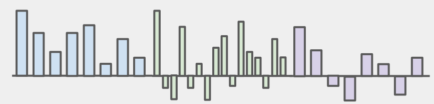
f

10 numbers giving scores for classes

Classification on Image Features



Feature Extraction



+

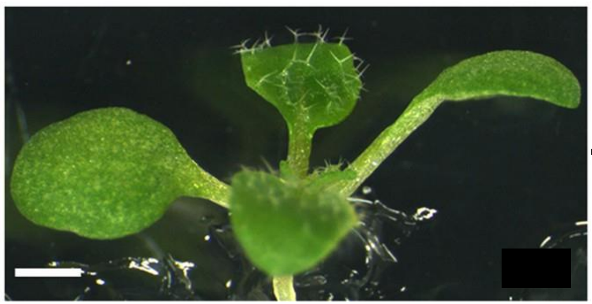
Class Label

W

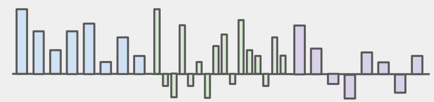
f

10 numbers giving scores for classes

Classification on Image Features



Feature Extraction



+

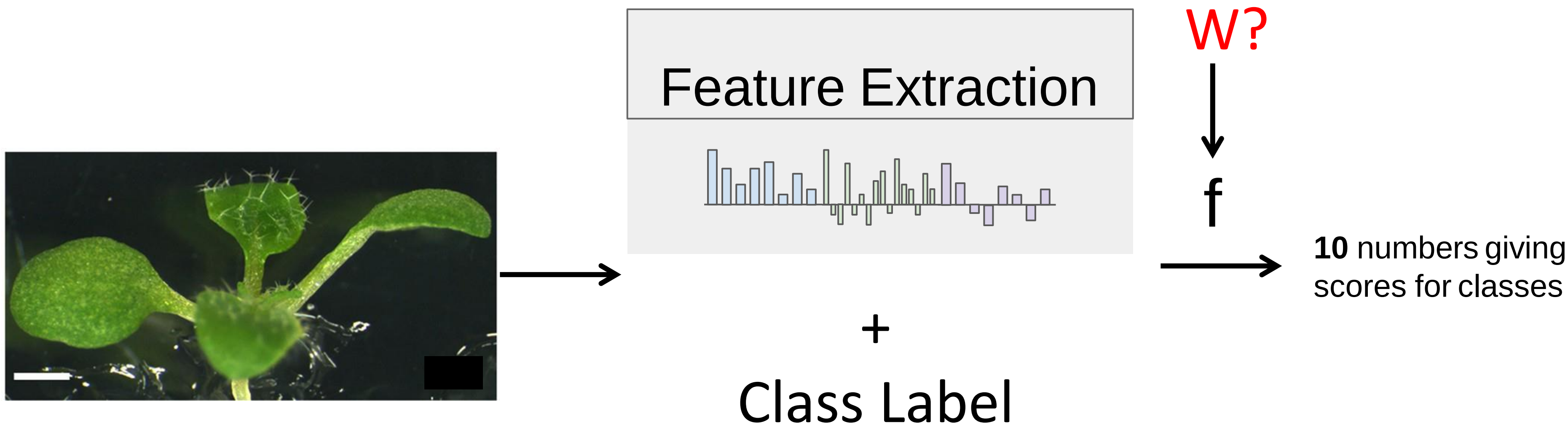
Class Label

W?

f

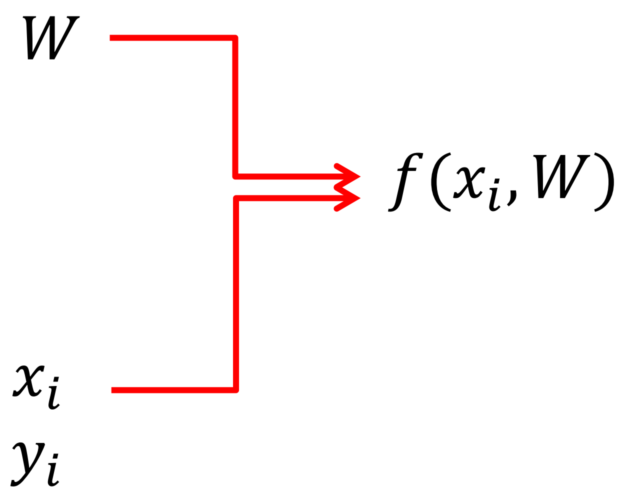
10 numbers giving scores for classes

Classification on Image Features

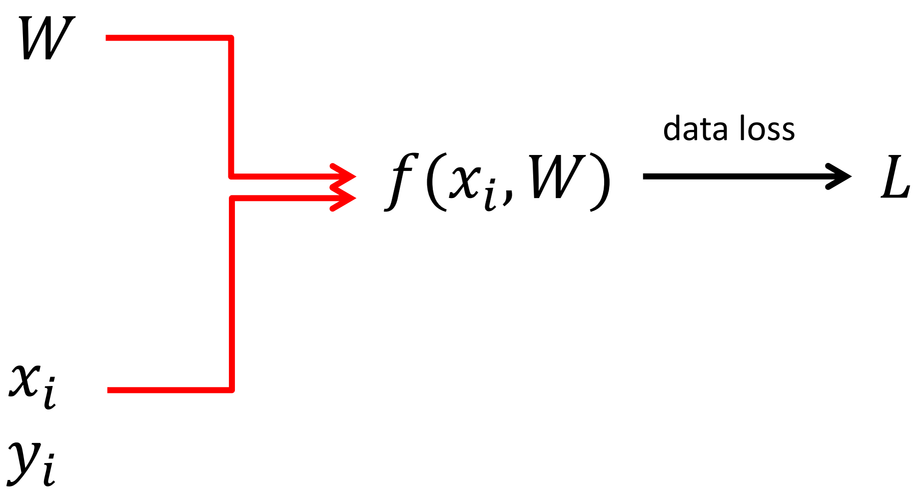


Measure how well a set of values for W classifies an input

How expressive are the values of W?

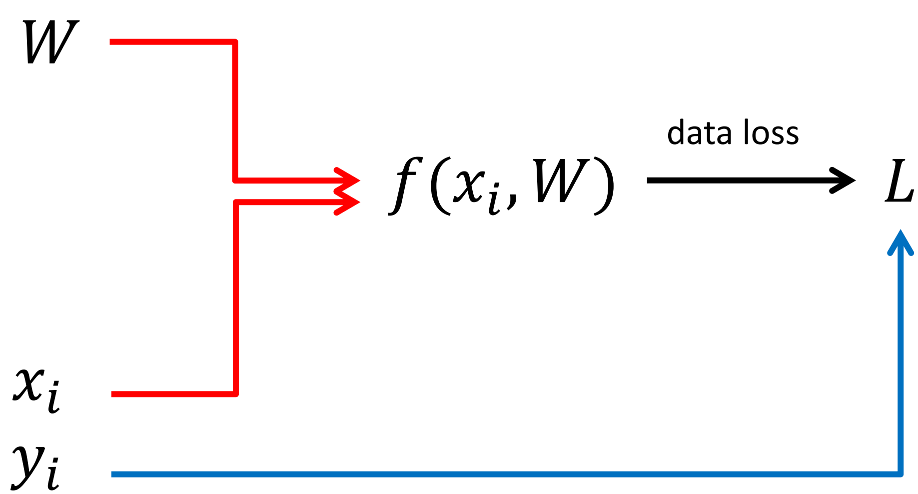


How expressive are the values of W?



L: Metric to assess what loss of data classification our model incurs

Loss Function



L: Metric to assess what loss of data classification our model incurs