Lecture 1

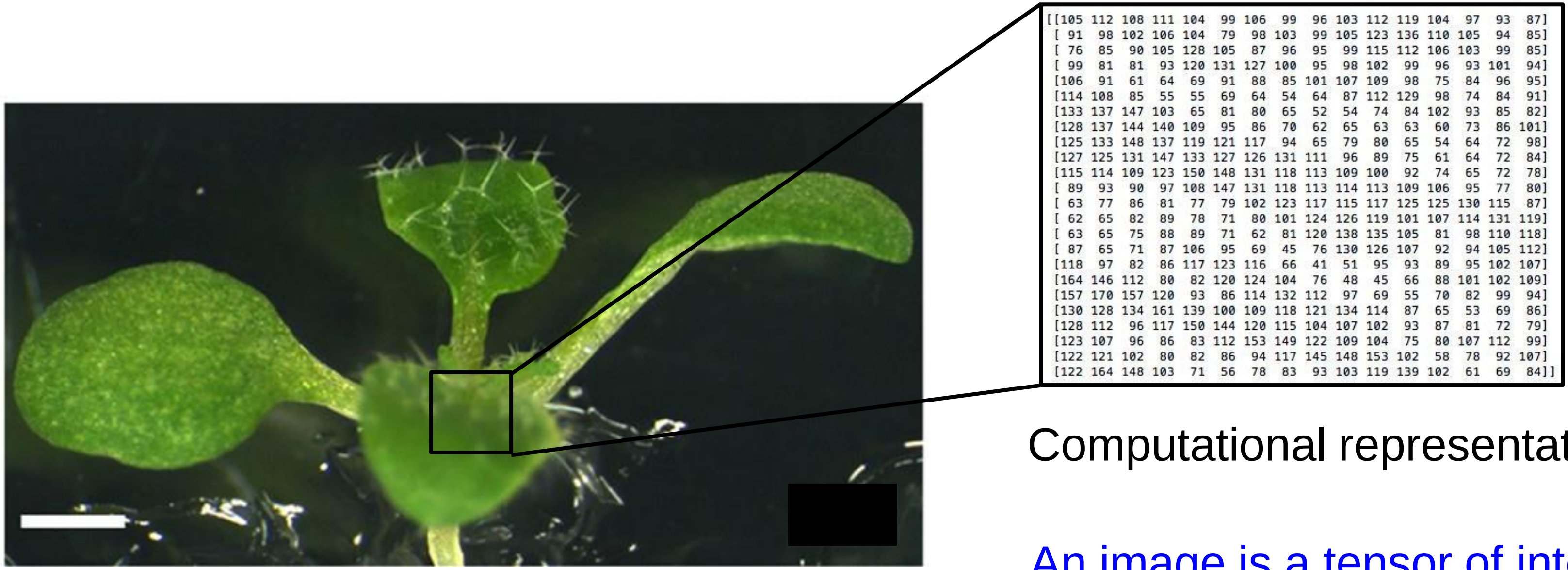
Image Classification

[redacted]

→ plant

Select correct class from a given set of classes

Image Classification



[redacted]

```
[[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


[redacted]
[redacted]
[redacted]
[redacted]

Pixel values change when the camera moves.

Challenges: Different Backgrounds

Challenges: Different Illumination

Challenges: Occlusion

Challenges: Variation

Image Classifier

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

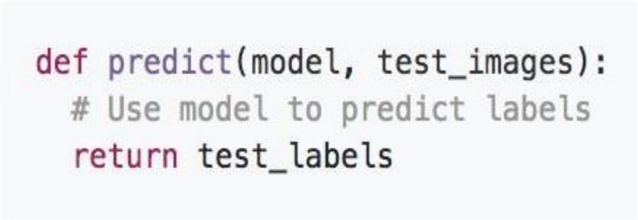
There is no deterministic, trivial way of selecting correct classes given just an input image

Rule-based Methods

Machine Learning: Data-Driven Approach

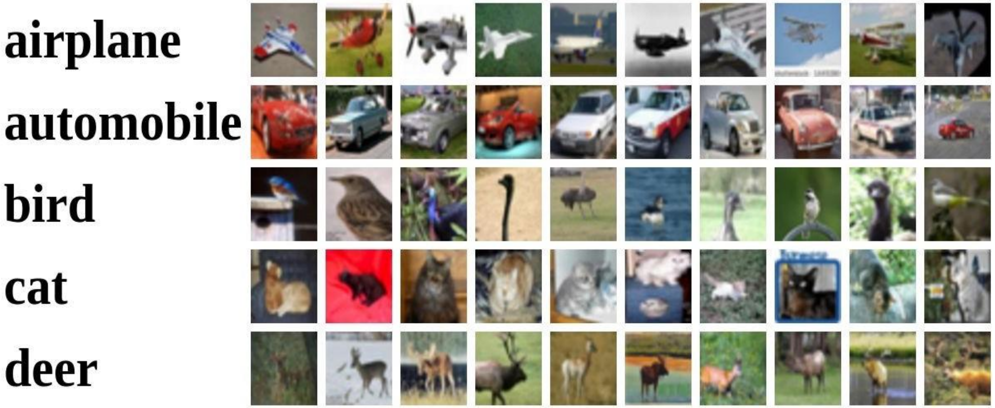
1. Collect a dataset of images and labels
2. Use Machine Learning algorithms to train a classifier
3. Evaluate the classifier on new images

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



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

Example training set

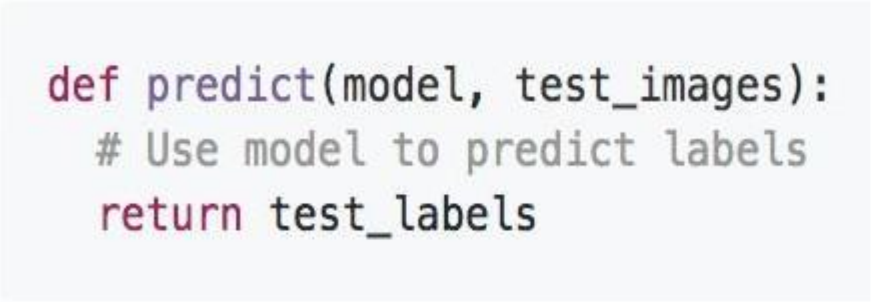


Nearest Neighbor Classifier

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**

Training data with labels

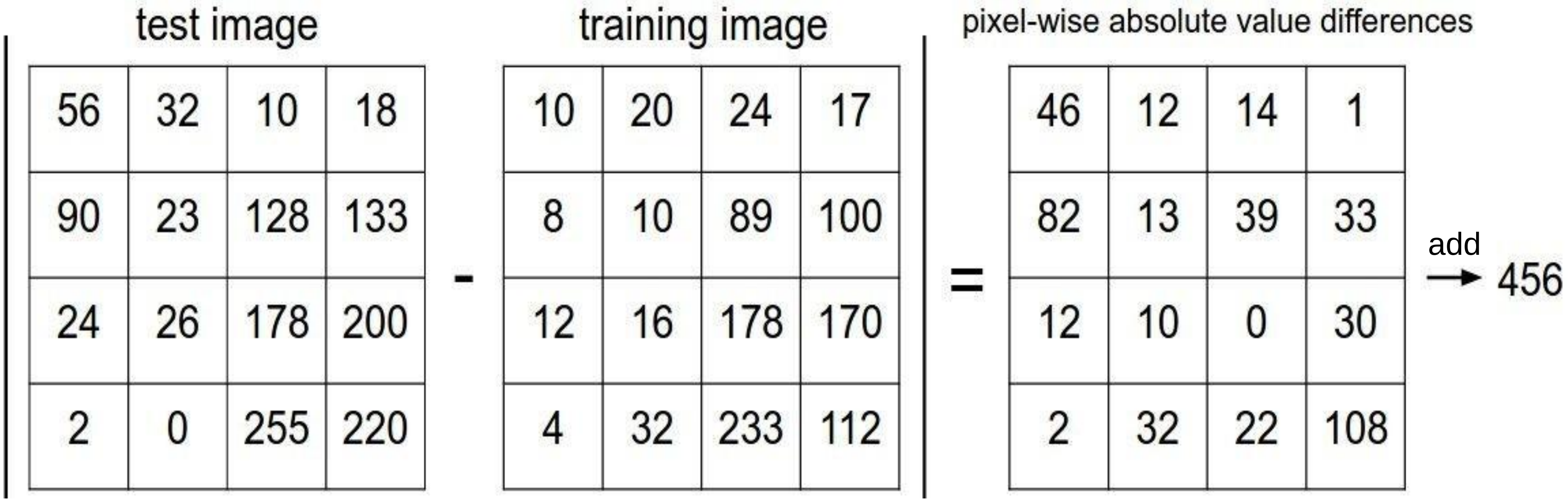
query data

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

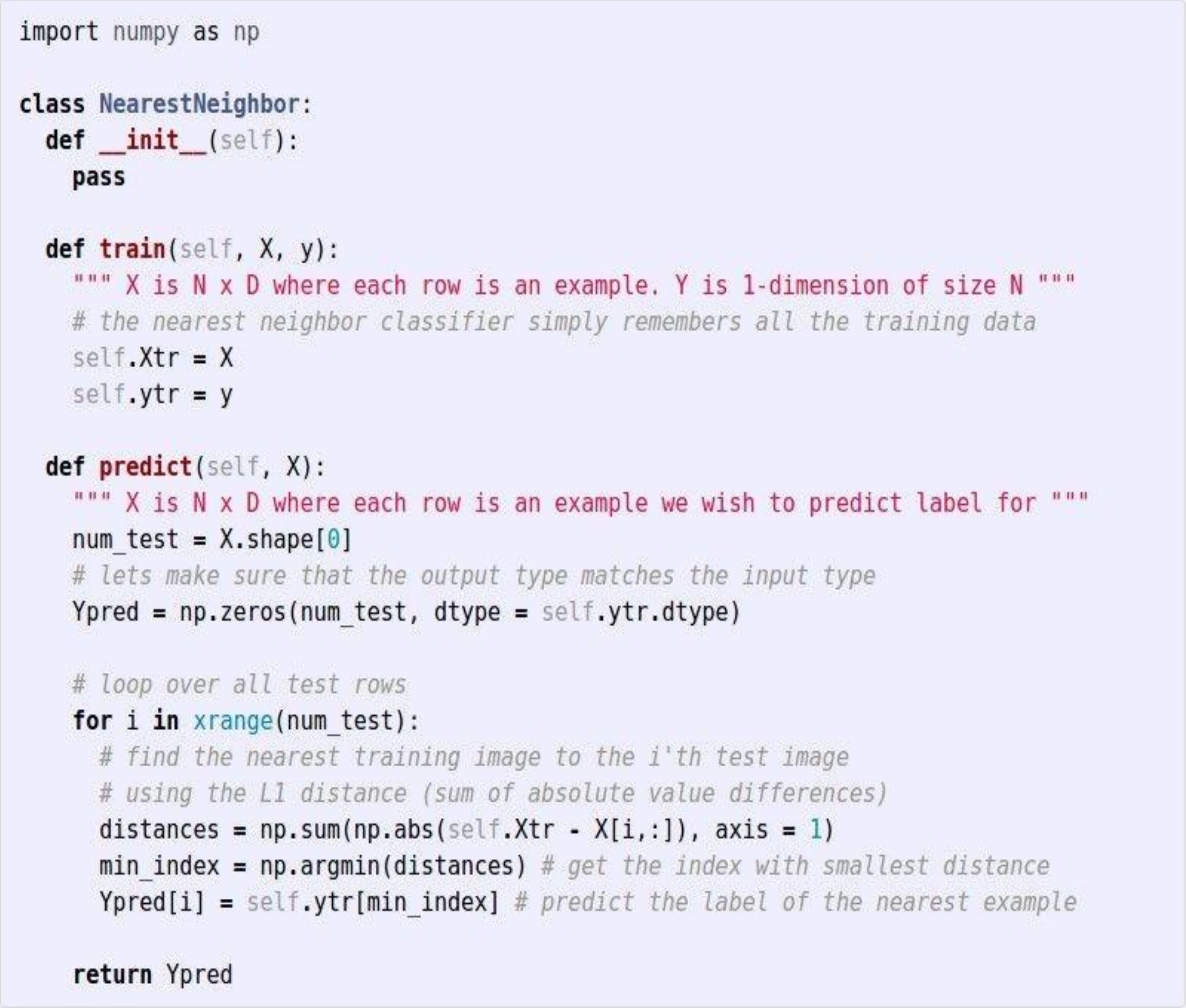
$=$

pixel-wise absolute value differences

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

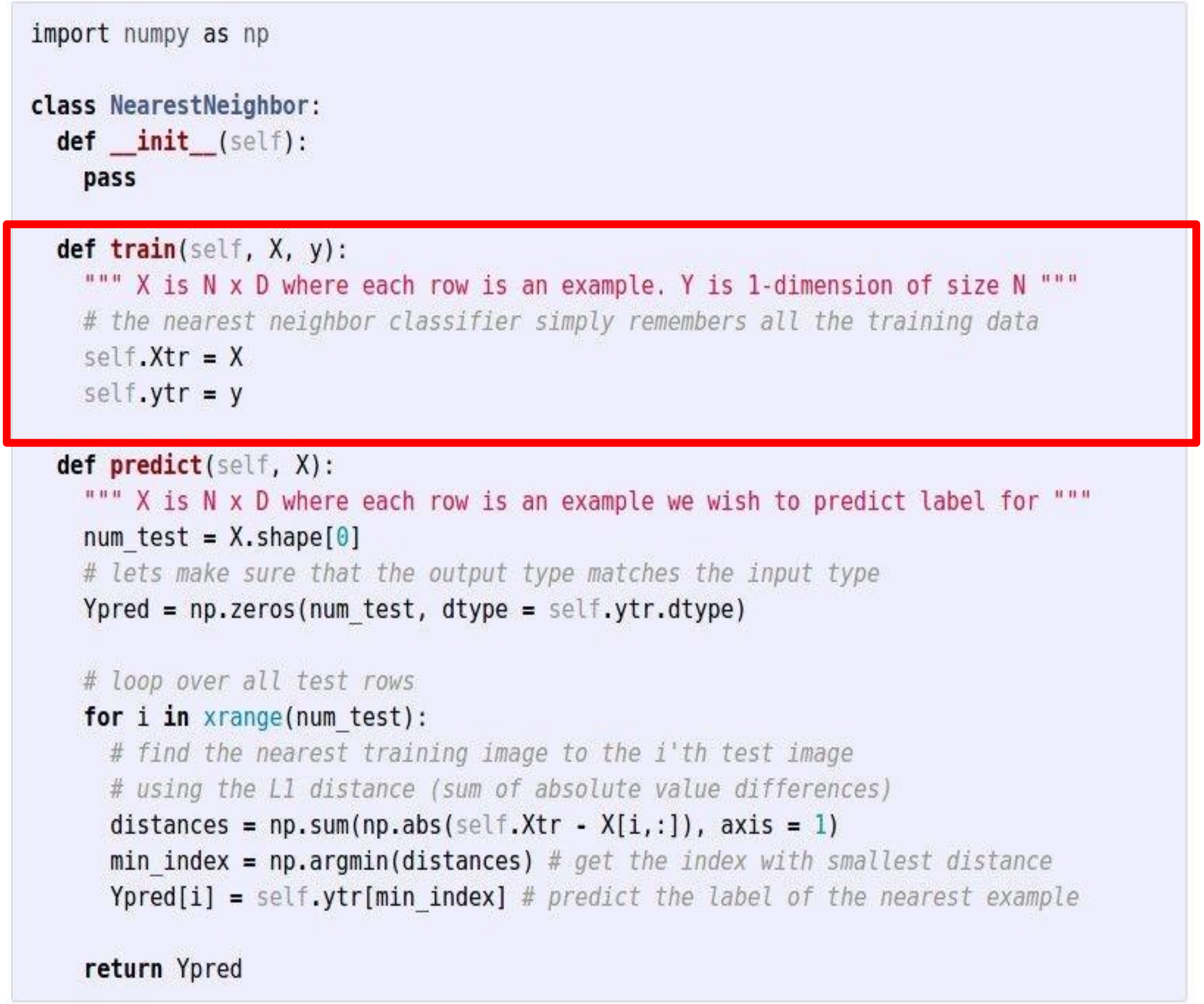
$\xrightarrow{\text{add}}$ 456

Nearest Neighbor classifier



```
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
```

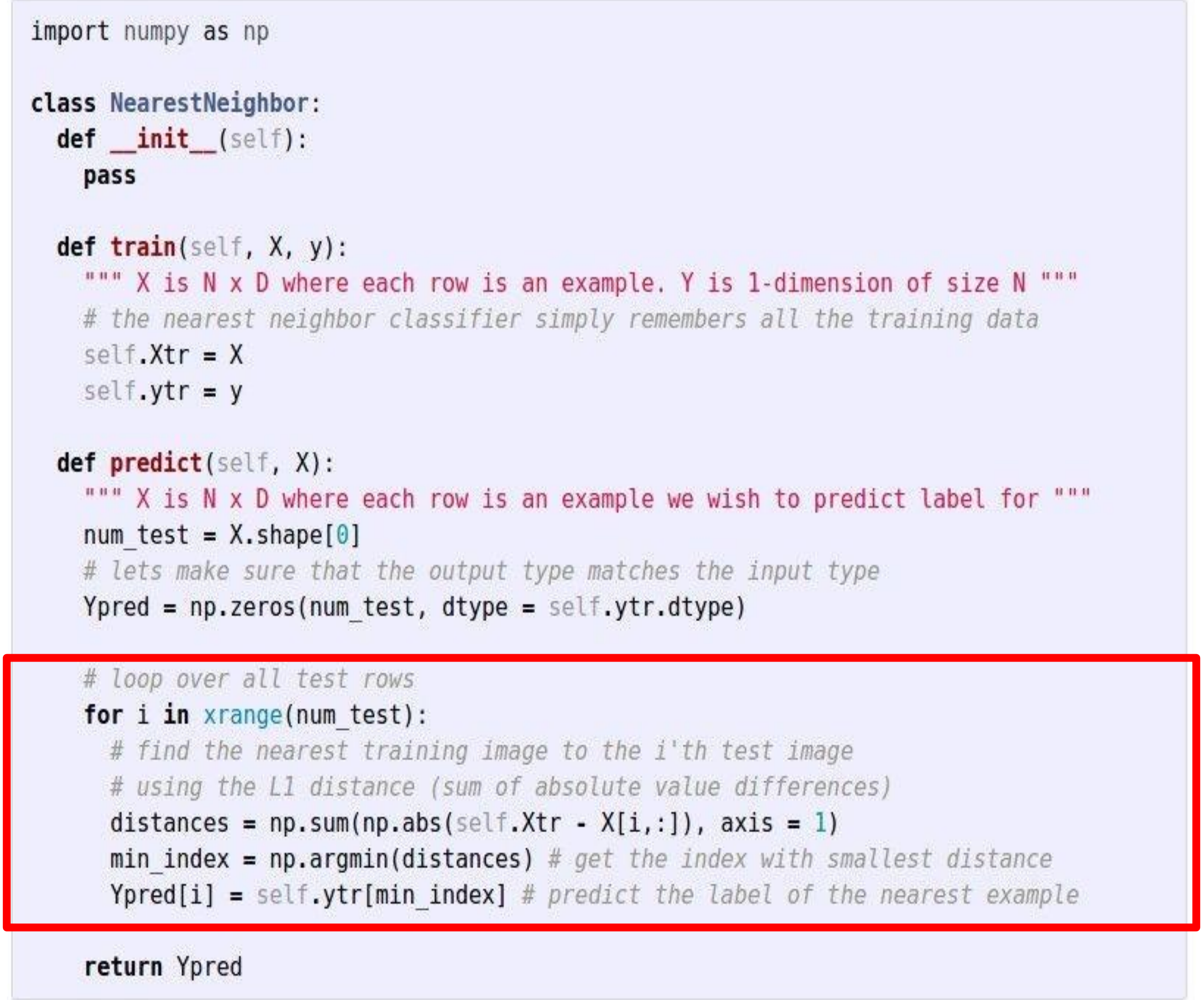
Nearest Neighbor classifier



```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

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

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

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

    return Ypred
```

Memorize training data

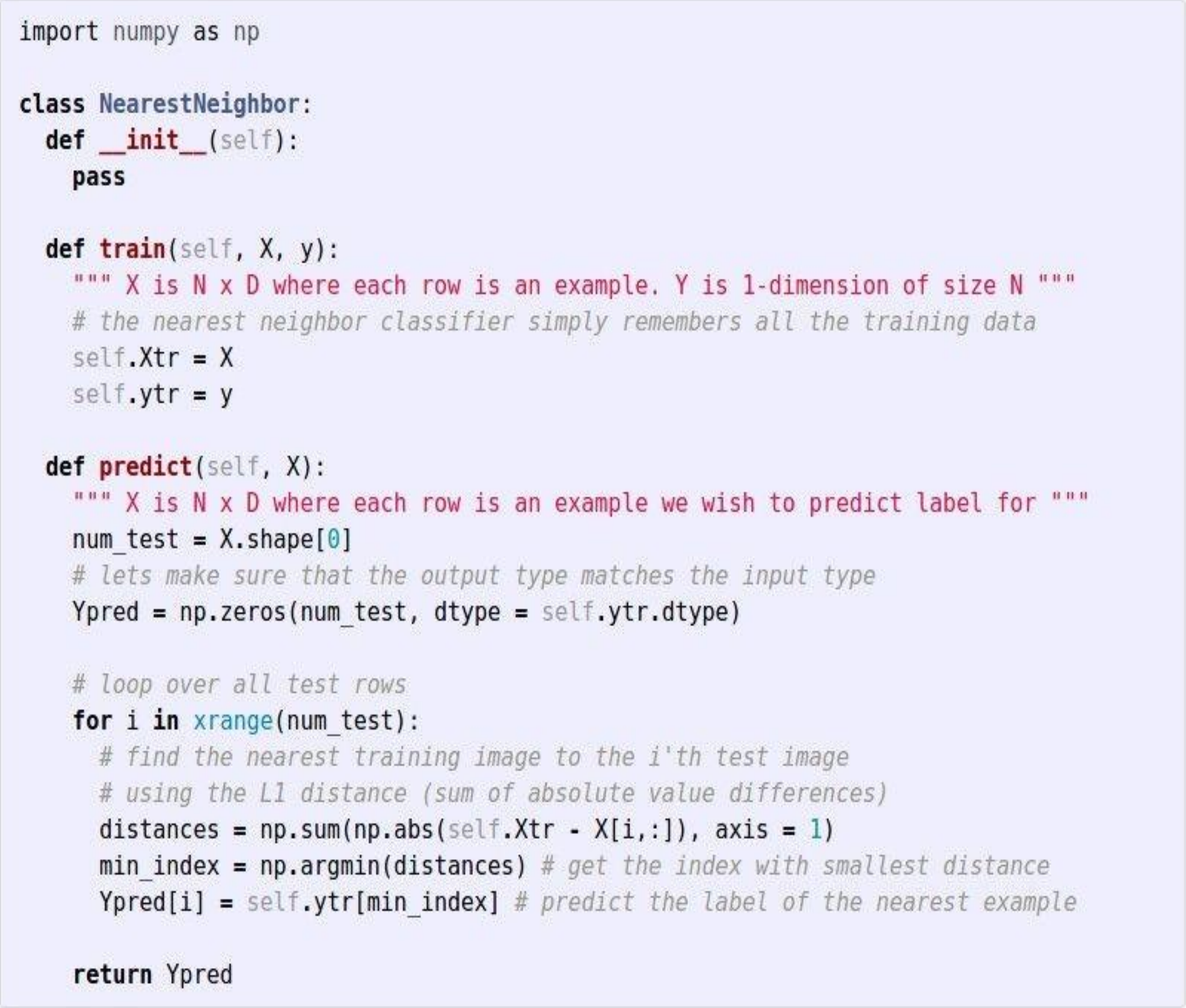
Nearest Neighbor classifier



```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

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

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

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

    return Ypred
```

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

Nearest Neighbor classifier

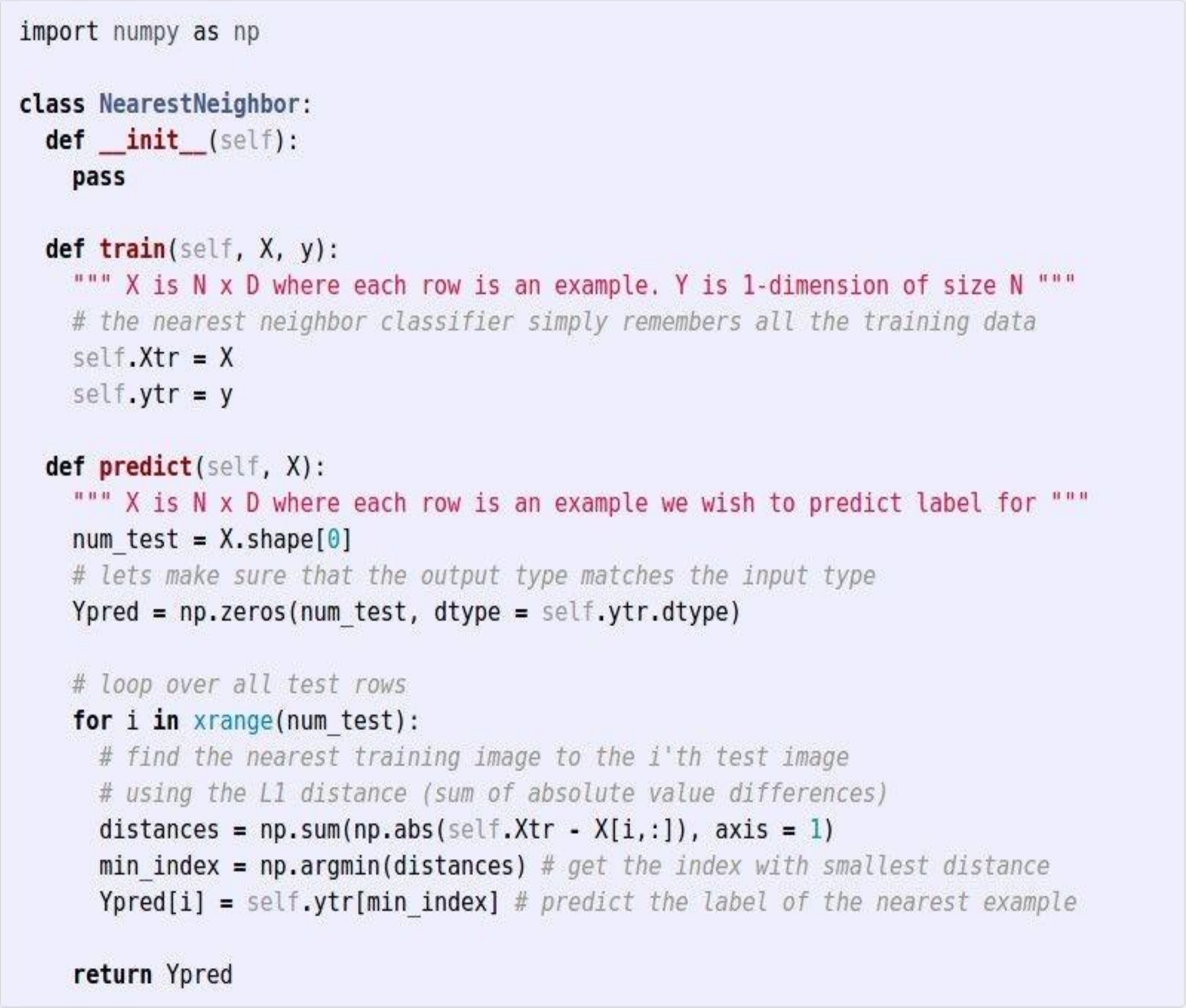


```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

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

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

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

    return Ypred
```

Q: With N examples, how fast are training and prediction?

Ans: Train O(1),
predict O(N)

This is bad: we want classifiers that are **fast** at prediction; **slow** for training is ok



```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
```

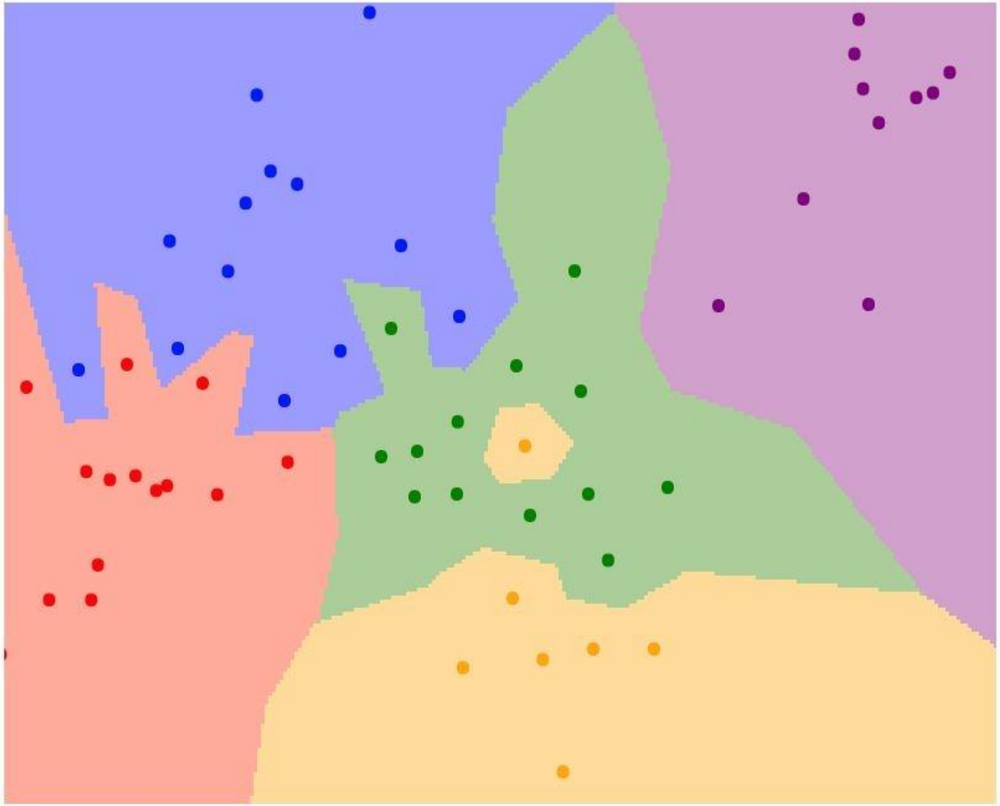
Nearest Neighbor classifier

Many methods exist for fast Nearest Neighbor

A good implementation:
https://github.com/facebookresearch/faiss

Johnson et al, “Billion-scale similarity search with GPUs”, arXiv 2017

Example

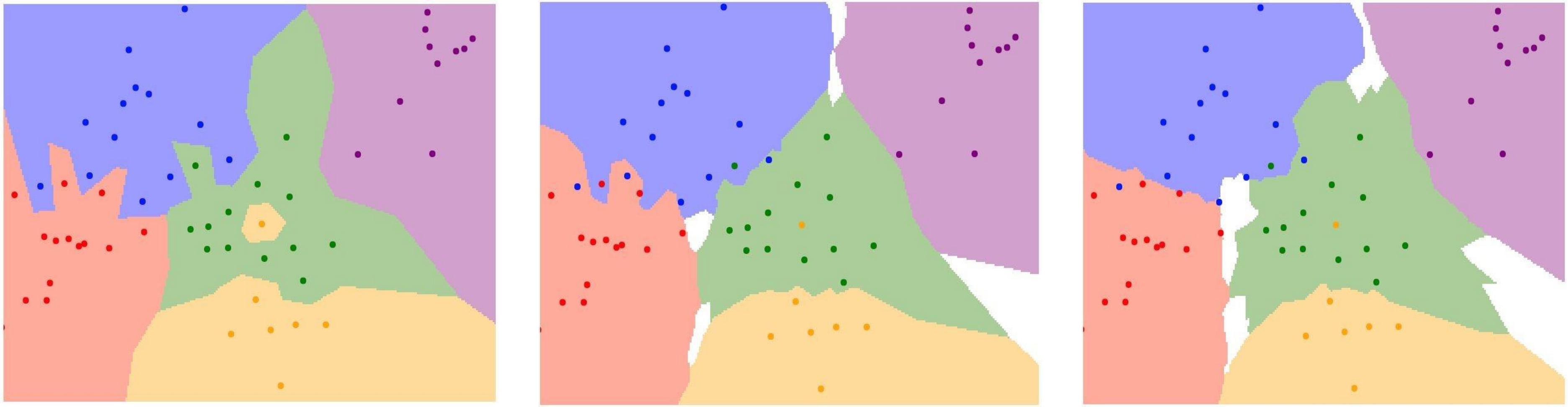


1-nearest neighbor

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

K-Nearest Neighbors

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



K = 1

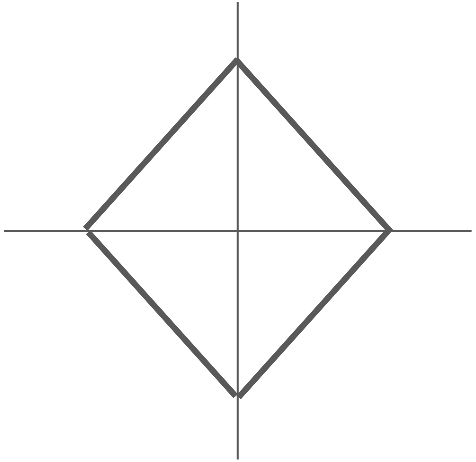
K = 3

K = 5

K-Nearest Neighbors: Distance Metric

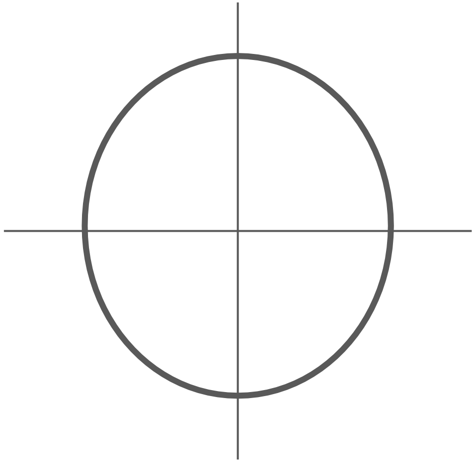
L1 (Manhattan) distance

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



L2 (Euclidean) distance

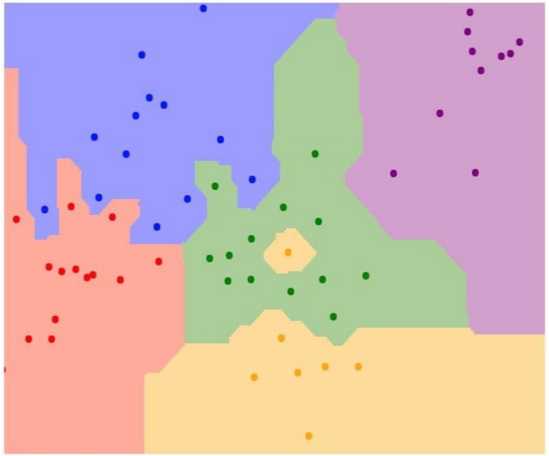
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

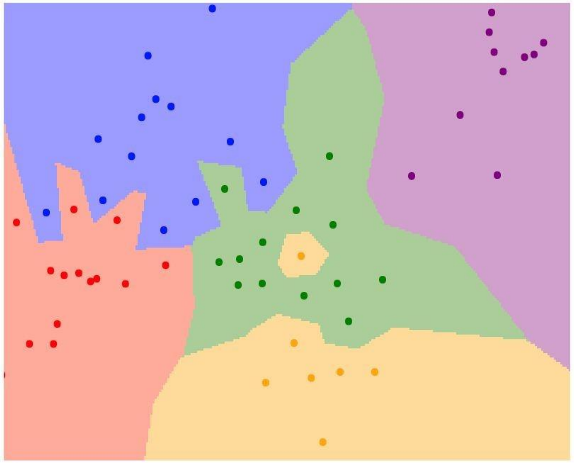
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



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

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.

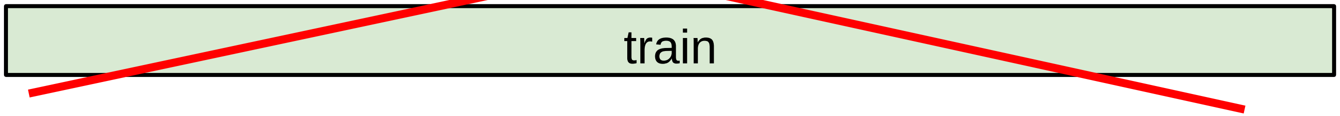
Setting Hyperparameters

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

train

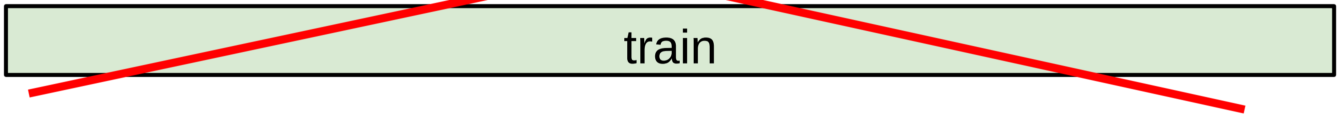
Setting Hyperparameters

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



Setting Hyperparameters

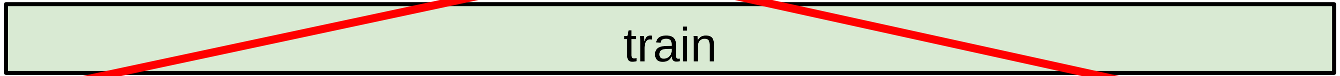
~~**Idea #1**: Choose hyperparameters~~
~~that work best on the **training data**~~



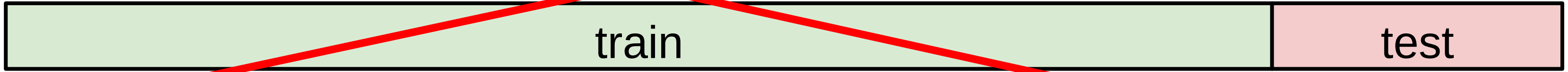
Idea #2: choose hyperparameters
that work best on **test** data

Setting Hyperparameters

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

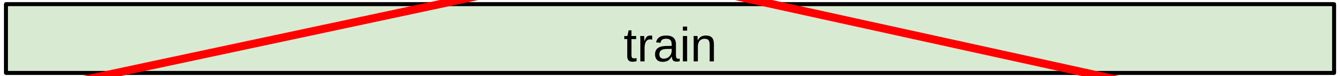

train

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

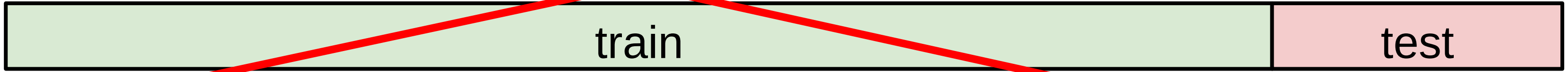

train
test

Setting Hyperparameters

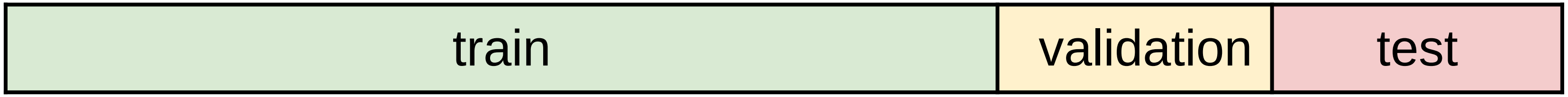
~~**Idea #1**: Choose hyperparameters that work best on the **training data**~~



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



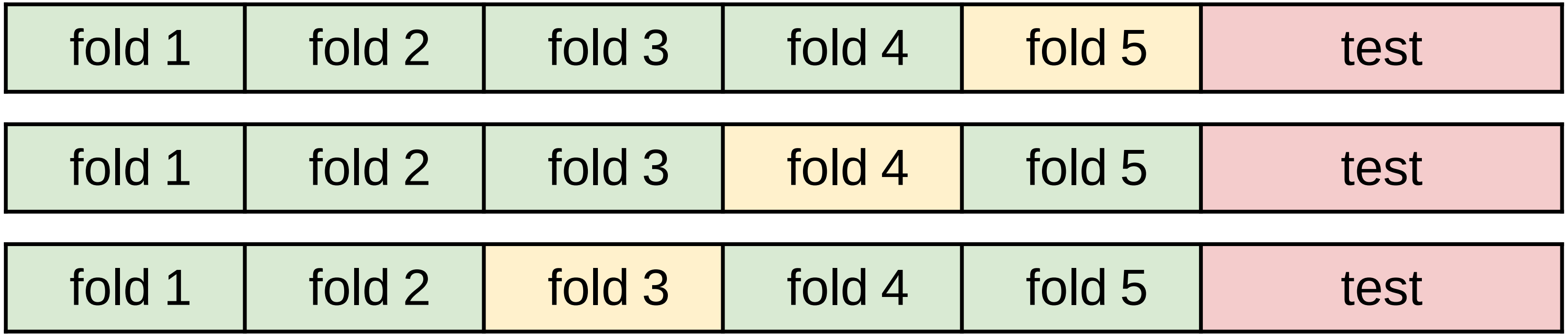
Idea #3: Split data into **train**, **val**; choose hyperparameters on val and evaluate on test



Setting Hyperparameters

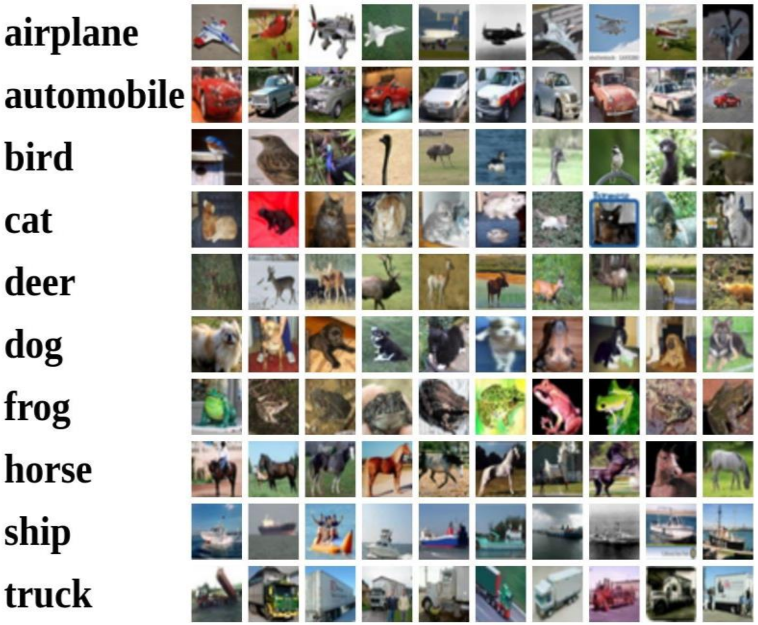
train

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



Example Dataset: **CIFAR10**

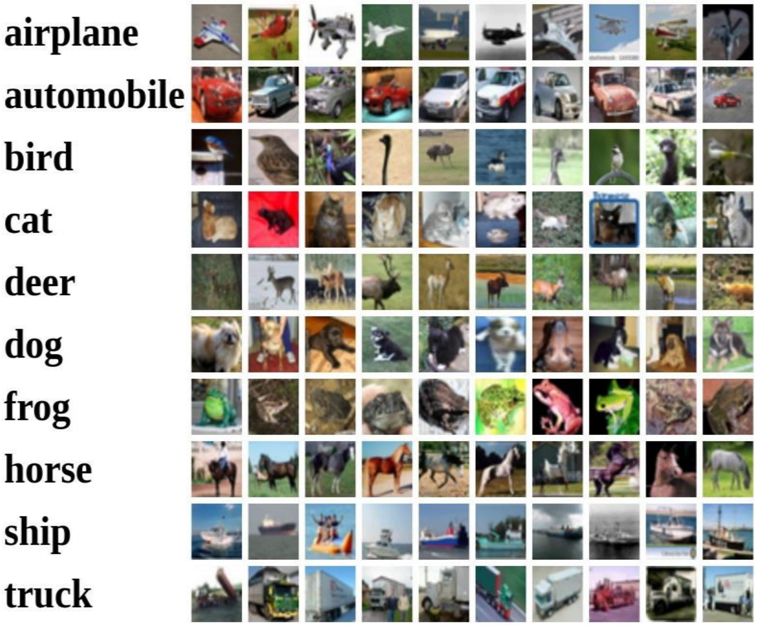
10 classes
50,000 training images
10,000 testing images



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

Example Dataset: **CIFAR10**

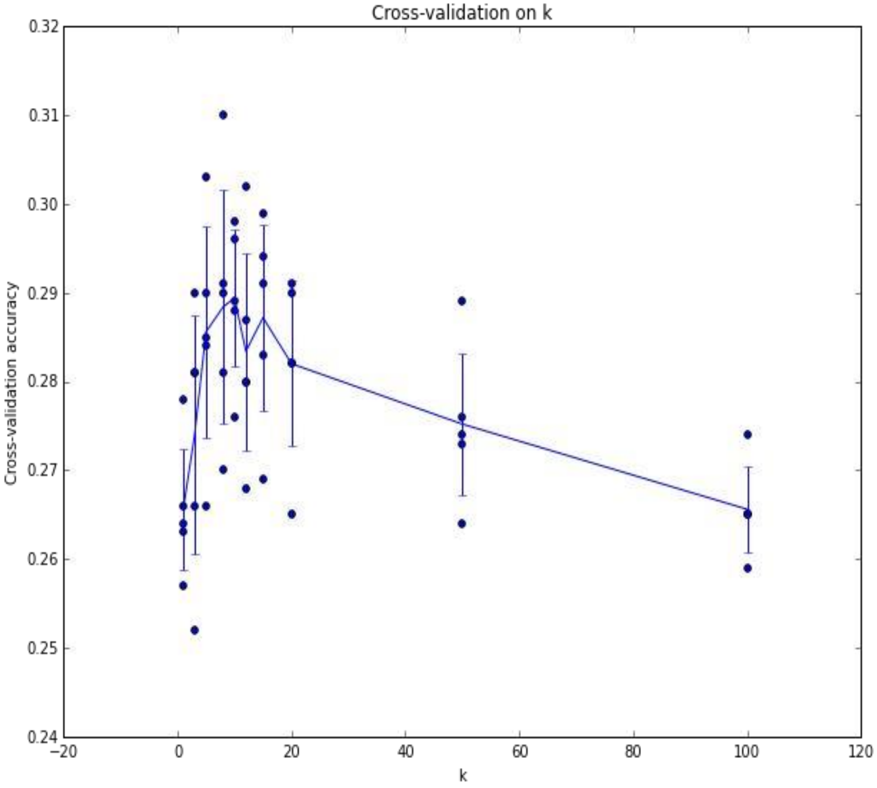
10 classes
50,000 training images
10,000 testing images



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.**

Points: single prediction outcomes

Line: mean

Bars: standard deviation

k ~= 7 achieves best performance

kNN Results

kNN Results

K-Nearest Neighbors Summary

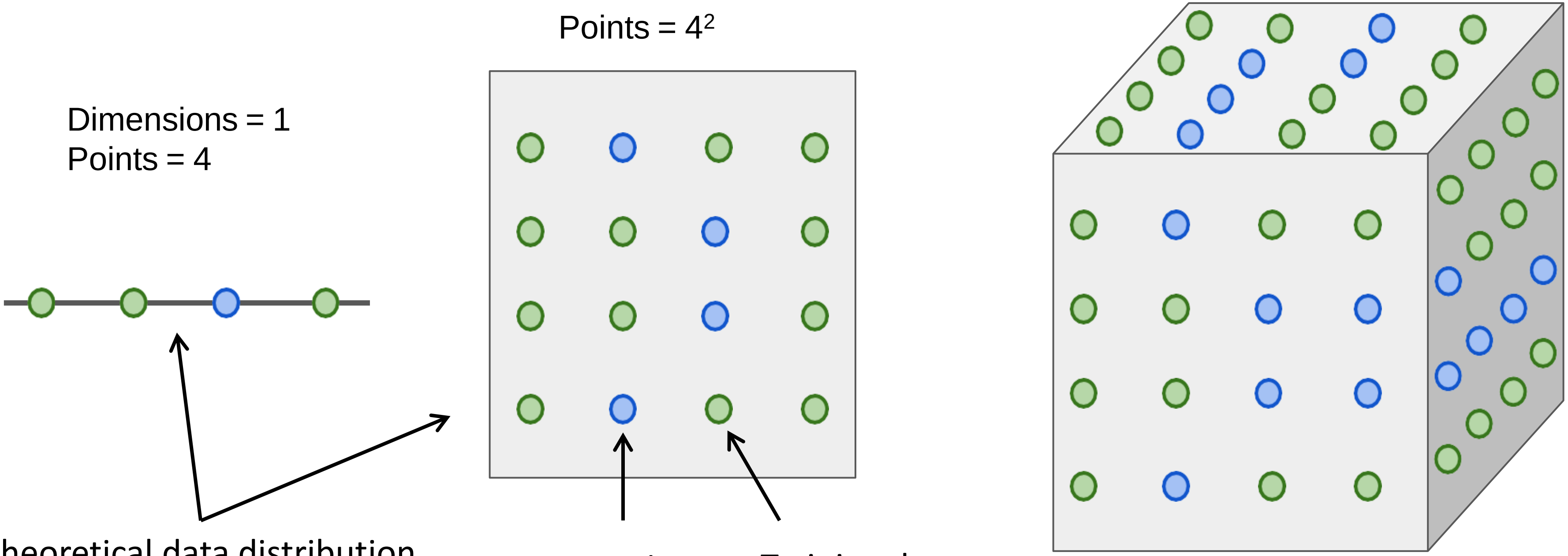
Image classification requires a **training set** of images and labels. It predicts labels on a **test set.**

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**

Spatial Coverage Needs Increases with Dimension



Dimensions = 1
Points = 4

Dimensions = 2
Points = 4^2

Dimensions = 3
Points = 4^3

Theoretical data distribution

Training data class A

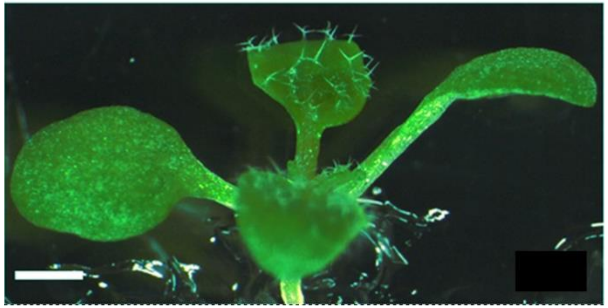
Training data class B

k-Nearest Neighbor Drawbacks

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

Original

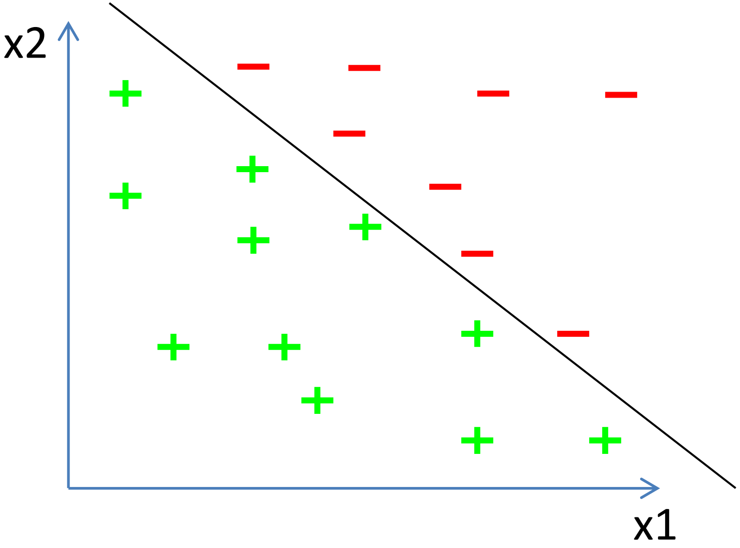
Tinted



Linear Classifier

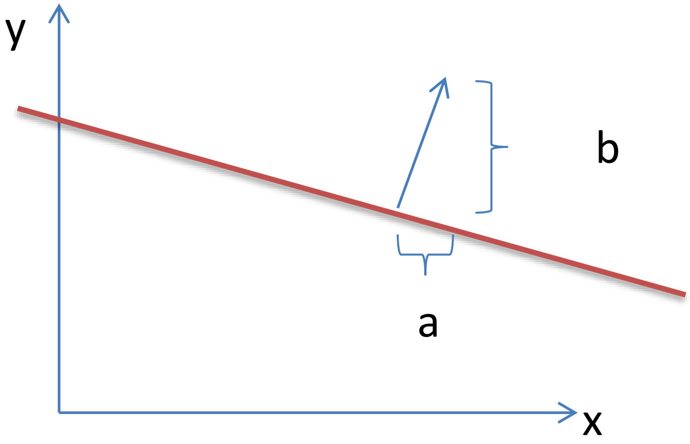
Linear classifiers : Motivation

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



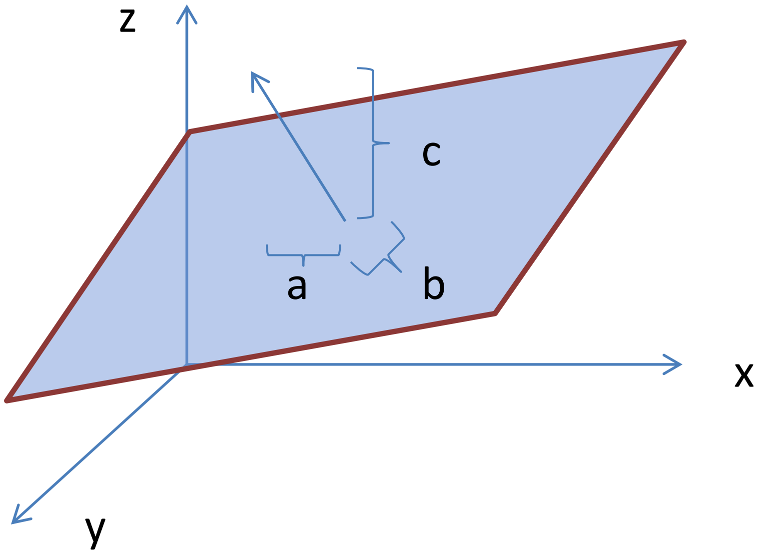
Plane Geometry

- 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



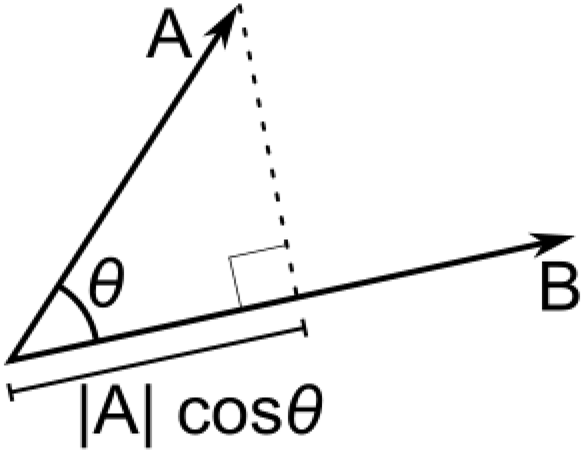
Linear Classifier

- 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)

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

10 numbers defining
class scores

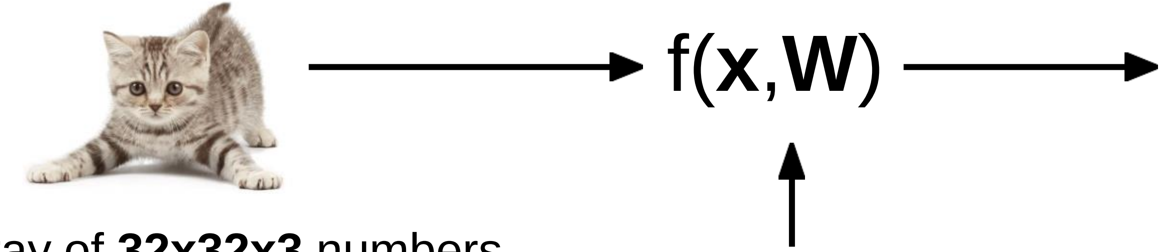
$$\uparrow$$

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 defining
class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

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

Shape: (10,1)

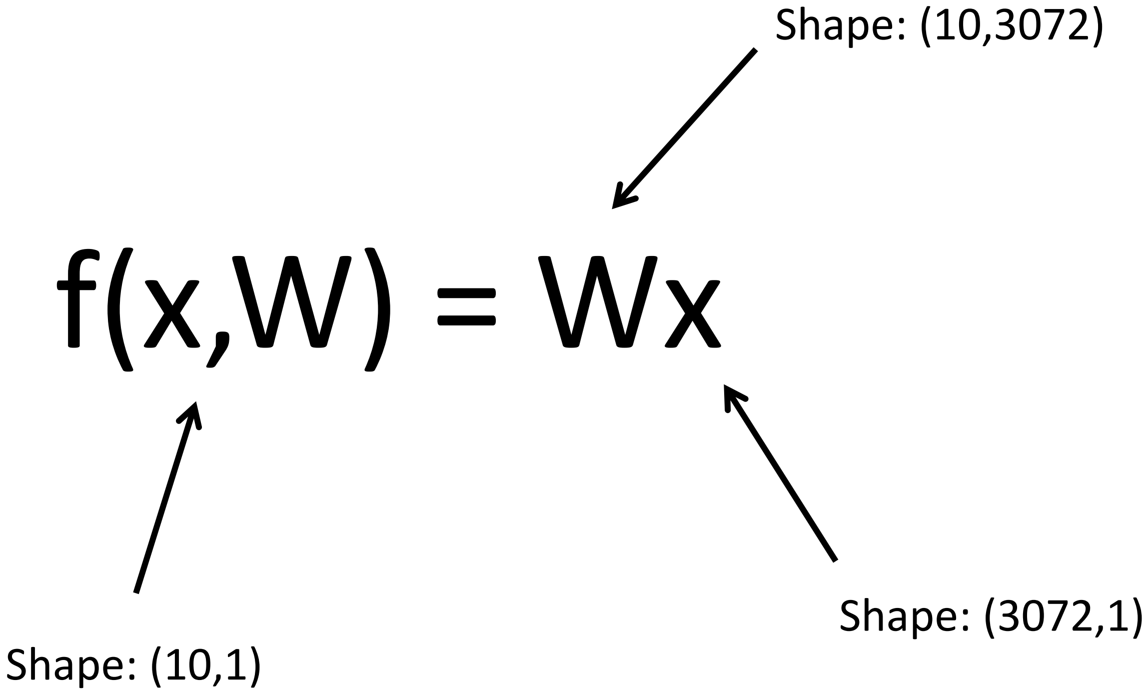
Parametric Approach: Linear Classifier

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

Shape: (3072,1)

Shape: (10,1)

Parametric Approach: Linear Classifier



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)

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

Shape: (3072,1)

Shape: (10,1)

Parametric Approach: Linear Classifier

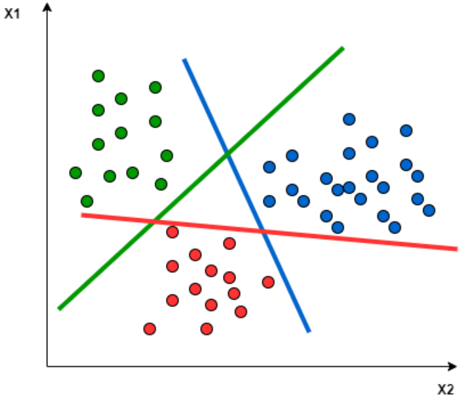
$\mathbf{w}_1\cdot\mathbf{x}=w_{1,1}{}^{*}x_1+\ldots+w_{1,3072}{}^{*}x_{3072}$

Shape: (10,3072)

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

Shape: (10,1)

Shape: (3072,1)

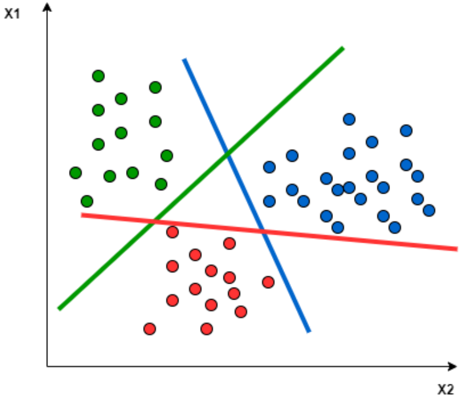


Parametric Approach: Linear Classifier

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

Shape: (10,3072)

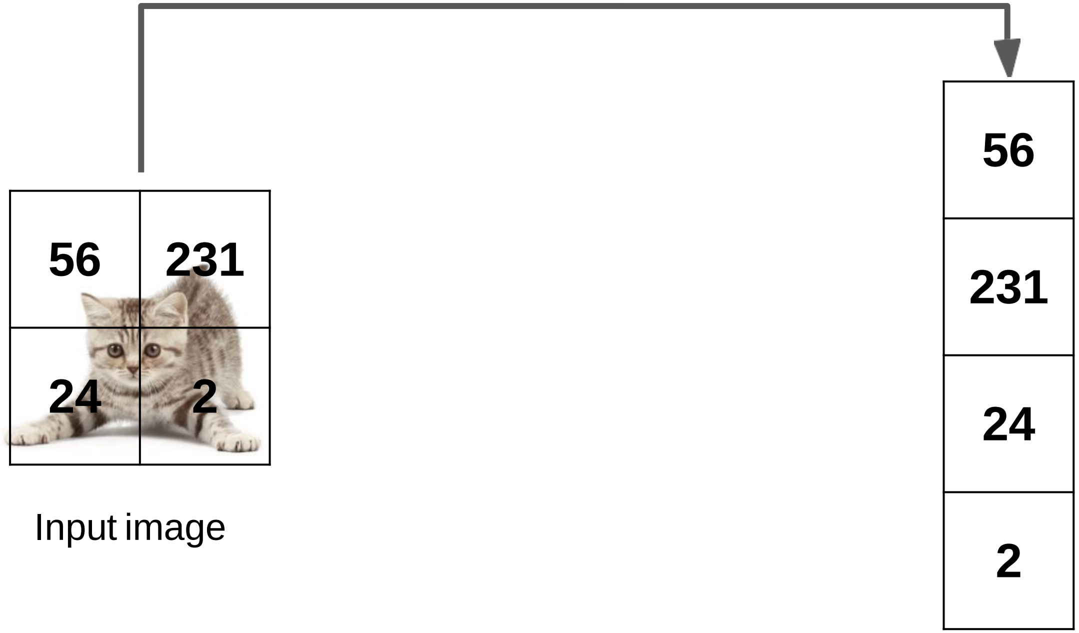
$$f(x,W) = Wx+b$$



Shape: (10,1)

Shape: (3072,1)

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



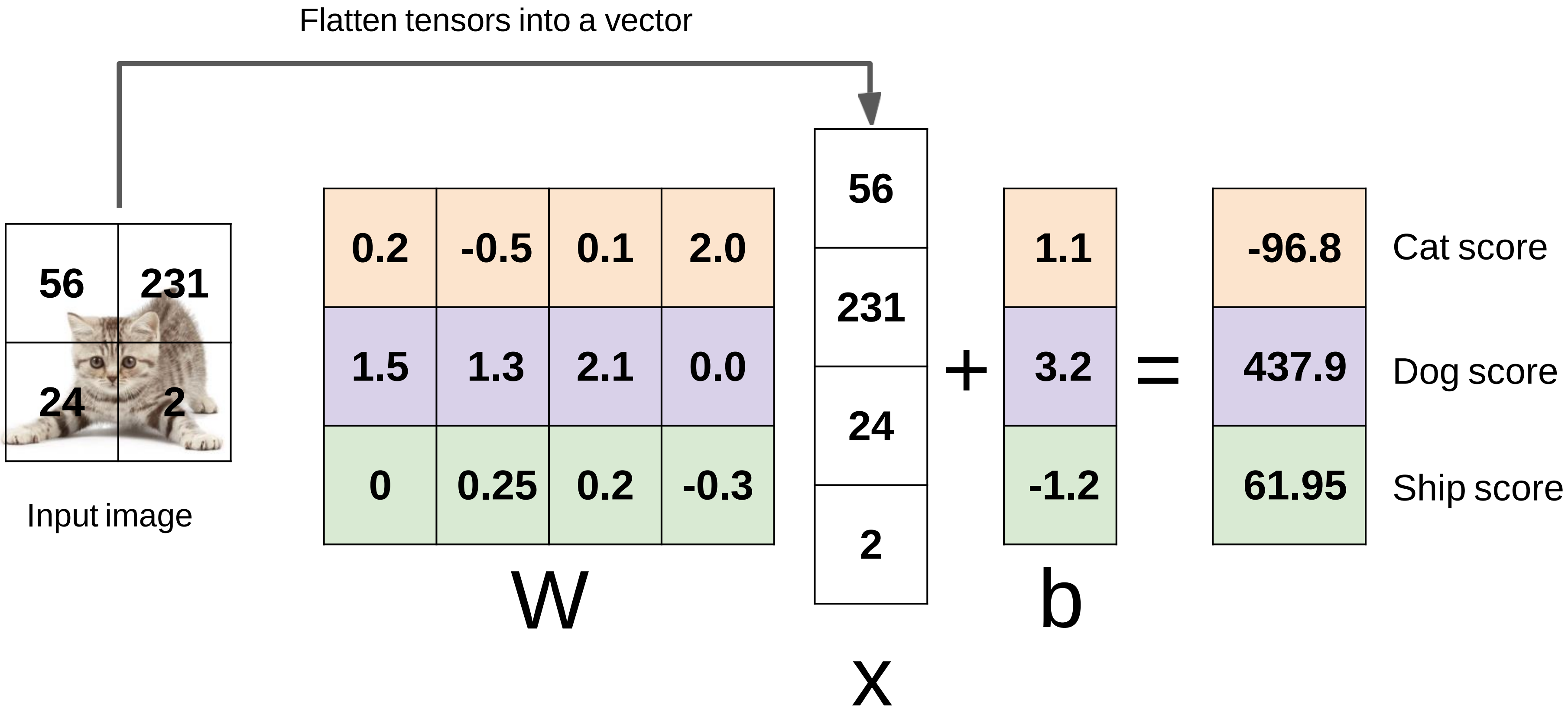
Flatten tensors into a vector

56	231
24	2

Input image

56
231
24
2

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



Flatten tensors into a vector

56	231
24	2

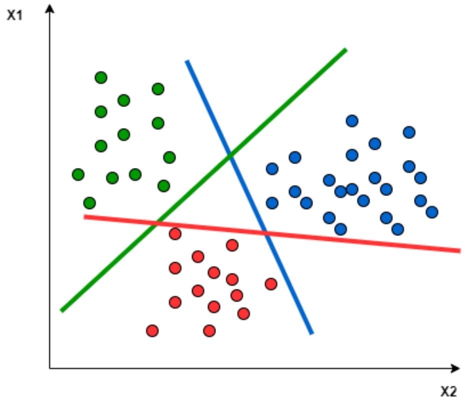
Input image

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} = \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} \begin{matrix} \text{Cat score} \\ \text{Dog score} \\ \text{Ship score} \end{matrix}$$

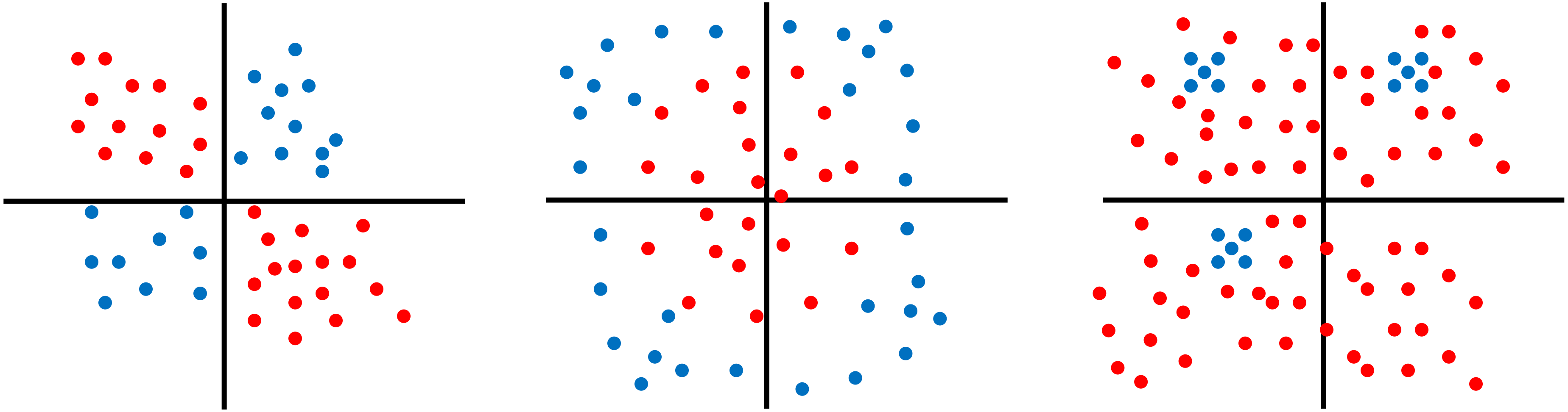
Linear Classifier Predict Efficiently

- Predict fast by generating scores with matrix-vector multiplications

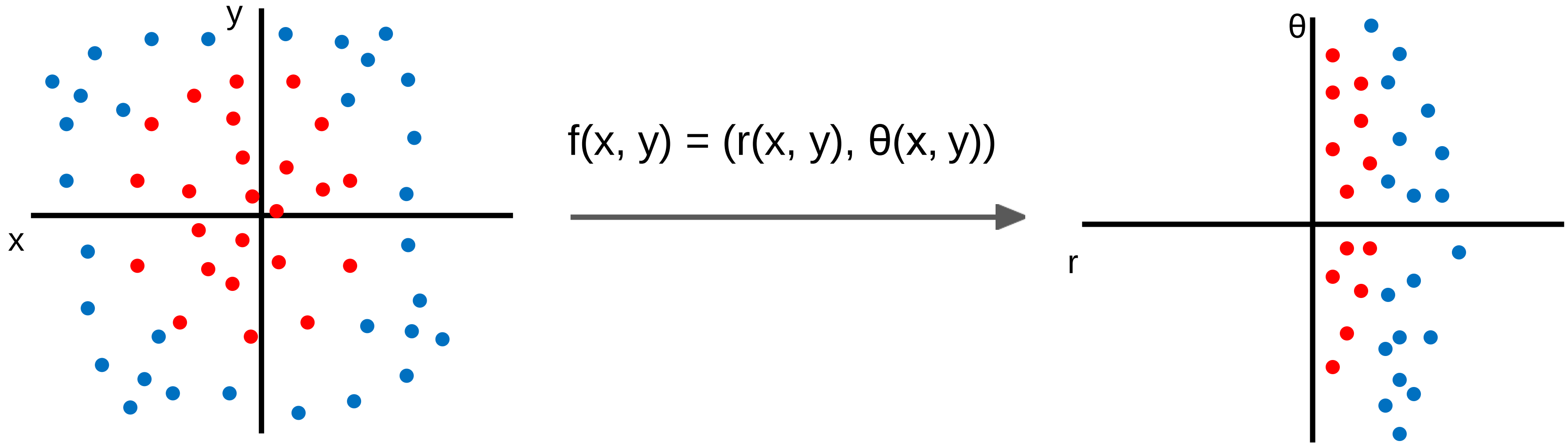
```
scores = W.dot(image) + b
```



Difficult cases for linear classifiers



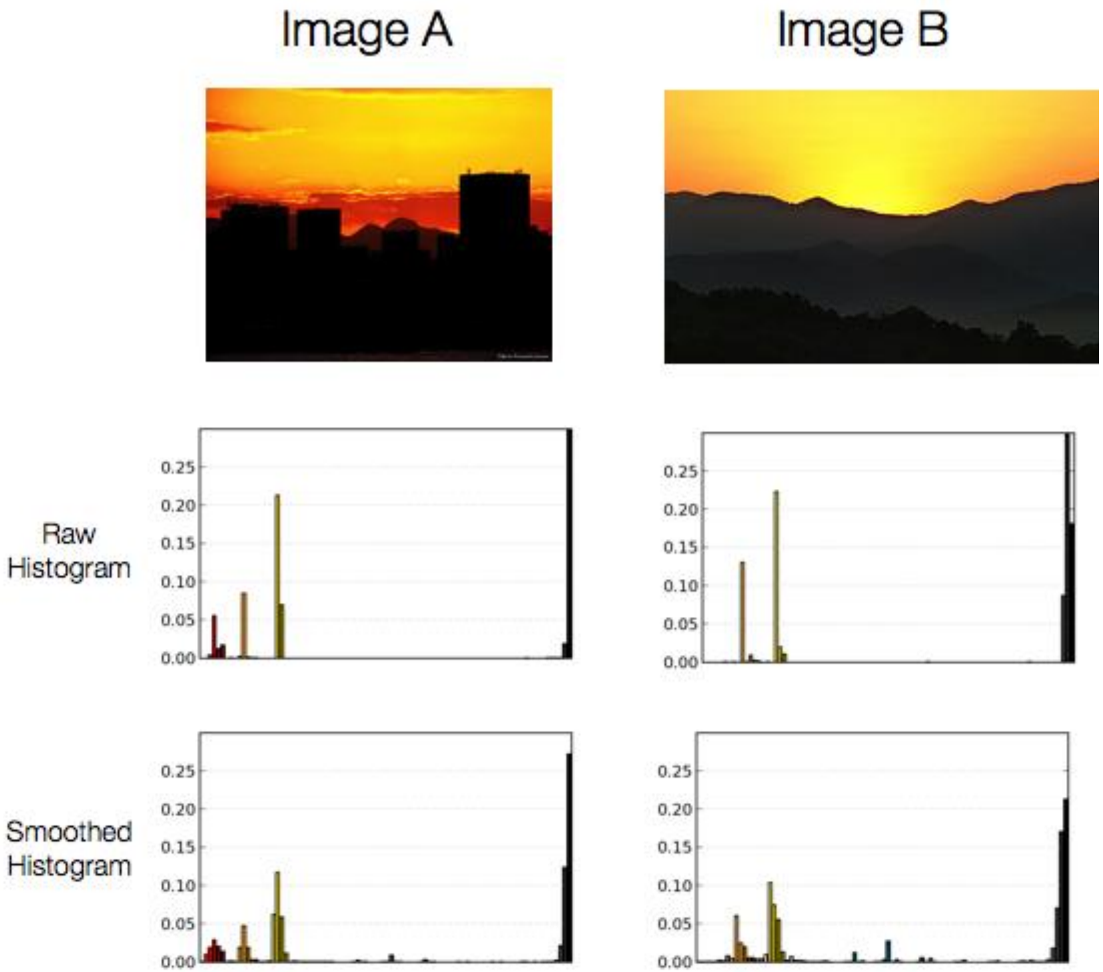
Apply Transformations



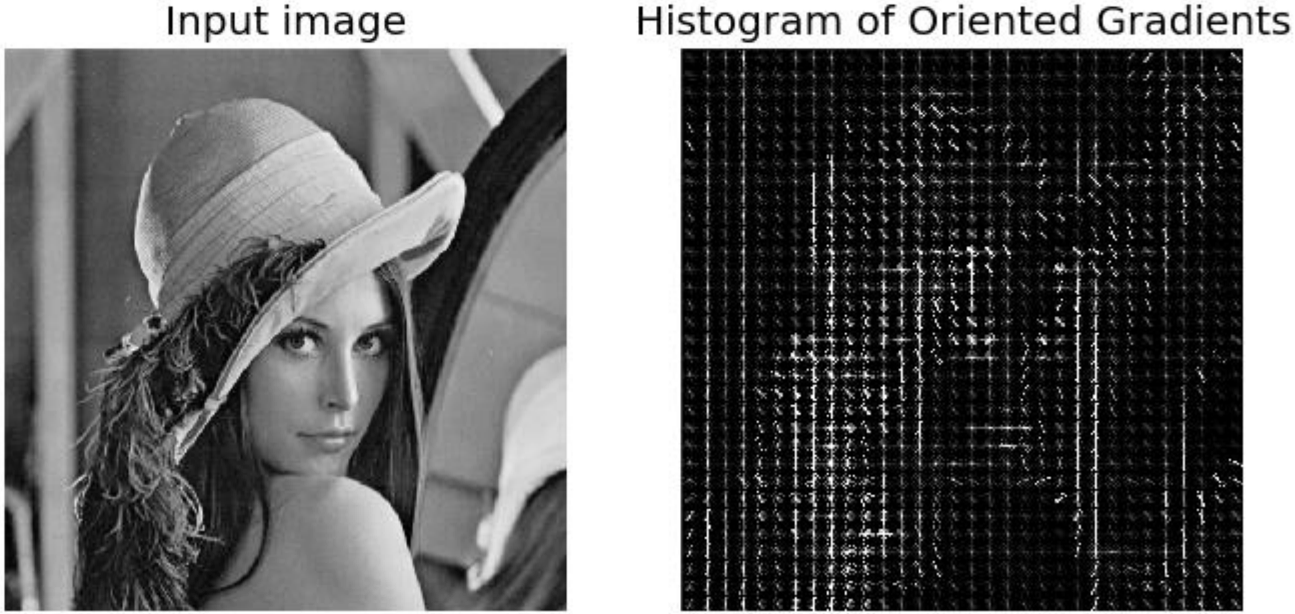
$f(x, y) = (r(x, y), \theta(x, y))$

Extract features using transformations

Example: Color Histogram



Example: Histogram of Oriented Gradients (HoG)

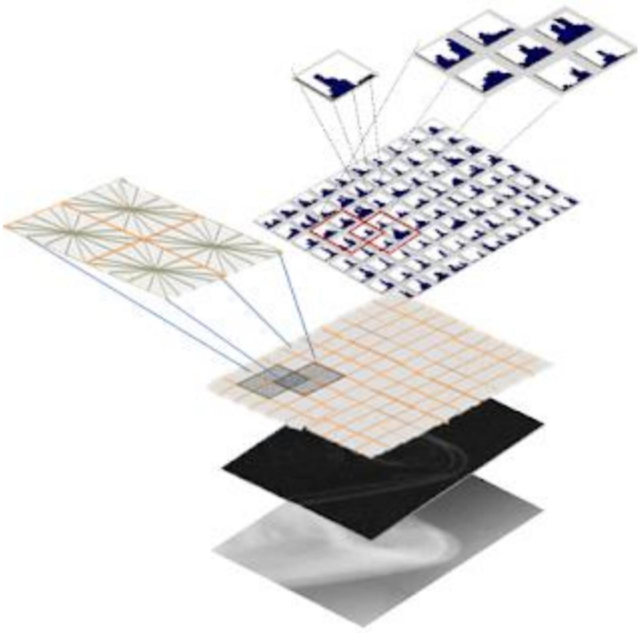


Input image

Histogram of Oriented Gradients

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)



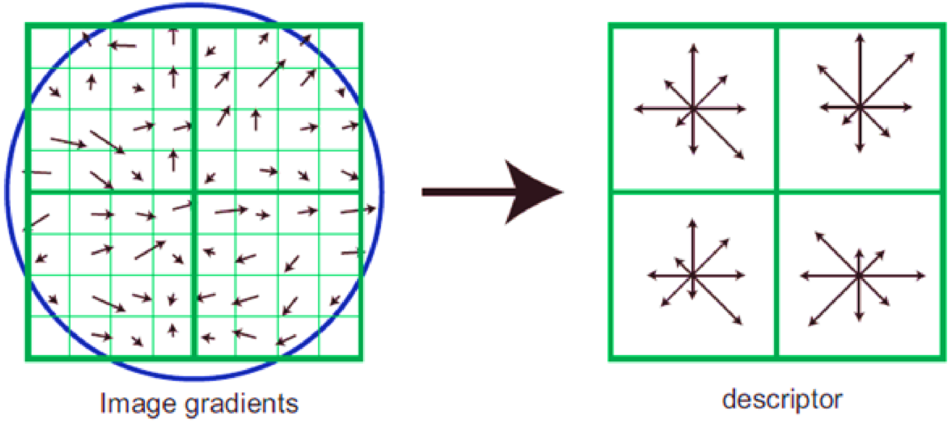
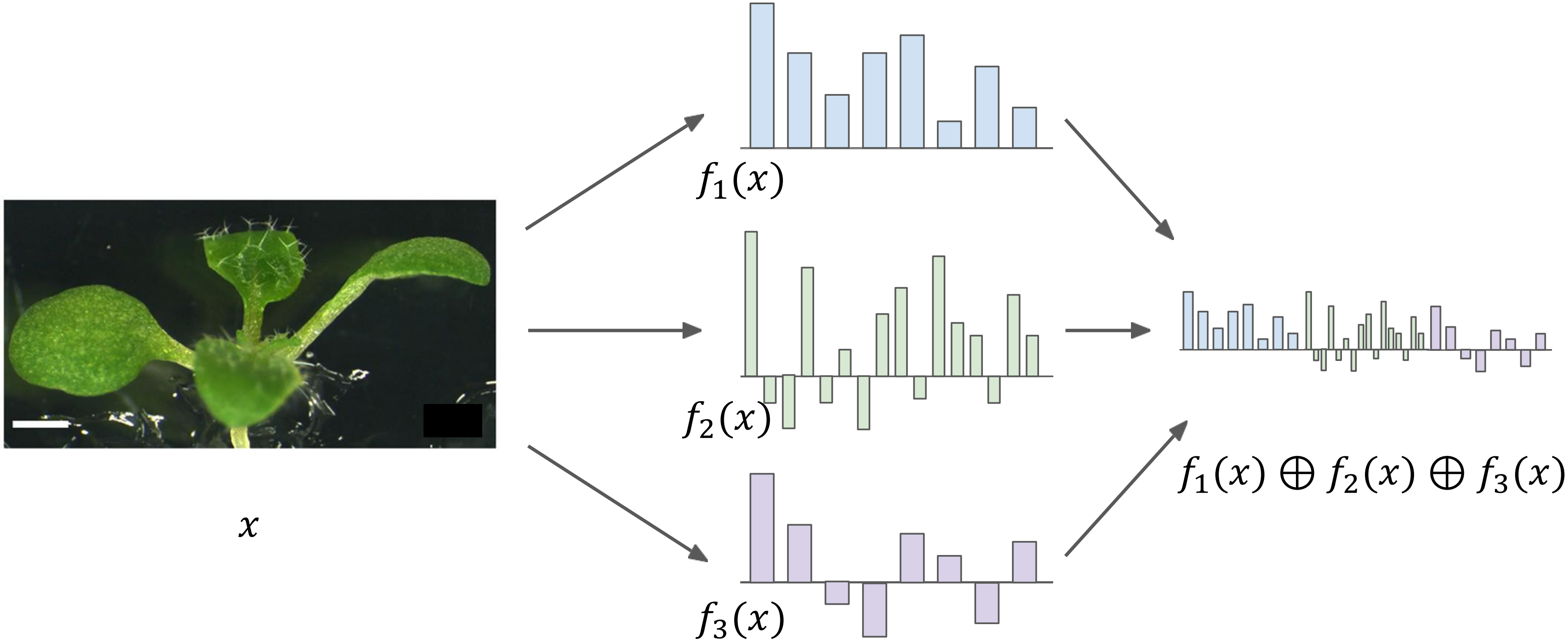
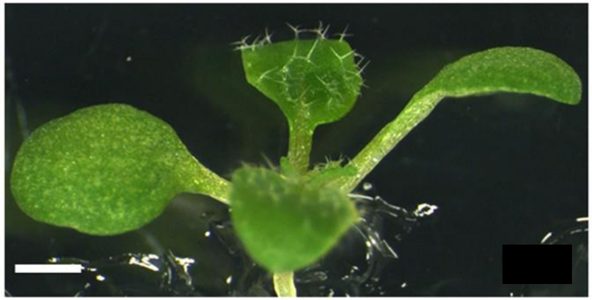


Image Feature Aggregation


[redacted]

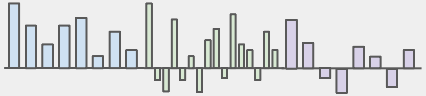
Classification on Image Features



[redacted]

→

Feature Extraction



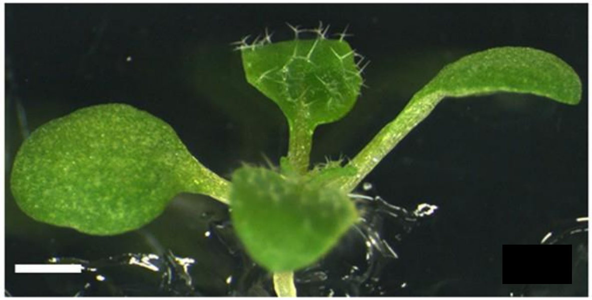
+

Class Label

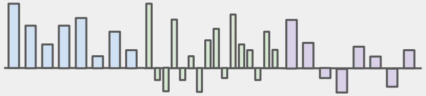
$\xrightarrow{f}$

10 numbers giving scores for classes

Classification on Image Features


[redacted]

Feature Extraction



\+

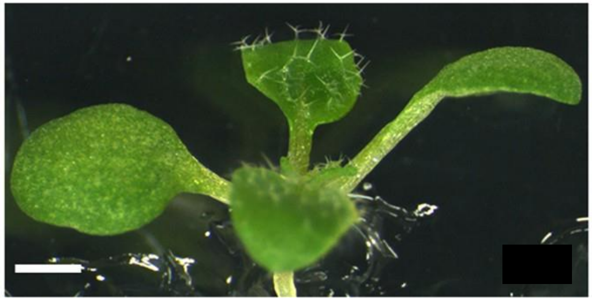
Class Label

W

f

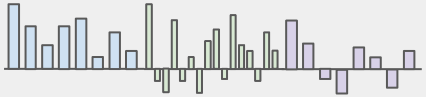
10 numbers giving scores for classes

Classification on Image Features



[redacted]

Feature Extraction



+

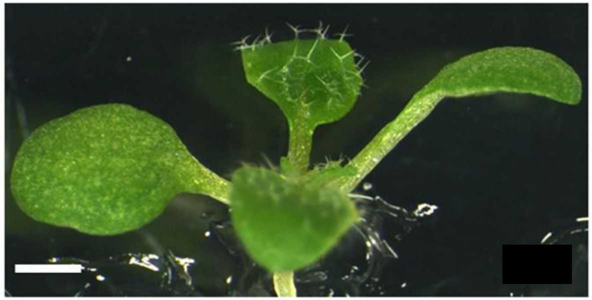
Class Label

W?

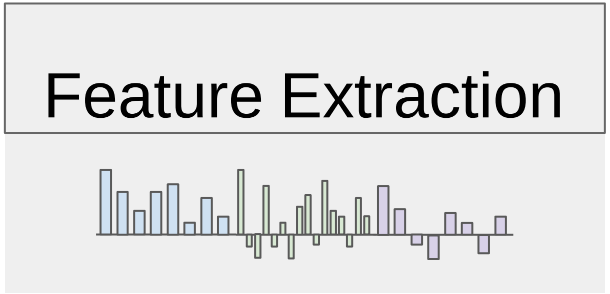
f

10 numbers giving
scores for classes

Classification on Image Features



[redacted]



Feature Extraction

+

Class Label

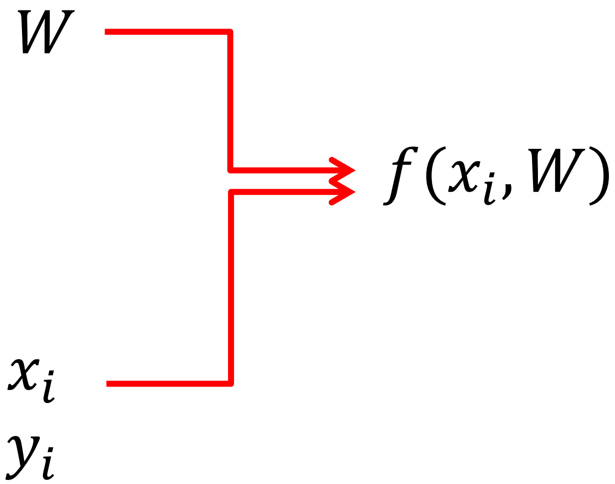
W?

f

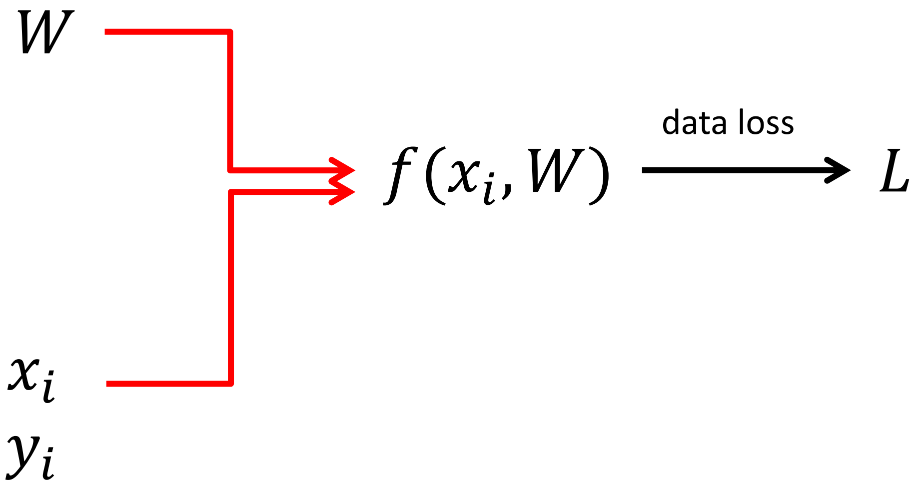
10 numbers giving scores for classes

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

How expressive are the values of W?

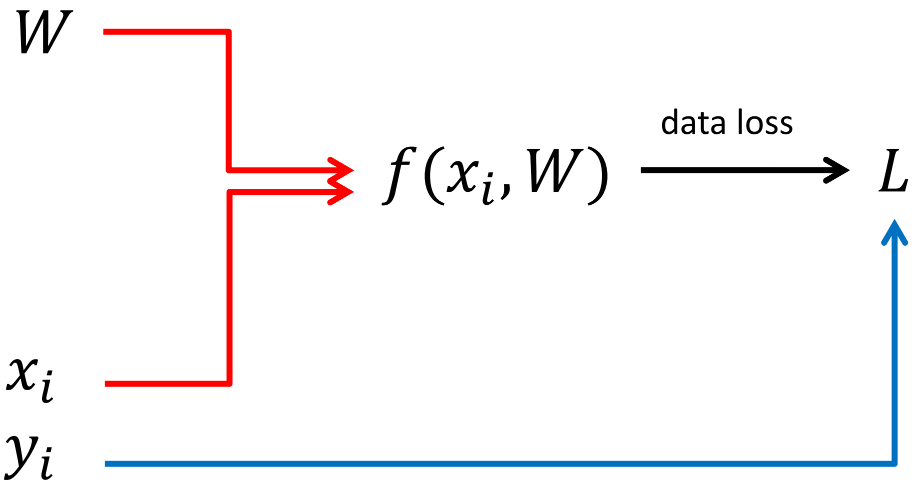


How expressive are the values of W?



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

Loss Function



W

$f(x_i, W) \xrightarrow{\text{data loss}} L$

x_i

y_i

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

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

A **loss function** measures the performance of a classifier

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

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

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

A **loss function** measures the performance of a classifier

Given a dataset of examples

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

Where x_i is image and
y_i is (integer) label

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

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

A **loss function** measures the performance of a classifier

Given a dataset of examples

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

Where x_i is image and
y_i is (integer) label

Loss over the dataset is an average of loss over examples:

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

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

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

Multiclass SVM loss:

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

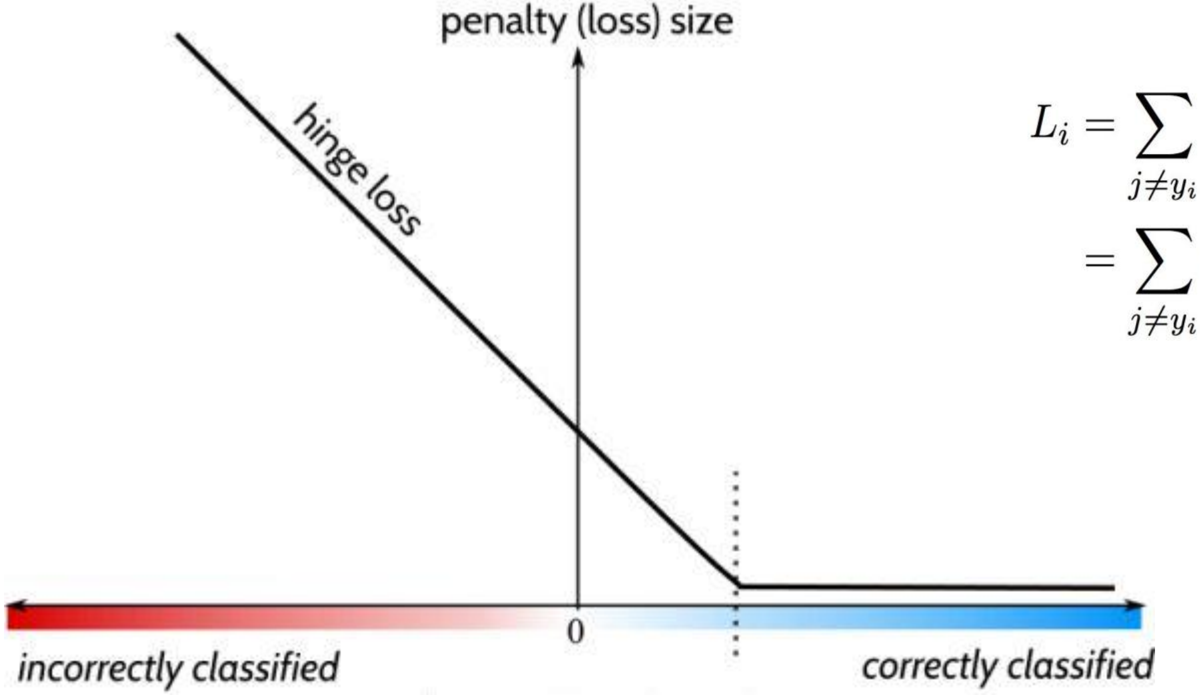
and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

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

Hinge loss



$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

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

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

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

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

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

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

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

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

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

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

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

			
cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9

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

			
cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

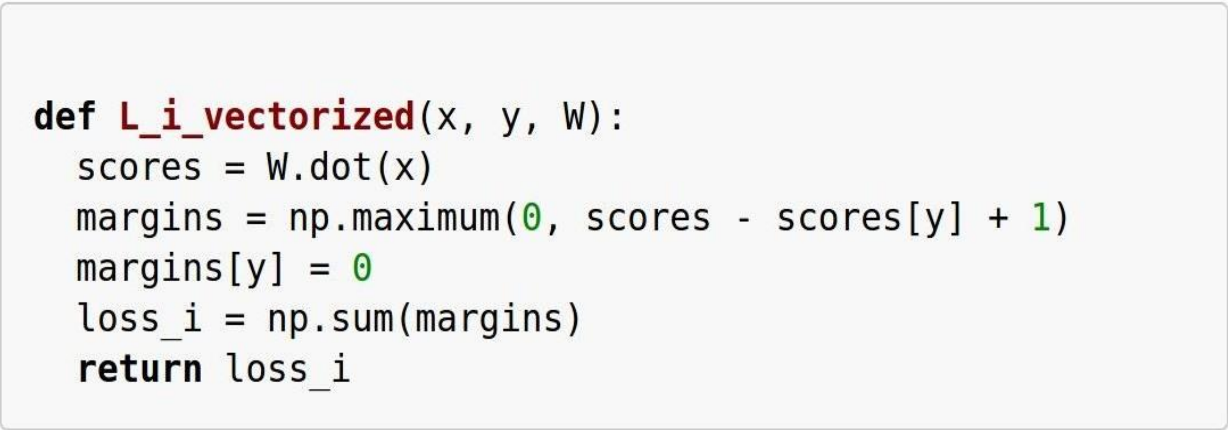
Loss over full dataset is average:

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

L = (2.9 + 0 + 12.9)/3
= **5.27**

Multiclass SVM Loss: Example code

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



```python
def L_i_vectorized(x, y, W):
  scores = W.dot(x)
  margins = np.maximum(0, scores - scores[y] + 1)
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i
```

Suppose we increase W for class 2 twofold

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

cat	**3.2**	2.6	2.2
car	5.1	**9.8**	2.5
frog	-1.7	4.0	**-3.1**
Losses:	2.9	0	12.9

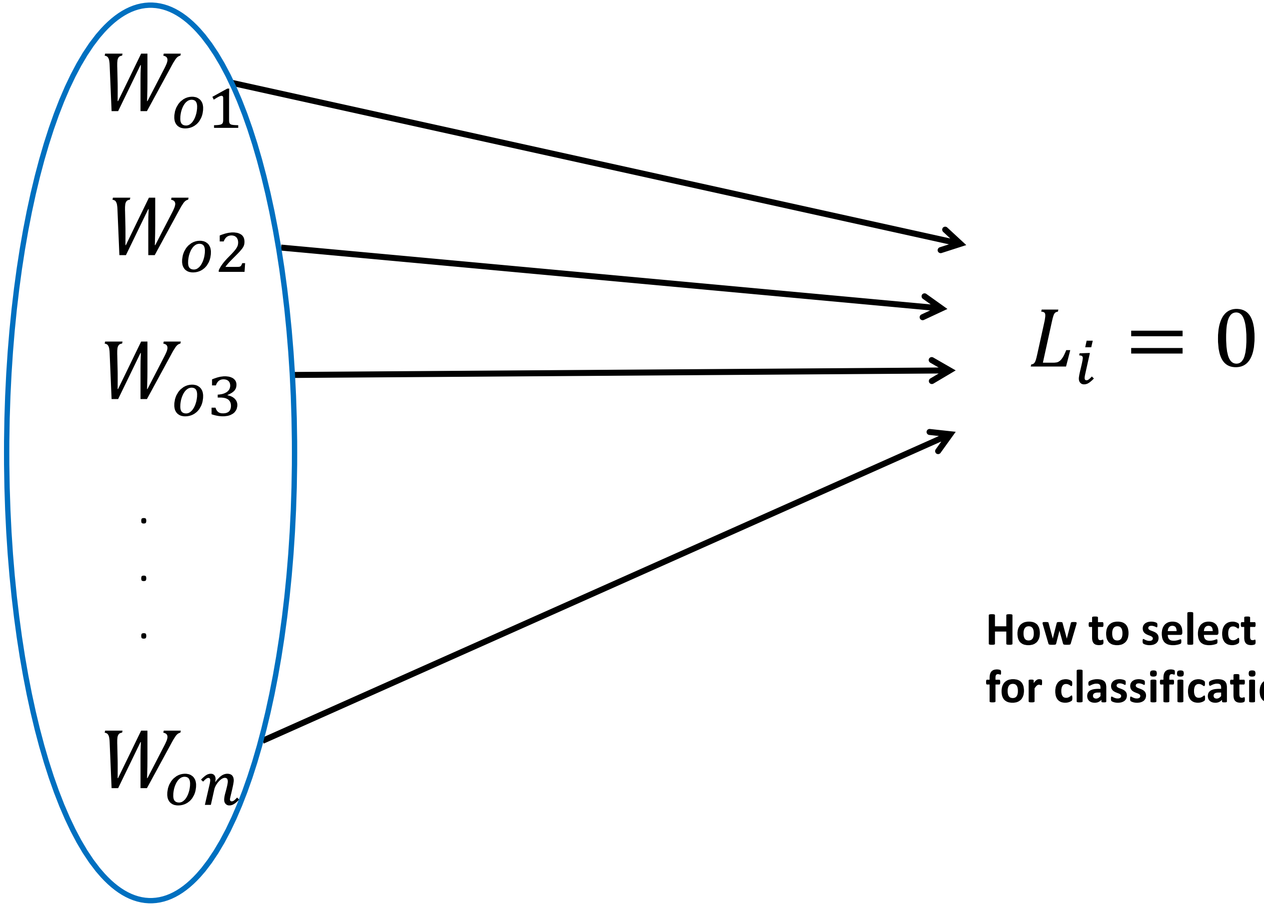
Before:

$= \max(0, 1.3 - 4.9 + 1)$
$\quad + \max(0, 2.0 - 4.9 + 1)$
$= \max(0, -2.6) + \max(0, -1.9)$
$= 0 + 0$
$= 0$

With W twice as large:

$= \max(0, 2.6 - 9.8 + 1)$
$\quad + \max(0, 4.0 - 9.8 + 1)$
$= \max(0, -6.2) + \max(0, -4.8)$
$= 0 + 0$
$= 0$

Set of weights W
that compute 0 loss
for a class



W_{o1}

W_{o2}

W_{o3}

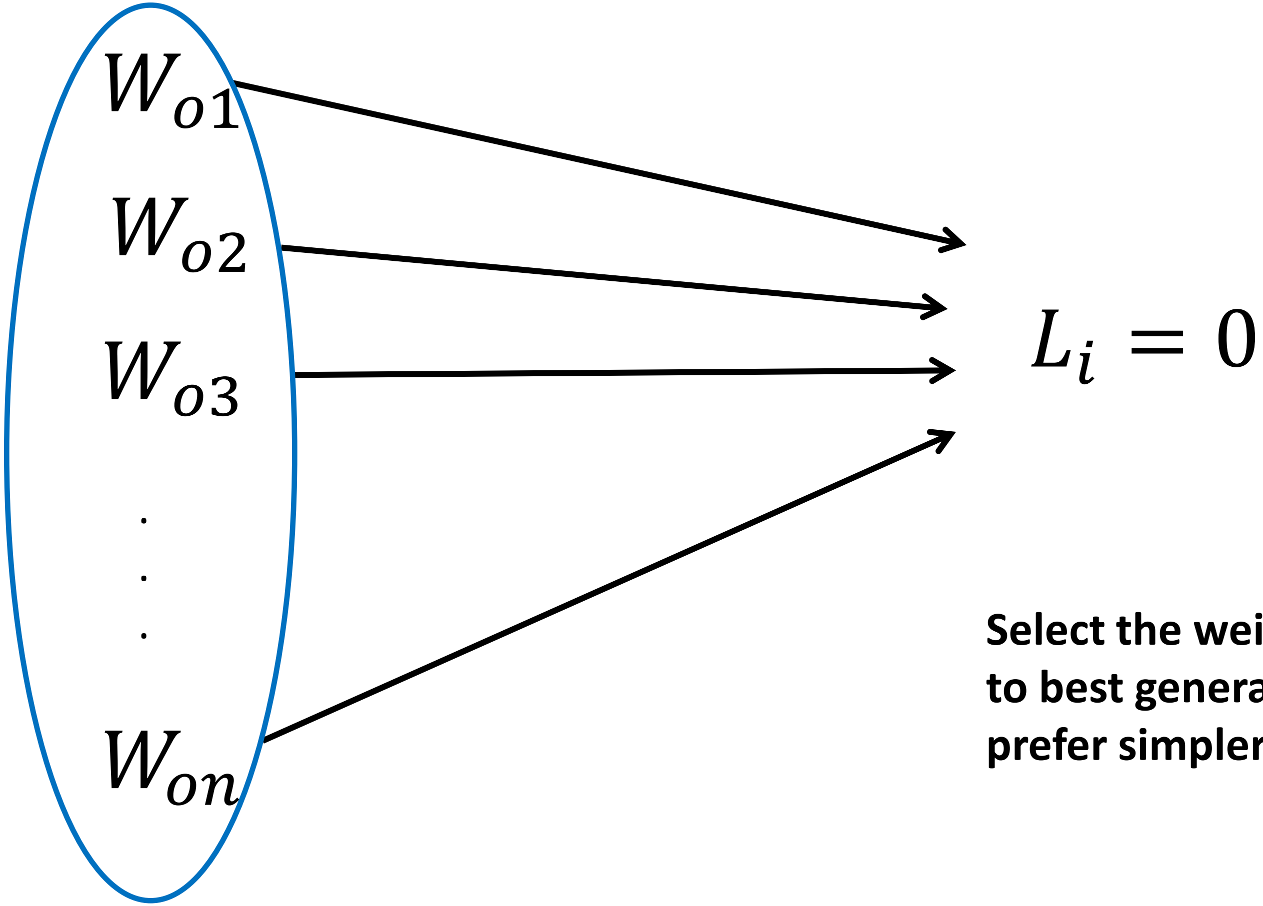
.
.
.

W_{on}

$L_i = 0$

How to select optimal weights for classification?

Set of weights W
that compute 0 loss
for a class



$$L_i = 0$$

Select the weights that lead to best generalization, i.e. prefer simpler models.

Regularization

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

Data loss: Model predictions should match training data

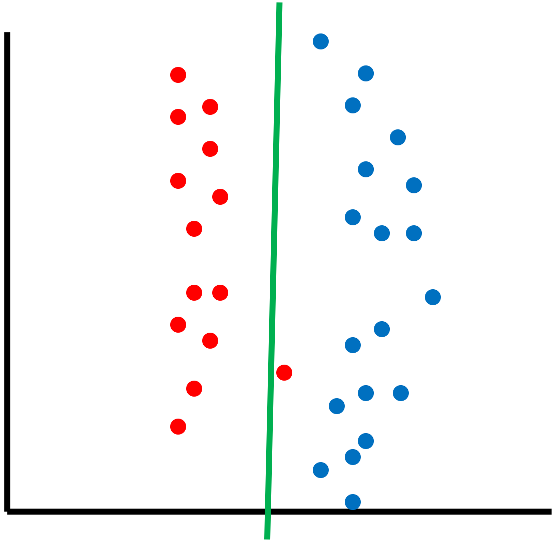
Regularization

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

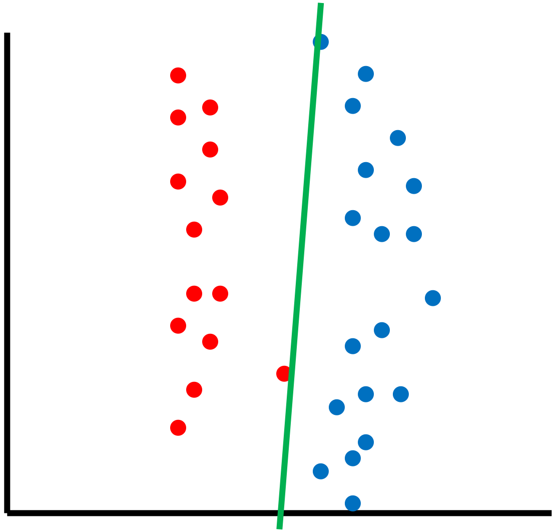
Data loss: Model predictions should match training data

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

Regularization

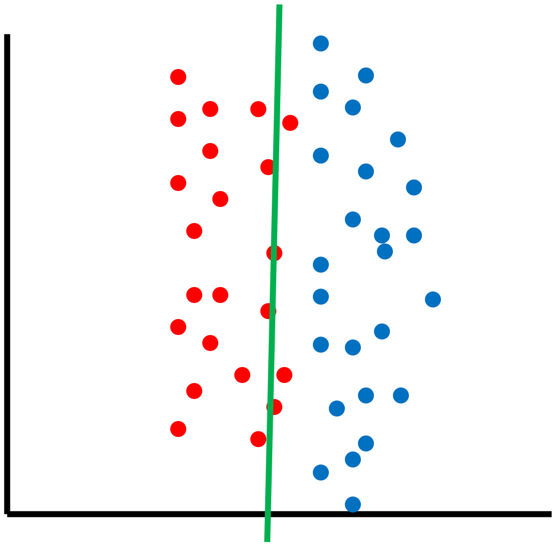


high λ

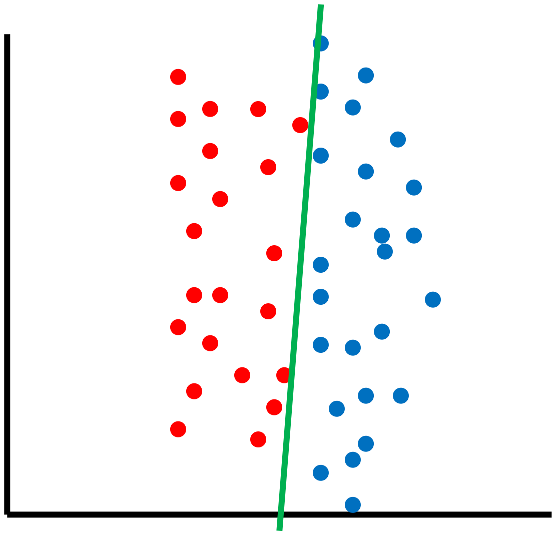


small λ

Regularization: Future Data Ex. 1

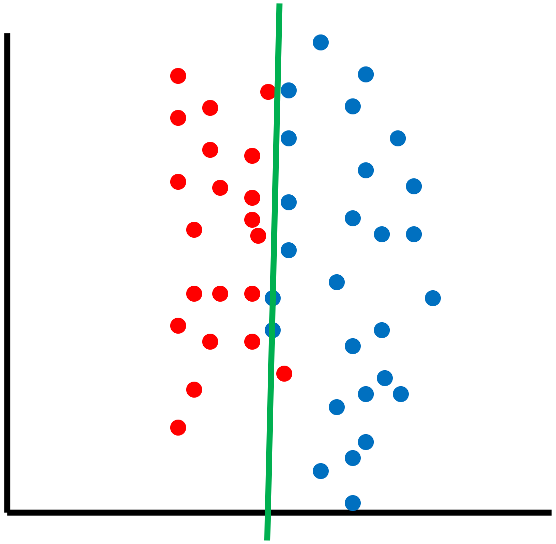


high λ

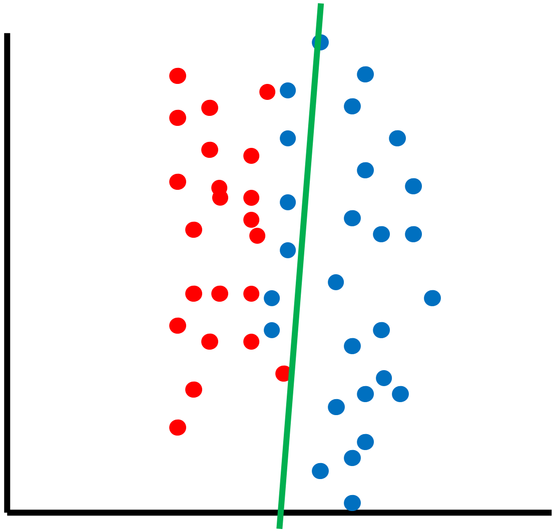


small λ

Regularization: Future Data Ex. 2



high λ



small λ

Regularization

λ = regularization strength
(hyperparameter)

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

Data loss: Model predictions should match training data

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

Regularization

λ = regularization strength (hyperparameter)

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

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

Regularization

λ = regularization strength
(hyperparameter)

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

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

Stochastic depth, fractional pooling, etc

Regularization: Expressing Preferences

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

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

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

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

L2 Regularization

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

Regularization: Expressing Preferences

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

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

$$w_2 = \boxed{[0.25, 0.25, 0.25, 0.25]}$$

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

L2 Regularization

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

L2 regularization prefers evenly spread weights close to 0

Softmax Classifier (Multinomial Logistic Regression)

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$\boxed{s = f(x_i; W)}$$

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

where

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

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

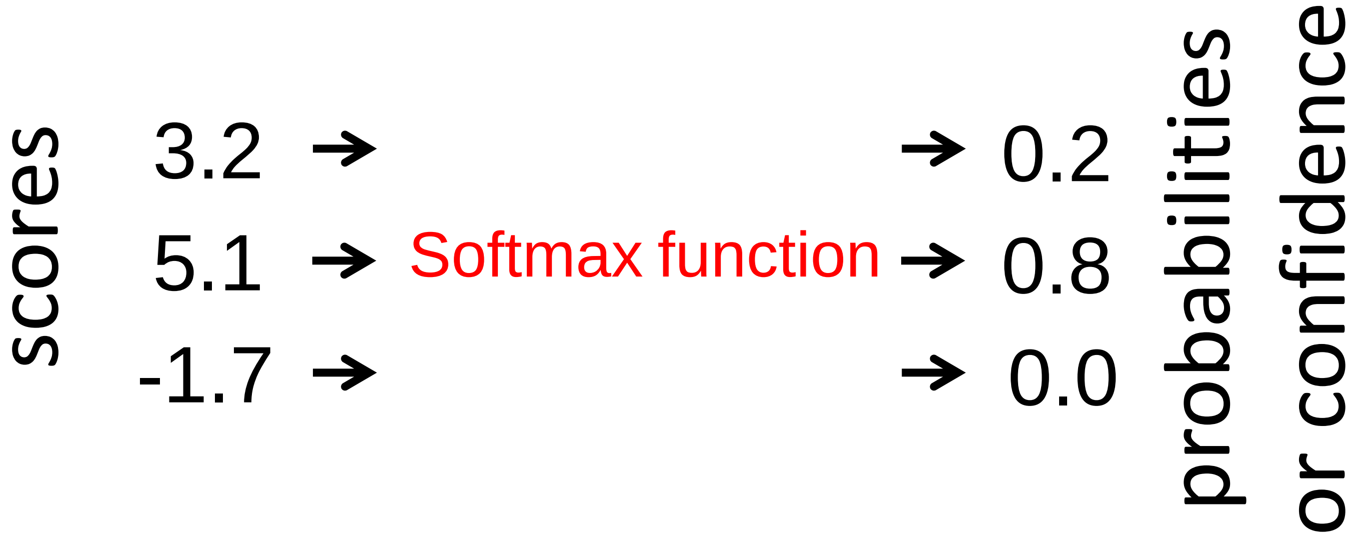
$P(Y = k | X = x_i) = \boxed{\frac{e^{s_k}}{\sum_j e^{s_j}}}$ where $s = f(x_i; W)$

Softmax function

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i) = \boxed{\frac{e^{s_k}}{\sum_j e^{s_j}}} \quad \text{where} \quad s = f(x_i; W)$$



scores

3.2 →

5.1 →

-1.7 →

Softmax function

→ 0.2

→ 0.8

→ 0.0

probabilities
or confidence

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$\boxed{P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}}$$ where $$\boxed{s=f(x_i;W)}$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

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

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

where

$$\boxed{s=f(x_i;W)}$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

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

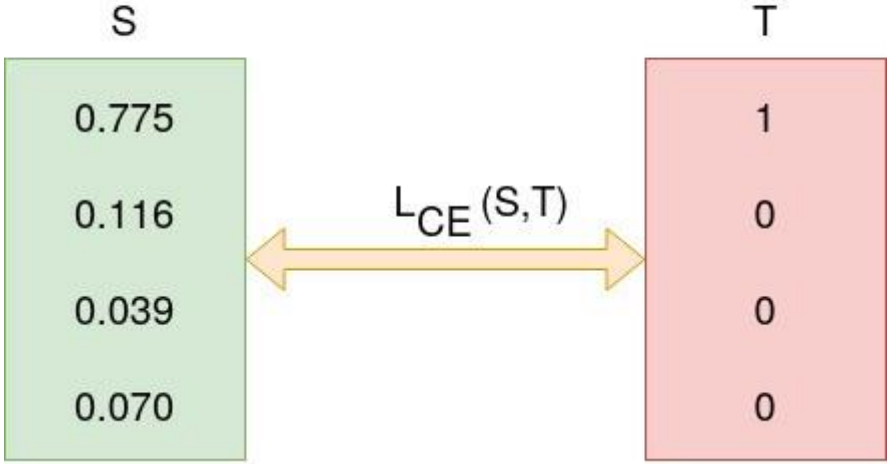
---

in summary:

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

Cross-Entropy Loss

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



S

0.775

0.116

0.039

0.070

$L_{CE}(S,T)$

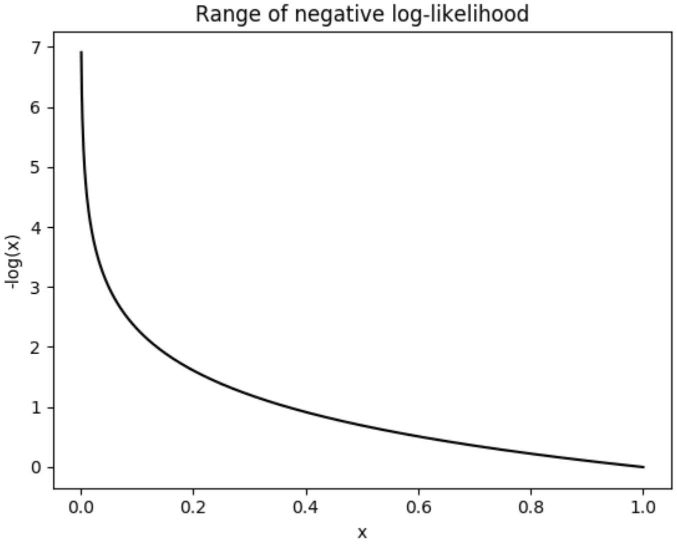
T

1

0

0

0



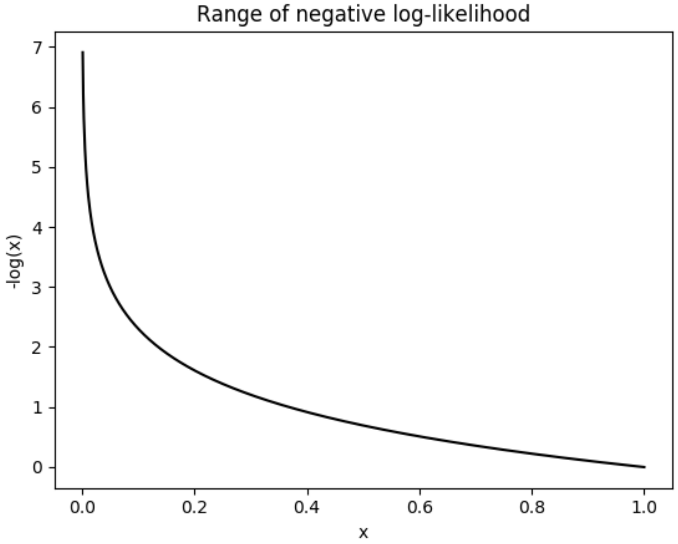
Cross-Entropy Loss

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

Multiplying many probabilities/likelihood may lead to very small numbers:
e.g. 0.9*0.1*0.01 = 0.0009 → this is **undesirable**

To avoid this we can express products as **sums** by using the **log function**:

$$\log(a \cdot b) = \log(a) + \log(b)$$



Softmax Classifier (Multinomial Logistic Regression)

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

cat	**3.2**
car	5.1
frog	-1.7

unnormalized log probabilities

Softmax Classifier (Multinomial Logistic Regression)

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

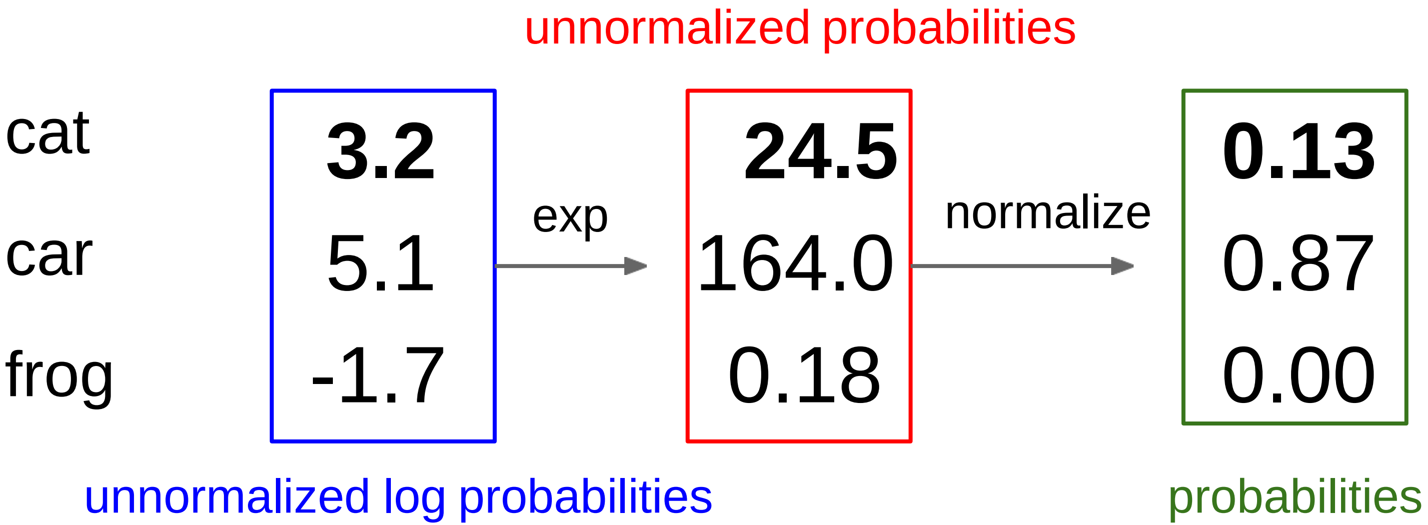
unnormalized probabilities

cat	**3.2**	exp →	**24.5**
car	5.1		164.0
frog	-1.7		0.18

unnormalized log probabilities

Softmax Classifier (Multinomial Logistic Regression)

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



unnormalized probabilities

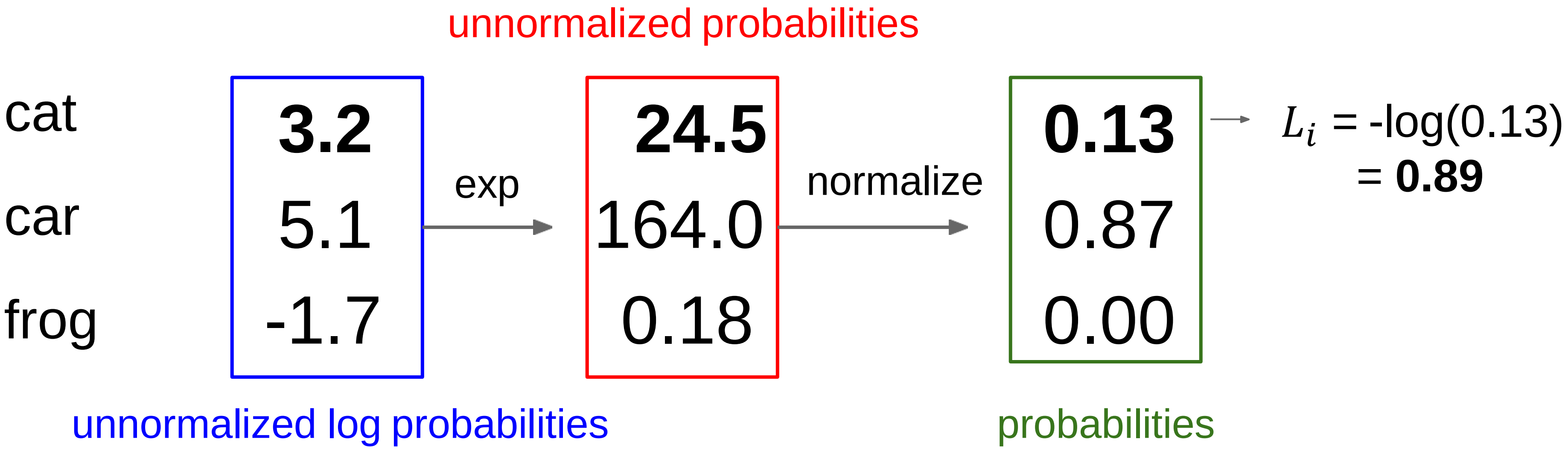
cat	**3.2**	exp →	**24.5**	normalize →	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

unnormalized log probabilities

probabilities

Softmax Classifier (Multinomial Logistic Regression)

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



unnormalized probabilities

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

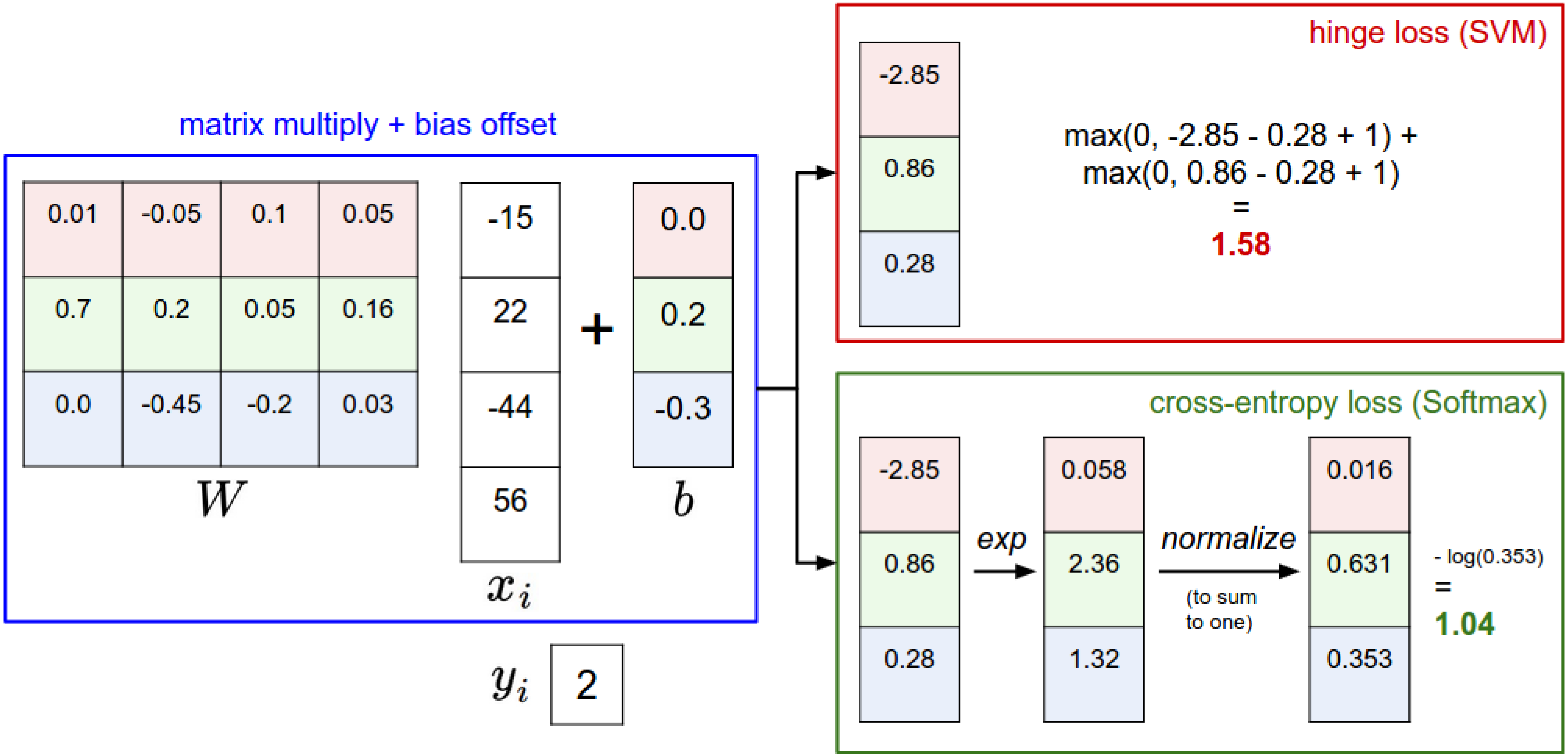
0.13
0.87
0.00

L_i = -log(0.13)
= **0.89**

unnormalized log probabilities

probabilities

Softmax vs. SVM



matrix multiply + bias offset

$$\underbrace{\begin{bmatrix} 0.01 & -0.05 & 0.1 & 0.05 \\ 0.7 & 0.2 & 0.05 & 0.16 \\ 0.0 & -0.45 & -0.2 & 0.03 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix} 0.0 \\ 0.2 \\ -0.3 \end{bmatrix}}_{b}$$

y_i $\boxed{2}$

hinge loss (SVM)

$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

max(0, -2.85 - 0.28 + 1) +
max(0, 0.86 - 0.28 + 1)
=
1.58

cross-entropy loss (Softmax)

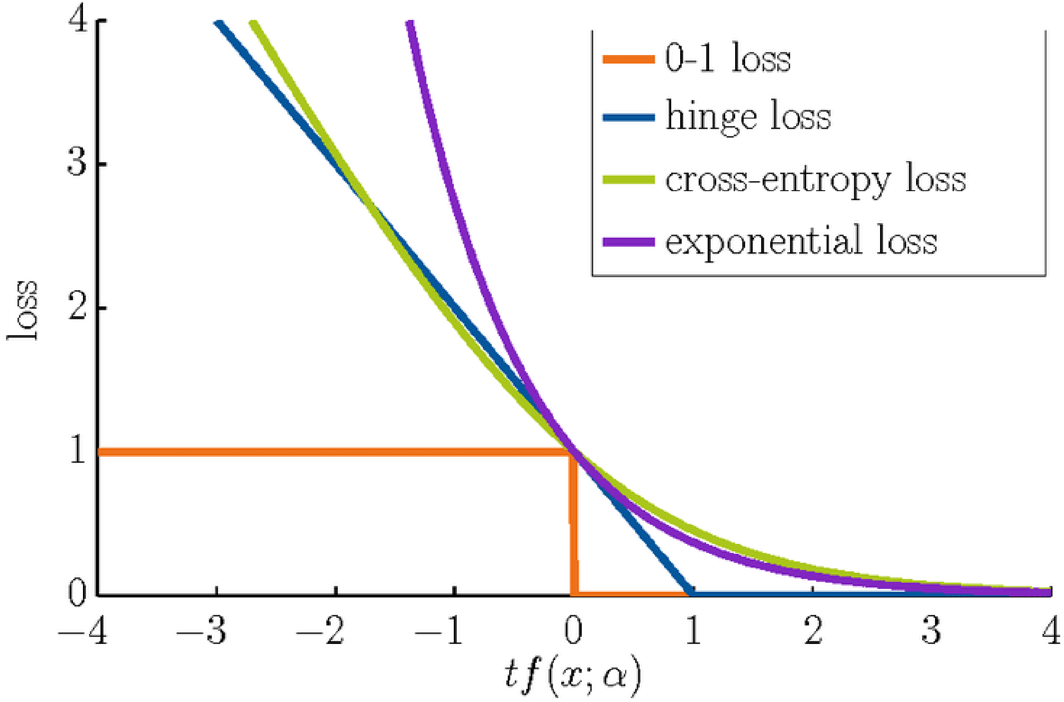
$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix} \xrightarrow{exp} \begin{bmatrix} 0.058 \\ 2.36 \\ 1.32 \end{bmatrix} \xrightarrow[\text{(to sum to one)}]{normalize} \begin{bmatrix} 0.016 \\ 0.631 \\ 0.353 \end{bmatrix}$$

- log(0.353)
=
1.04

Softmax vs. SVM

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

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



Summary

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

$$s = f(x;W) \overset{\text{e.g.}}{=} Wx$$

Softmax

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

SVM

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

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

Complete loss