CST31211
: Deep Learning

Classifiers

Image Classification: a core task in Computer Vision

Input: image

Output:Assign image to one of a fixed set of categories {dog, cat, truck, plane, ...}

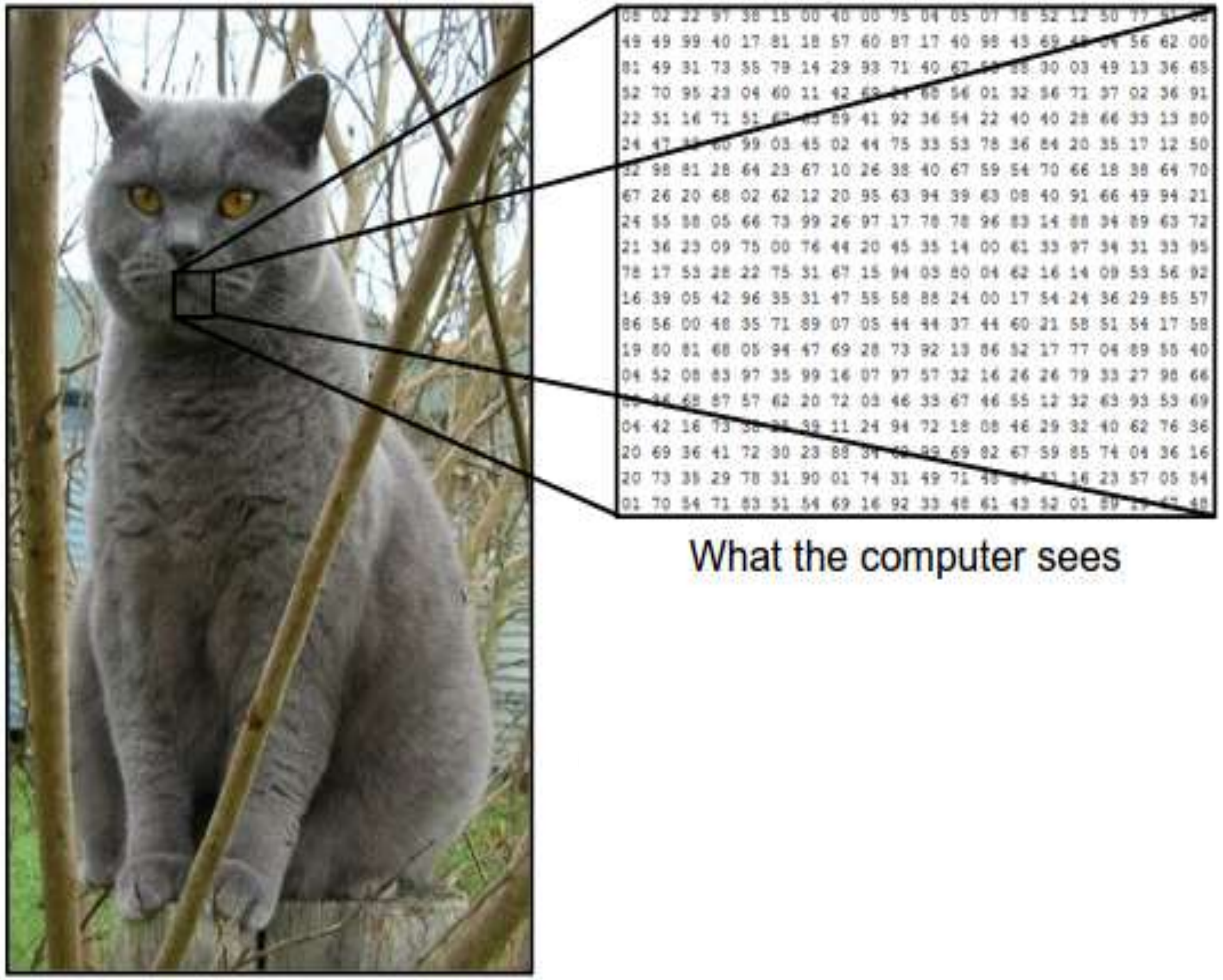
→ cat

The problem: *semantic gap*

Images are represented as 3D arrays of numbers, with integers between [0, 255].

E.g.
300 x 100 x 3

(3 for 3 color channels RGB)



Challenges: Illumination

Challenges: Deformation

Challenges: Occlusion



Challenges: Background clutter

Challenges: Intraclass variation

An image classifier

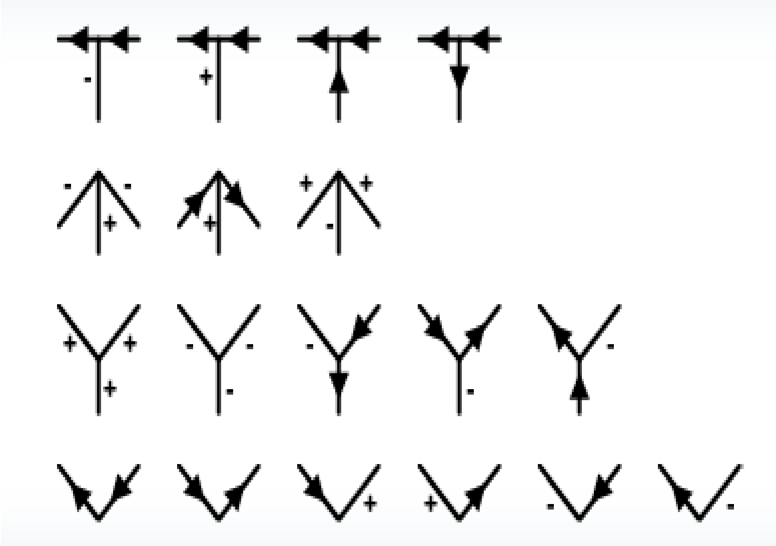
```
def predict(image):
    # ????
    return class_label
```

Unlike e.g. sorting a list of numbers,

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

Classifying using constraints? ?

???



Data-driven approach:

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

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Example training set

First classifier: Nearest Neighbor Classifier

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Remember all training images and their labels

Predict the label of the most similar training image

data-driven approach

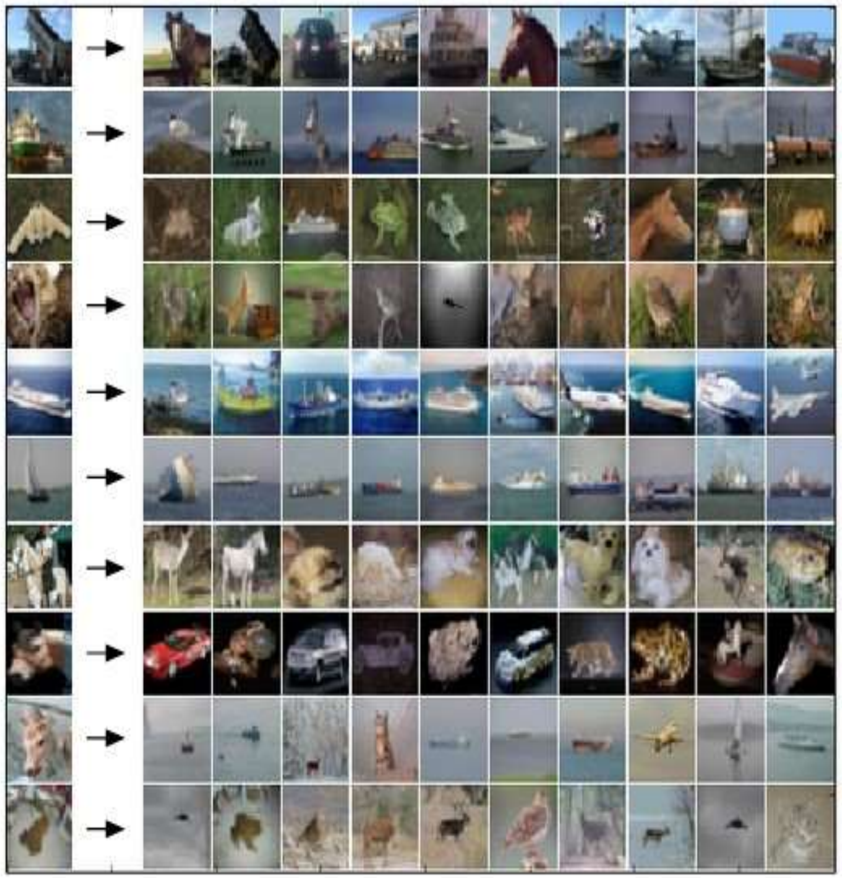
Example dataset: CIFAR-10

10 labels
50,000 training RGB images, each image is tiny: 32x32, (5k per class)
10,000 test images. (1k per class)

Example dataset: CIFAR-10

10 labels
50,000 training images
10,000 test images.

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



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

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

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

L1 (Manhattan) distance

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

L2 (Euclidean) distance

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

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
```

Numpy：http://www.numpy.org.cn/article/basics/python_numpy_tutorial.html

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
```

remember the training data

Nearest Neighbor classifier

```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

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

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

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

    return Ypred
```

for every test image:

- find nearest train image with L1 distance
- predict the label of nearest training 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: how does the classification speed depend on the size of the training data?

Nearest Neighbor classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

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

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

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

    return Ypred
```

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

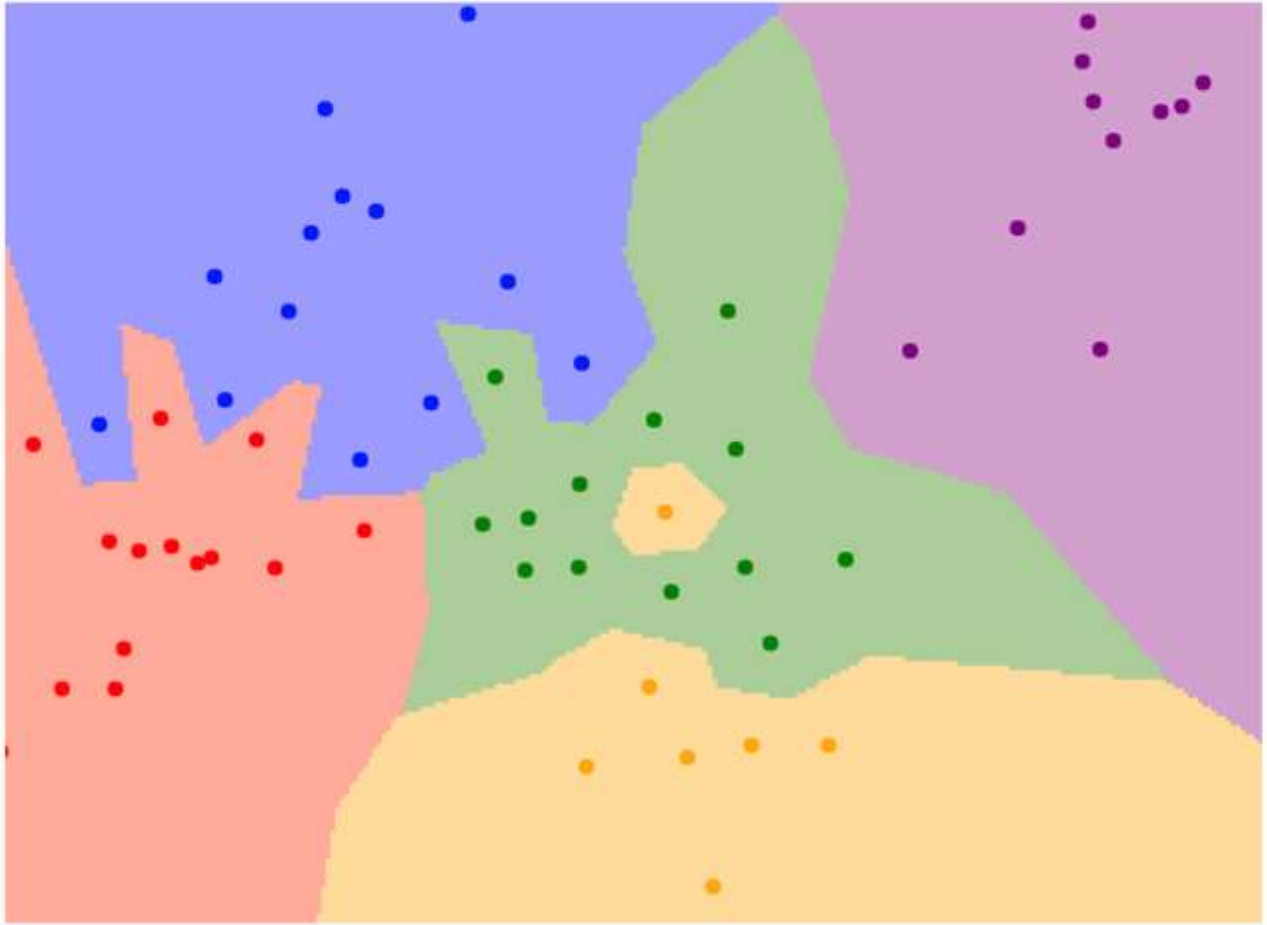
This is **backwards**:
- test time performance is usually much more important in practice.
- CNNs flip this: expensive training, cheap test evaluation

There are many methods for fast / approximate nearest neighbors; e.g. see
https://github.com/facebookresearch/faiss

Nearest Neighbor Decision Boundaries 决策边界

- Points are training examples; colors give training labels

- Background colors give the category a test point would be assigned



How to smooth out decision boundaries?
Use more neighbors!

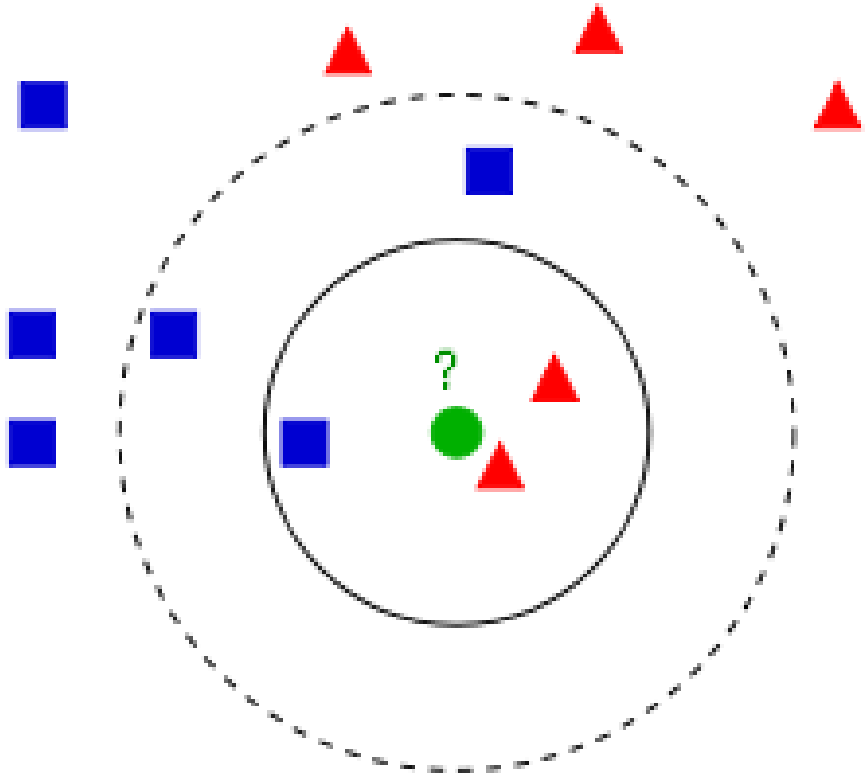
- **Decision boundary** is the boundary between two classification regions

边界不平滑，
泛化性差,
易受噪声影响

k-Nearest Neighbor

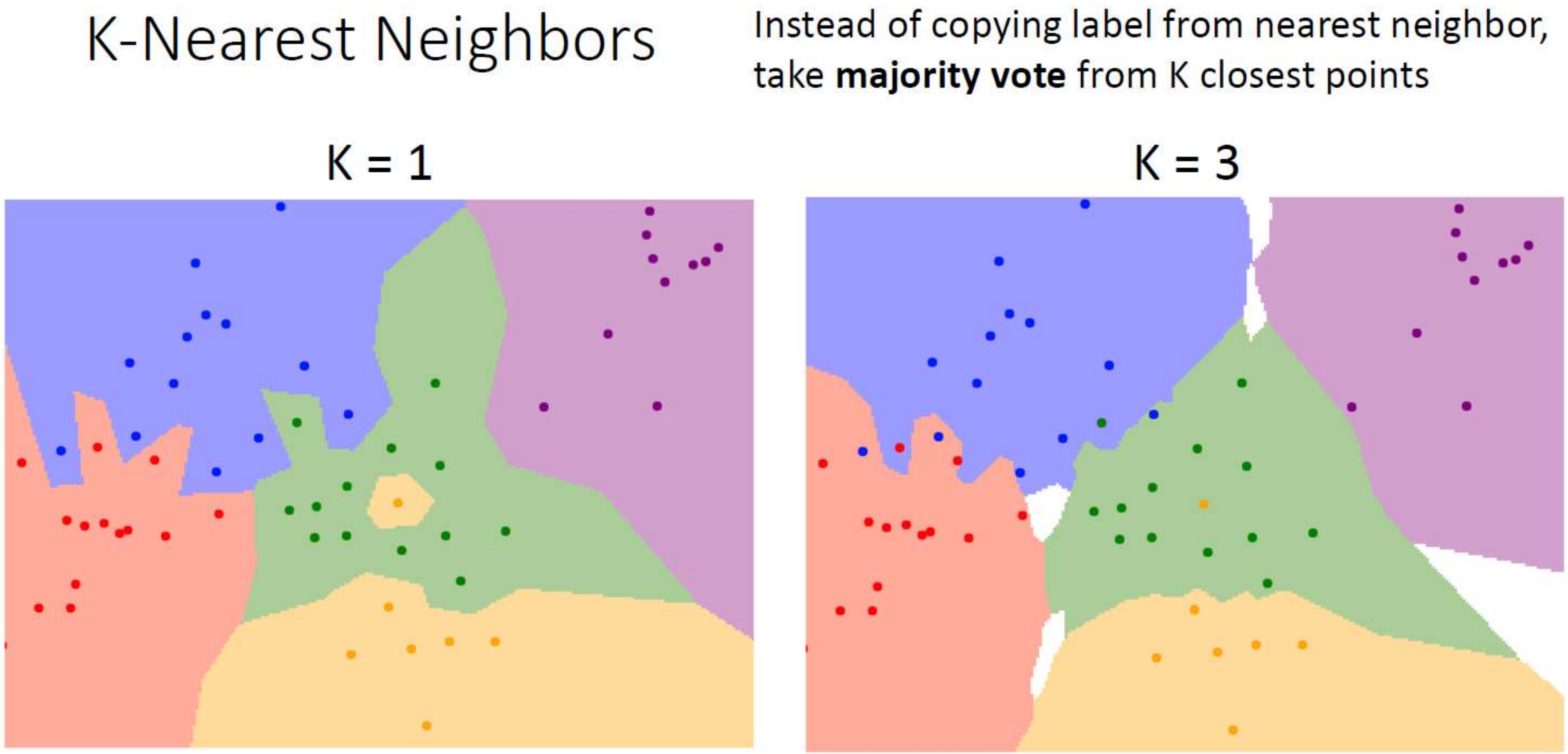
find the k nearest images, have them vote on the label

K=1,K=3,K=5?



k-Nearest Neighbor

find the k nearest images, have them vote on the label

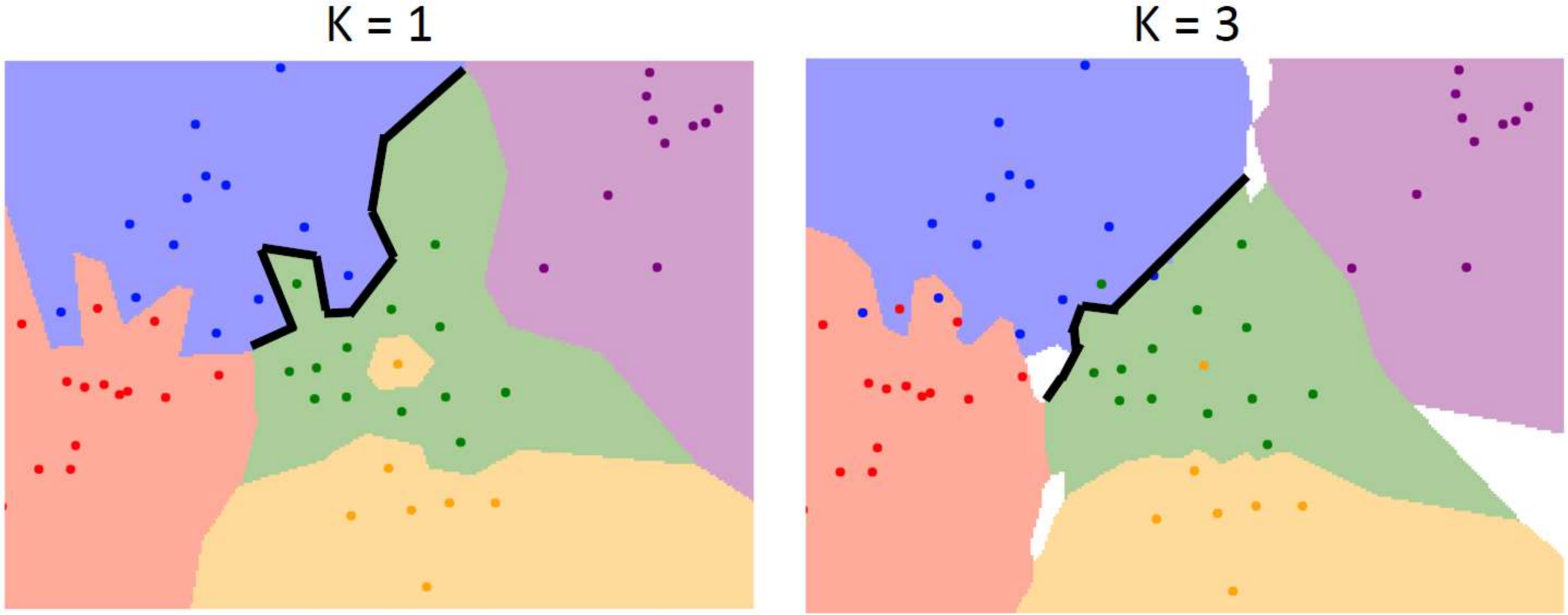


vision.stanford.edu/teaching/cs231n-demos/knn/

http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

K-Nearest Neighbors

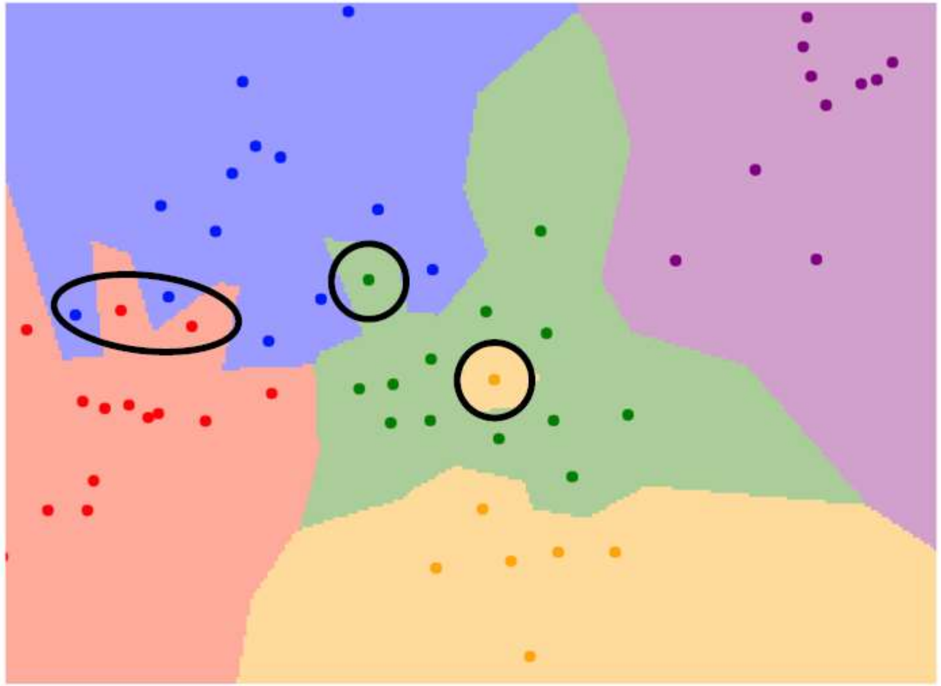
Using more neighbors helps smooth out rough decision boundaries



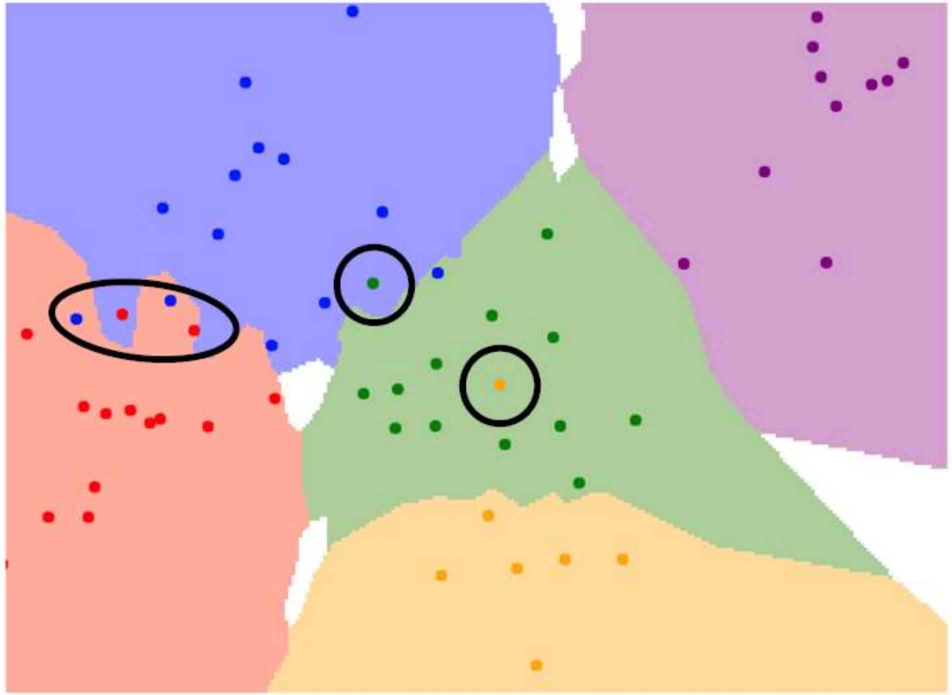
K-Nearest Neighbors

Using more neighbors helps reduce the effect of outliers

K = 1

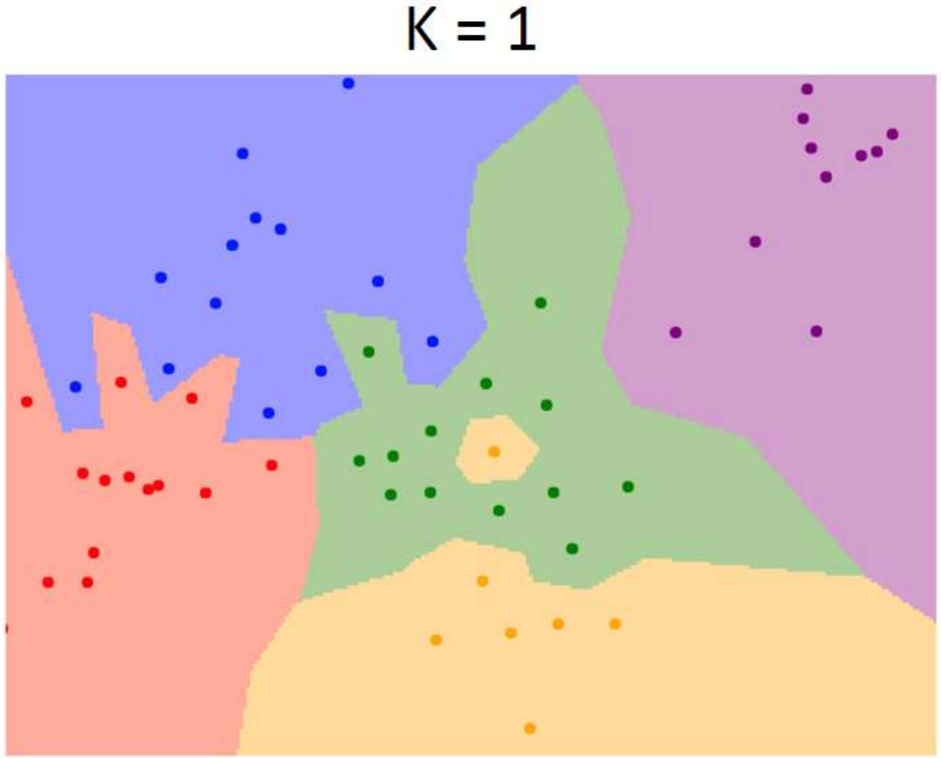


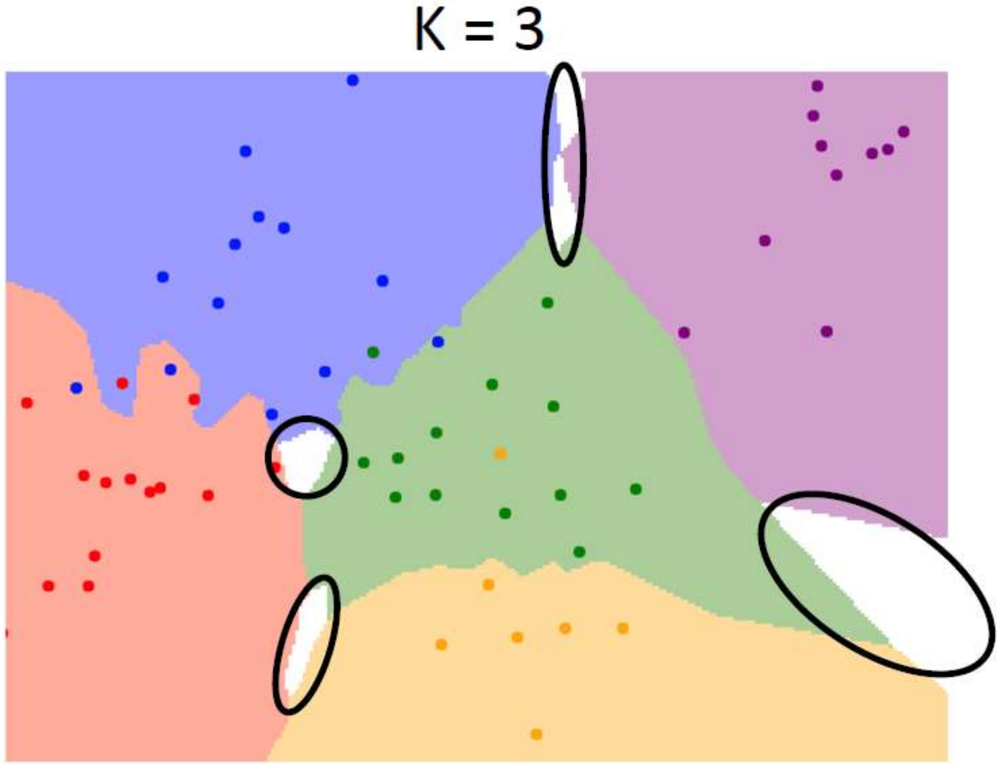
K = 3



K-Nearest Neighbors

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

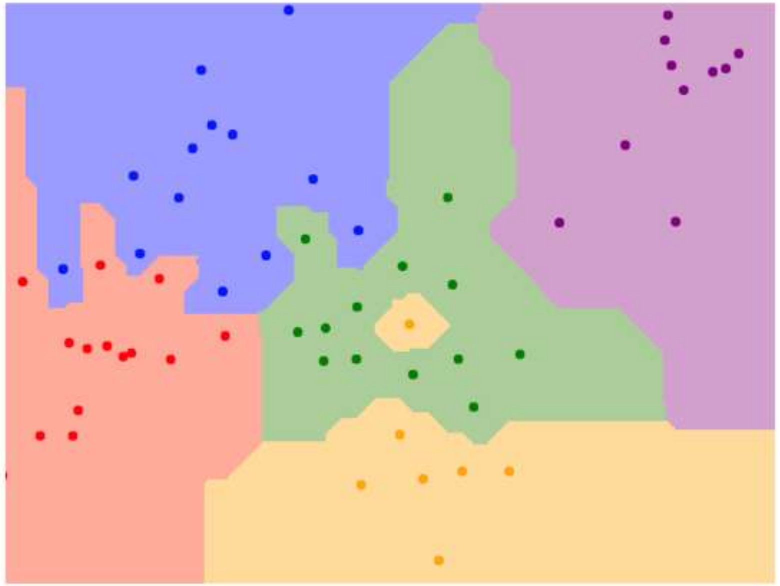




K-Nearest Neighbors: Distance Metric

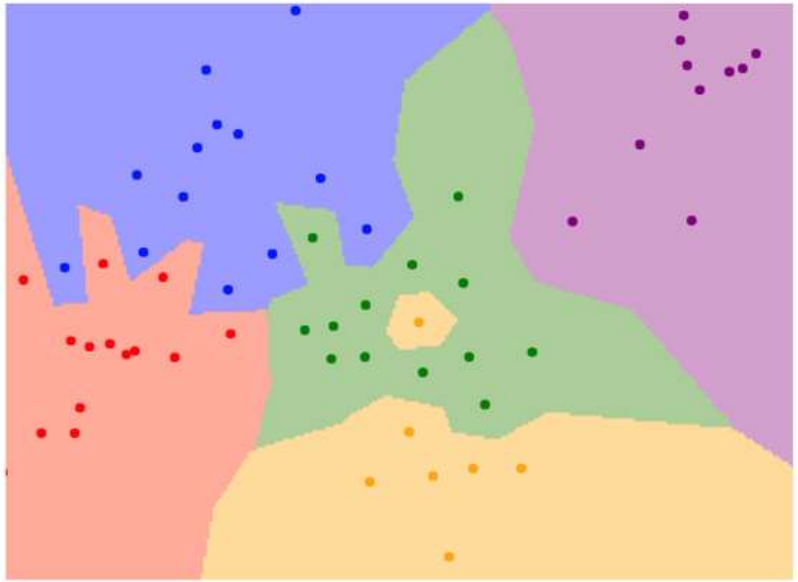
L1 (Manhattan) distance

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

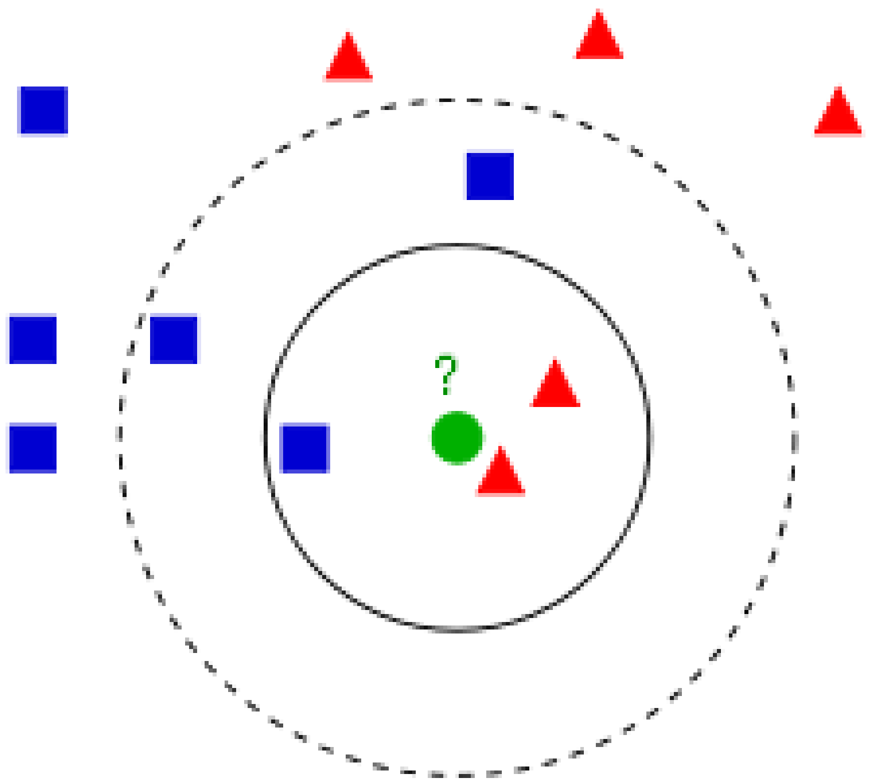


L2 (Euclidean) distance

$$d_1(I_1, I_2) = \left(\sum_p \left(I_1^p - I_2^p\right)^2\right)^{\frac{1}{2}}$$



K = 1



k值设置过小会降低分类精度；若设置过大且测试样本属于训练**集中包含数据**较少的类，则会增加噪声，降低分类效果

Hyperparameter tuning:

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

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

经验规则：**k一般低于**训练样本数的平方根

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

train	test

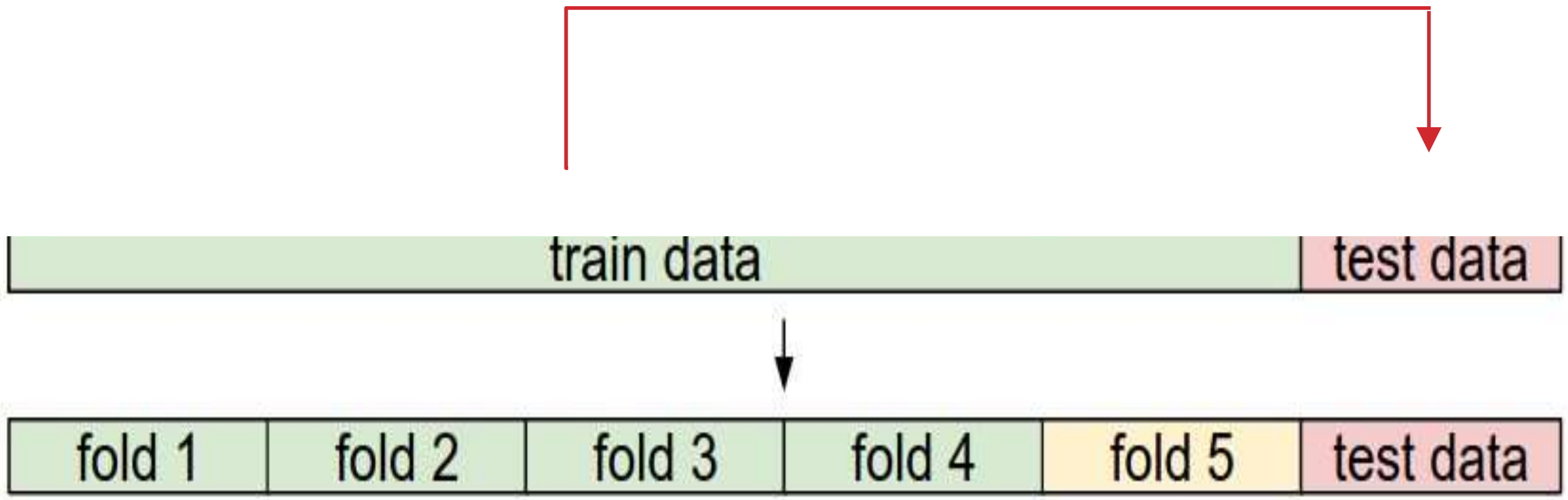
Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

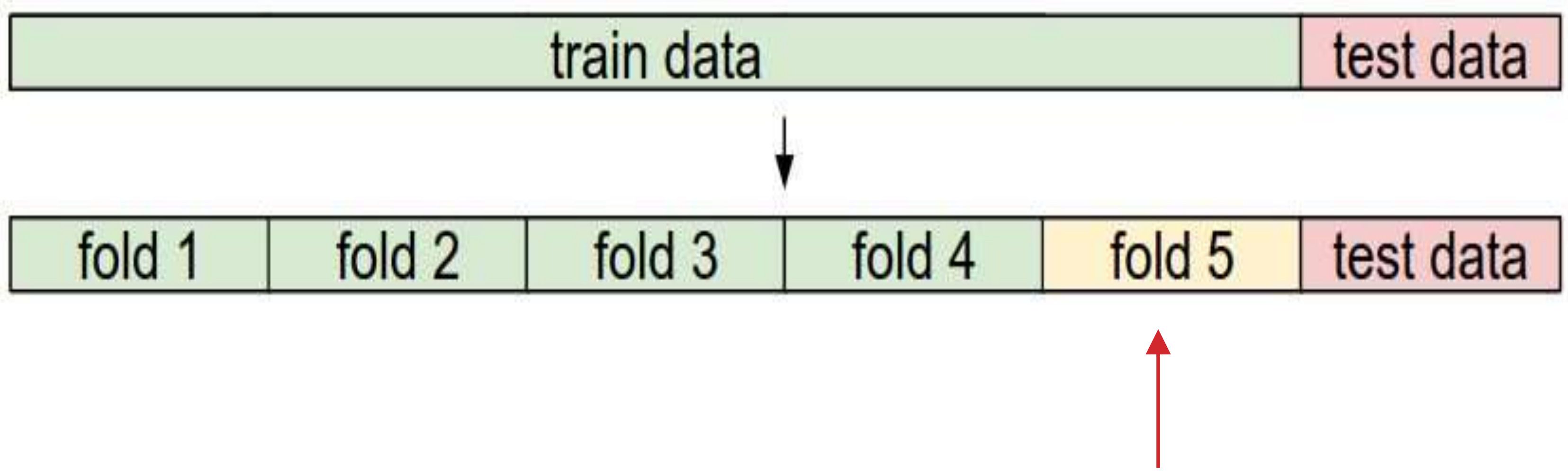
train	validation	test

Trying out what hyperparameters work best on test set:

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



Validation Set:

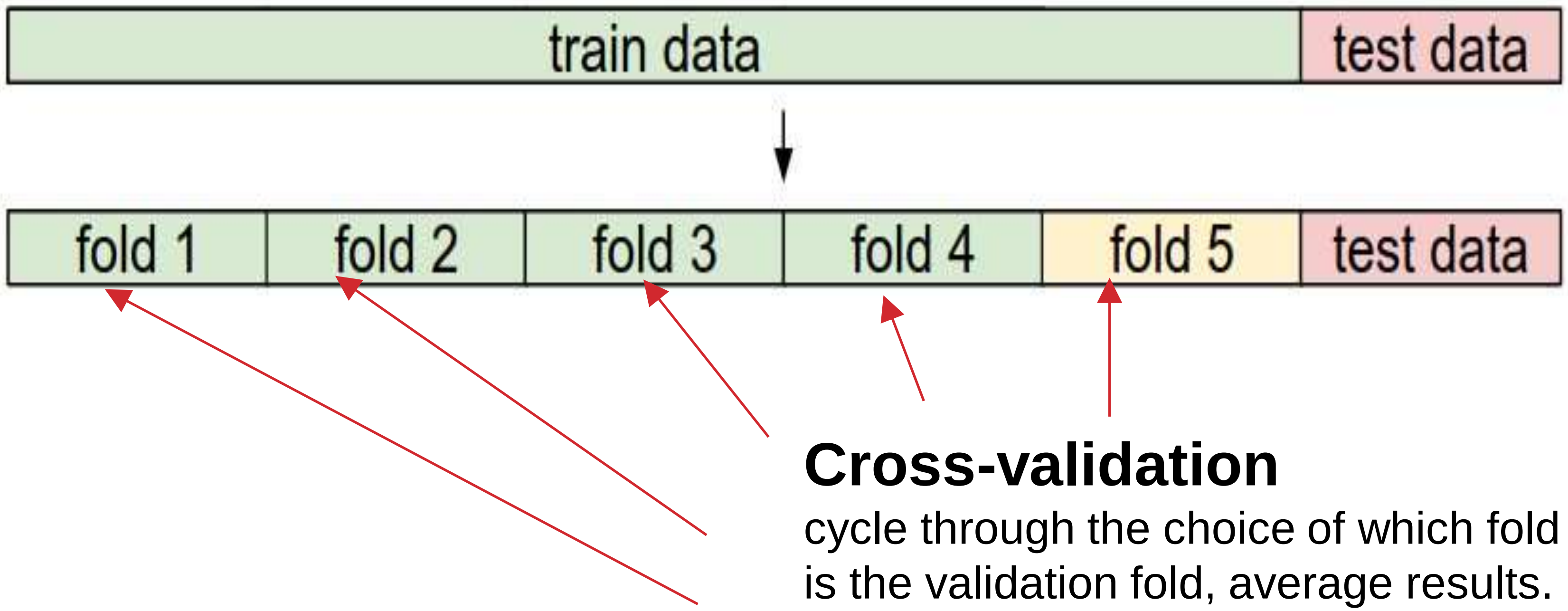


Validation data

use to tune hyperparameters

Validation Set:

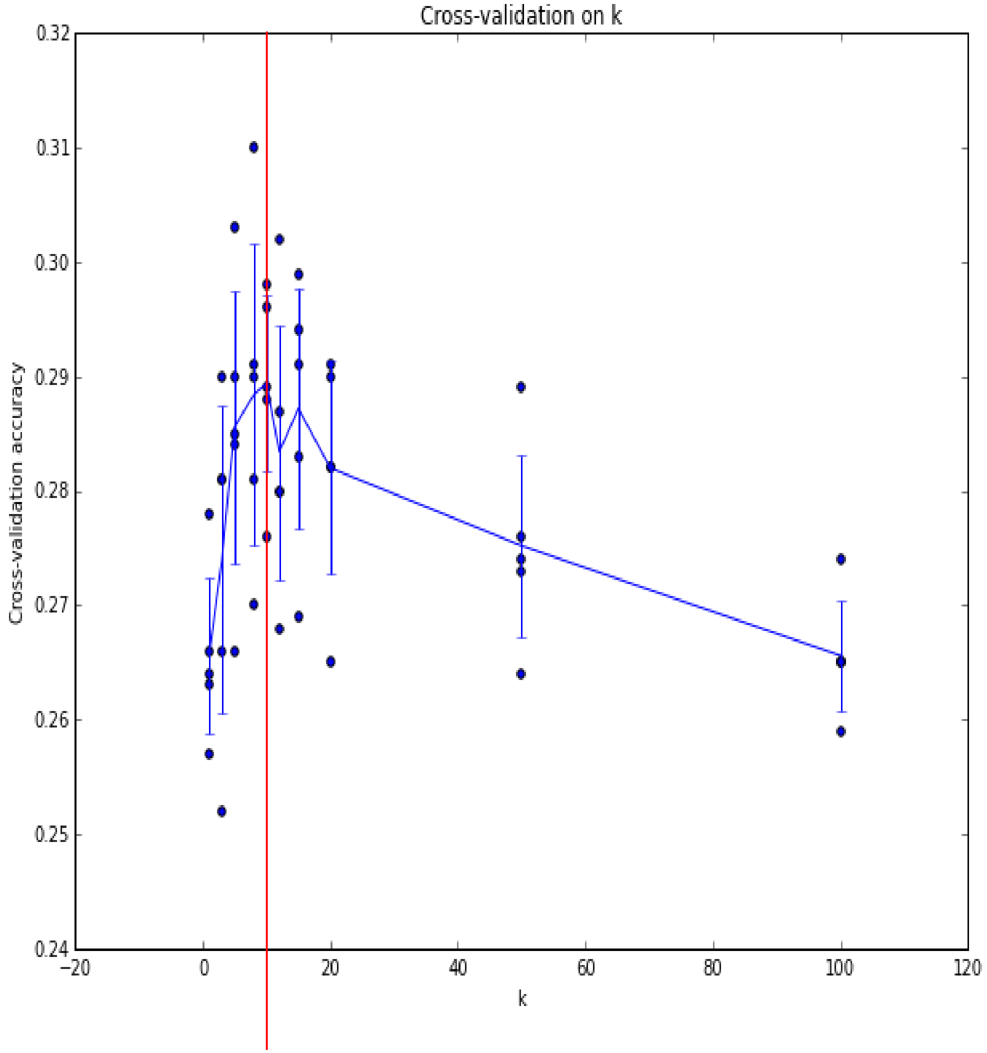
5-fold cross validation



Cross-validation

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

Hyperparameter tuning:



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

Each point: single
outcome.

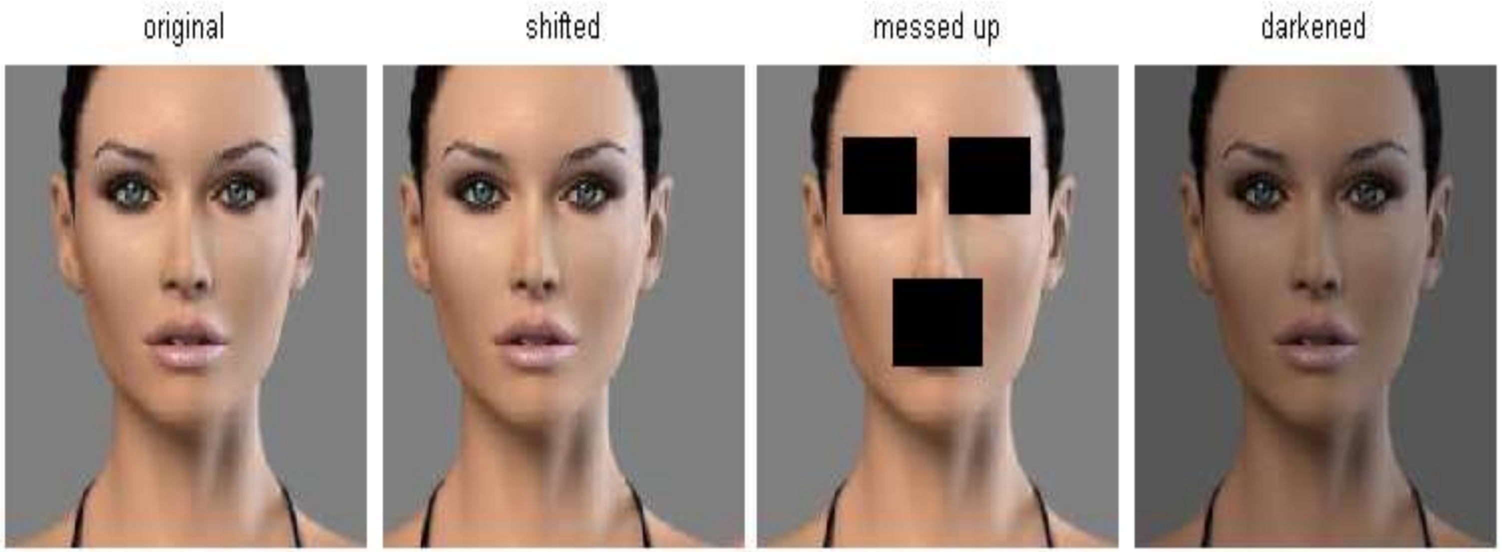
The line goes
through the mean, bars
indicated standard
deviation

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

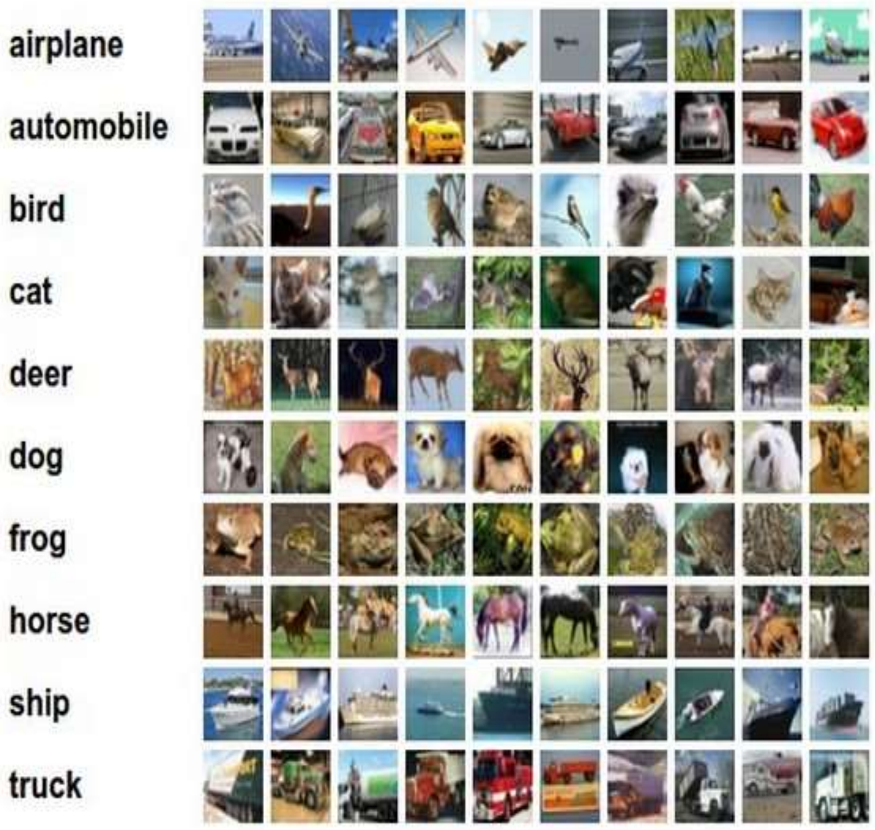
k-Nearest Neighbor on images **never used.**

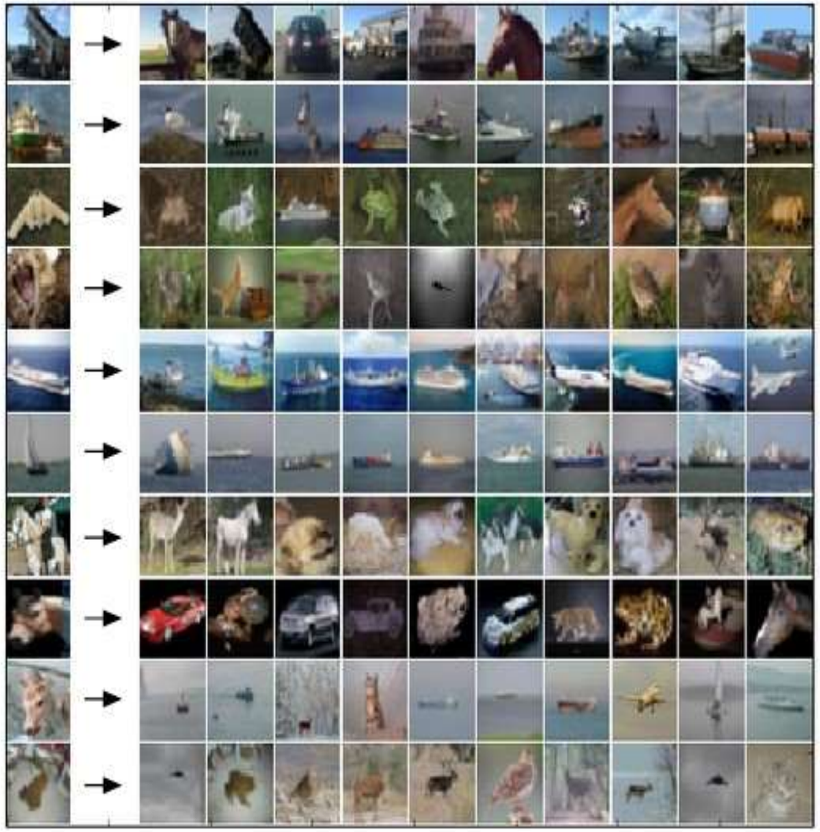
WHY?

- terrible performance at test time
- distance metrics on level of whole images can be very unintuitive



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


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

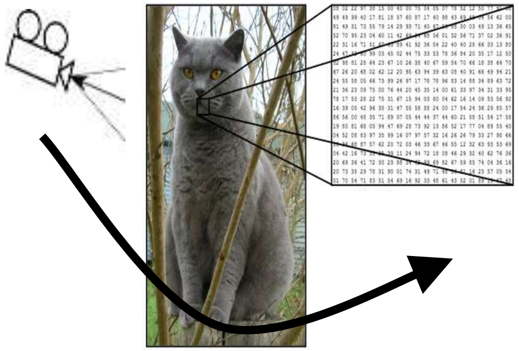


Summary

- **Image Classification:** given a **Training Set** of labeled images, predict labels on **Test Set**. Common to report the **Accuracy** of predictions (fraction of correct predictions)
- We introduced the **k-Nearest Neighbor Classifier**, which predicts labels based on nearest images in the training set
- We saw that the choice of distance and the value of k are **hyperparameters** that are tuned using a **validation set**, or through **cross-validation** if the size of the data is small.
- Once the best set of hyperparameters is chosen, the classifier is evaluated once on the test set, and reported as the performance of kNN on that data.

Recall from last time… Challenges in Visual Recognition

Camera pose



Illumination

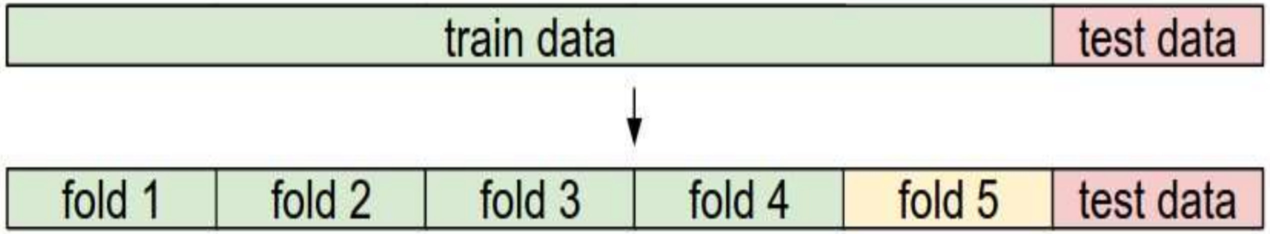
Deformation

Occlusion

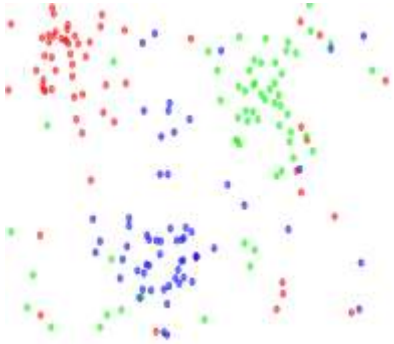
Background clutter

Intraclass variation

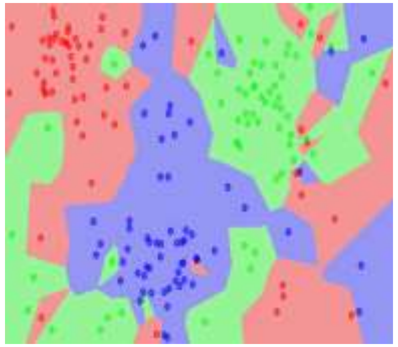
Recall from last time… data-driven approach, kNN



the data



NN classifier



5-NN classifier

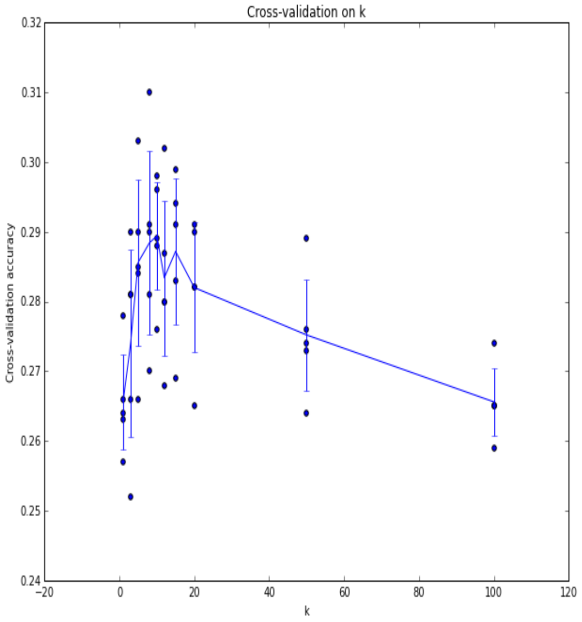
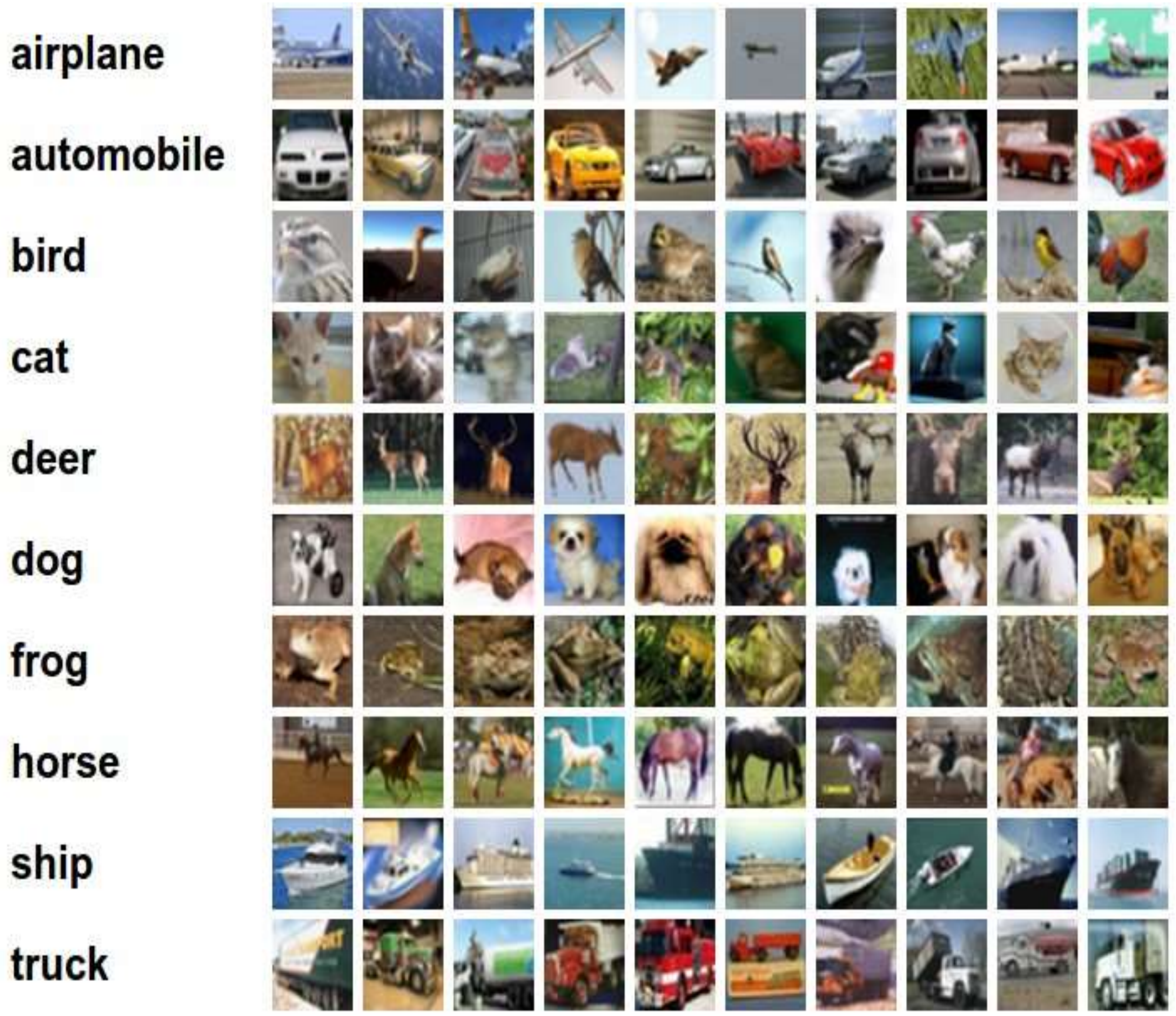


Image Classification



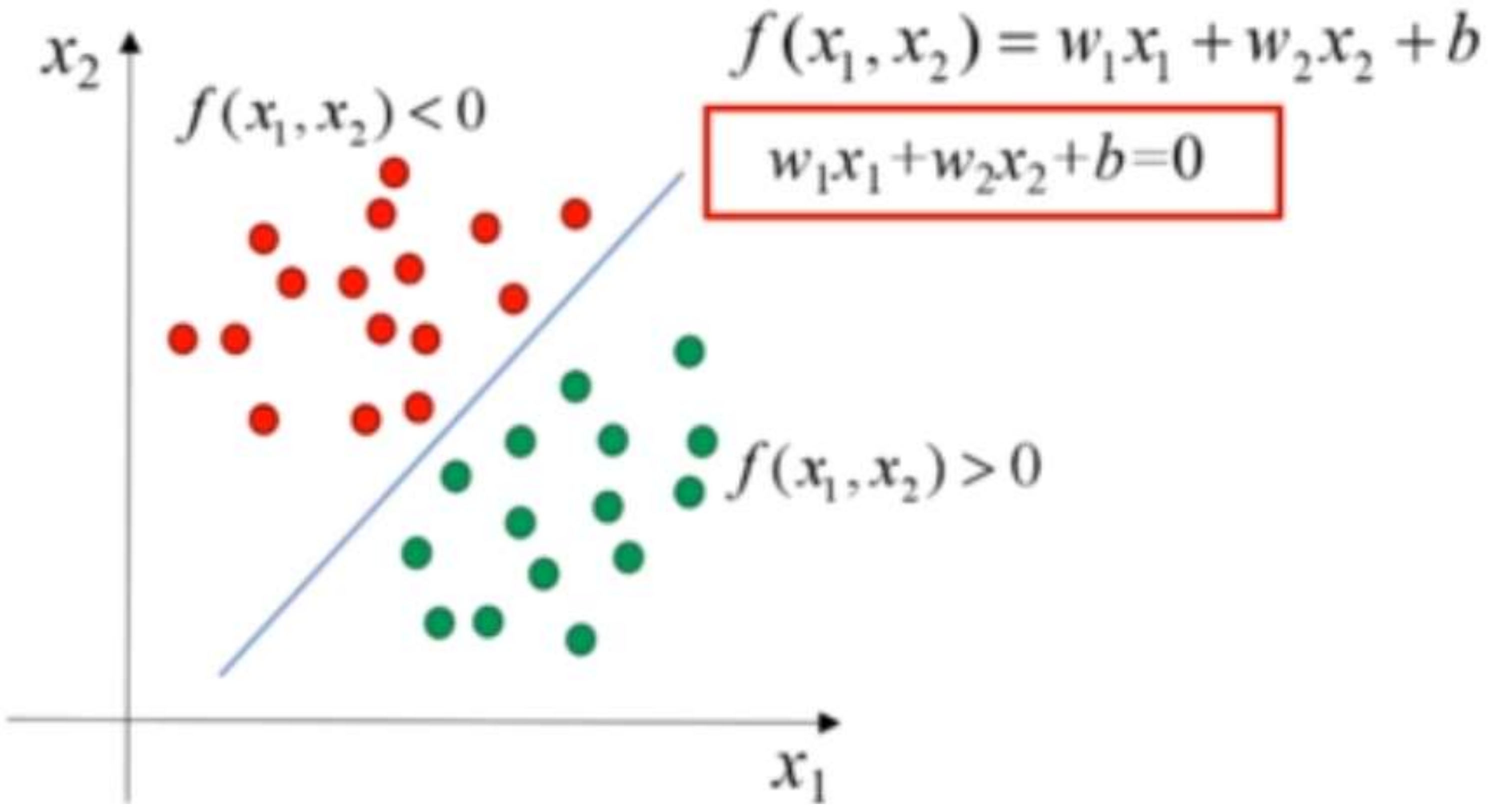
CIFAR-10
10 labels
50,000 training images
each image is **32x32x3**
10,000 test images.

如何才能学习到各部分的重要性呢？

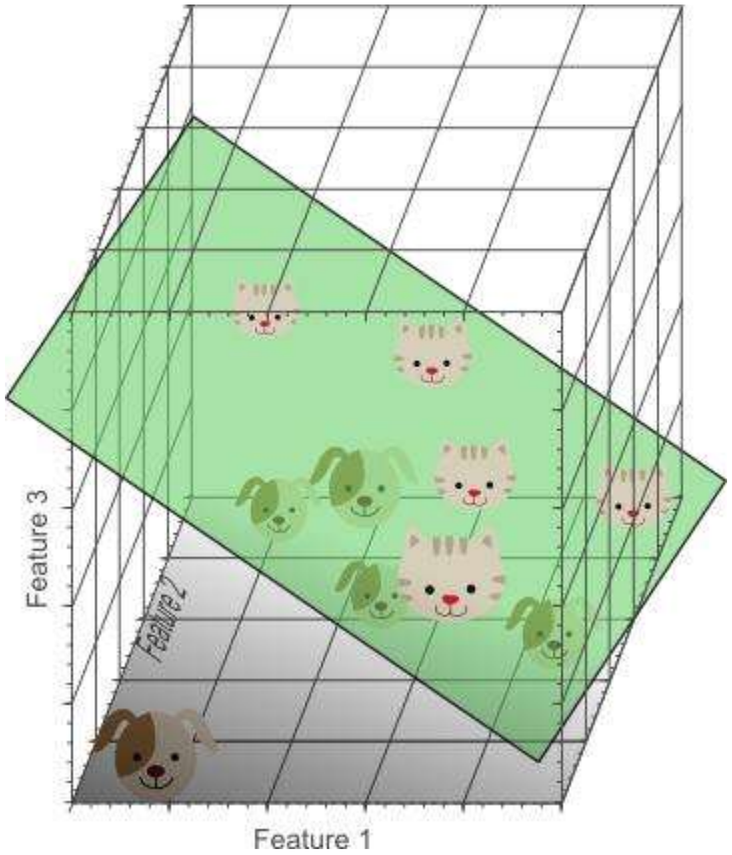
Neural Network

Linear classifiers

Linear Classifiers



Linear Classification



Linear Classifiers

[32x32x3]
array of numbers 0...1
(3072 numbers total)

image parameters

f(x,W)

10 numbers,
indicating class
scores

Parametric approach: **Linear classifier**

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

10 numbers,
indicating class
scores

[32x32x3]
array of numbers 0...1

Parametric approach: **Linear classifier**

$$\boxed{f(x, W)} = \boxed{W}\boxed{x}$$

3072x1

10x1 **10x3072**

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or “weights”

Parametric approach: **Linear classifier**

3072x1

$$\boxed{f(x,W)} = \boxed{W}\boxed{x} \qquad \boxed{(+b)}$$

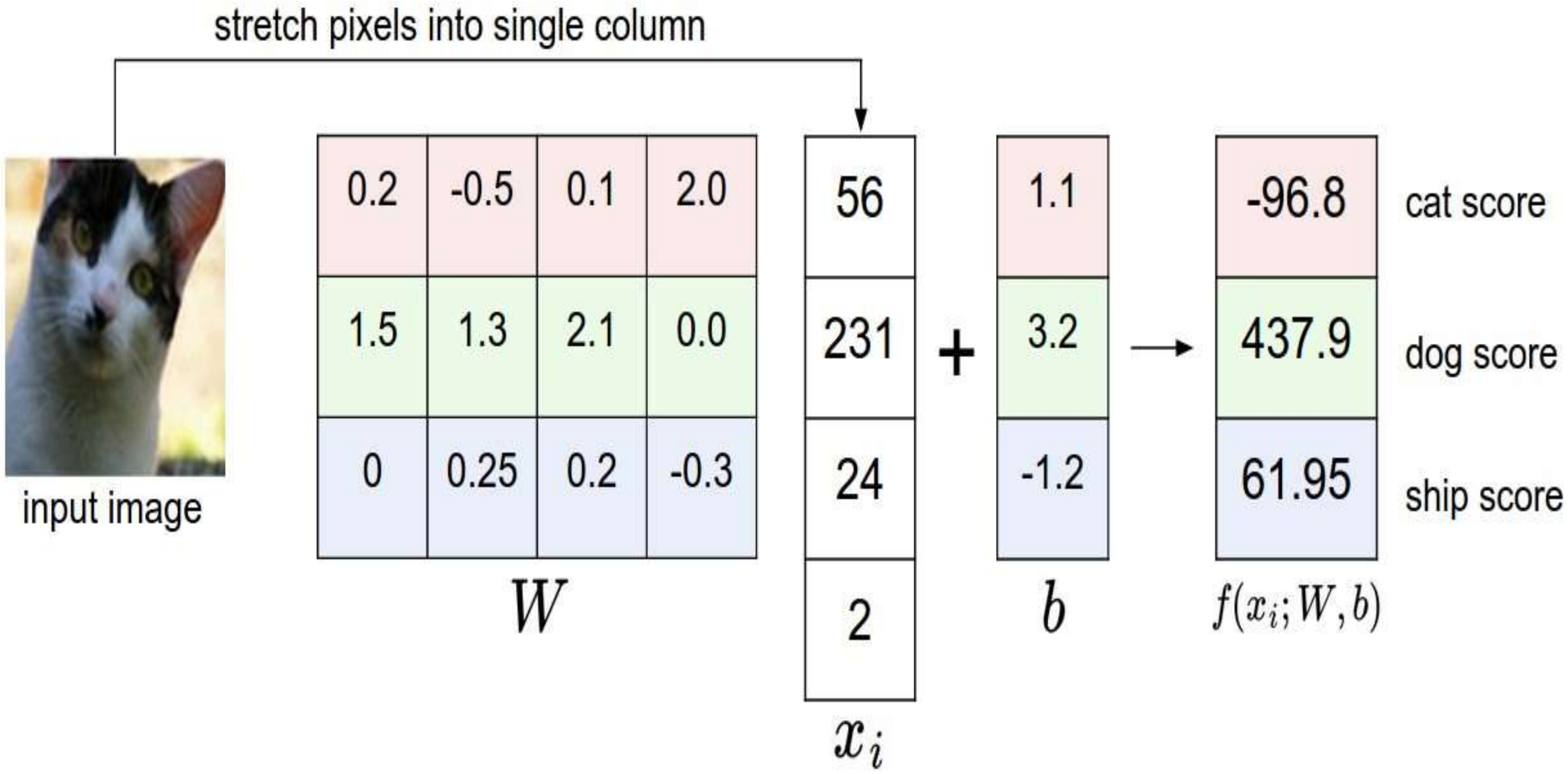
10x1 **10x3072** **10x1**

[32x32x3]
array of numbers 0...1

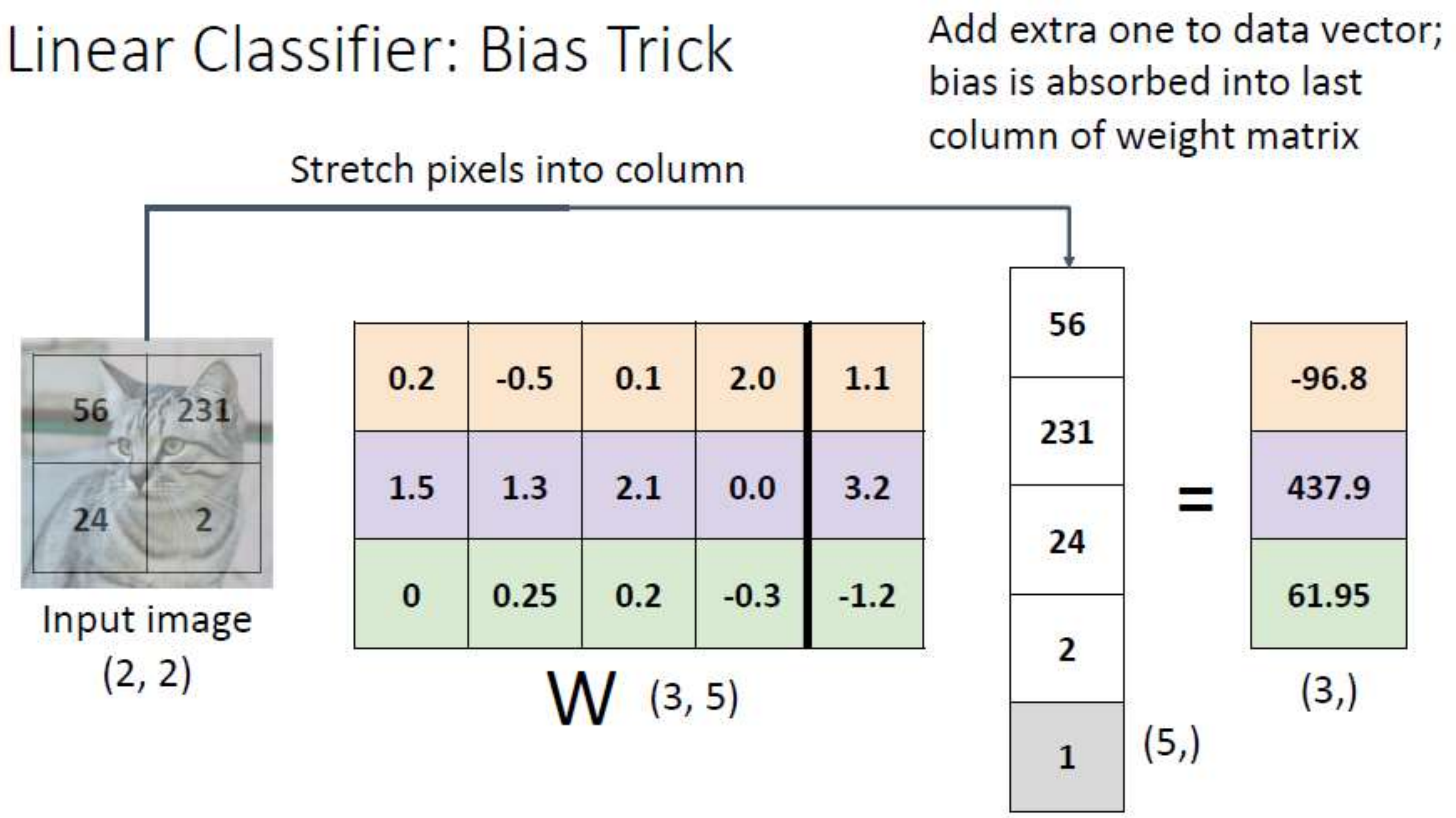
10 numbers, indicating class scores

parameters, or “weights”

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



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



Interpreting a Linear Classifier

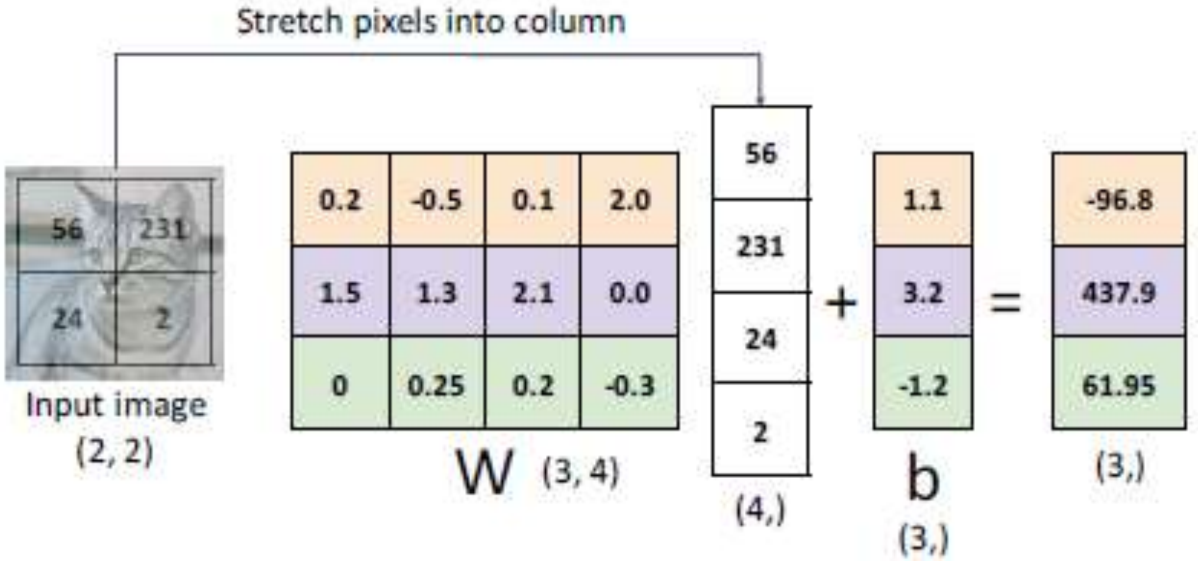
$$f(x_i, W, b) = Wx_i + b$$

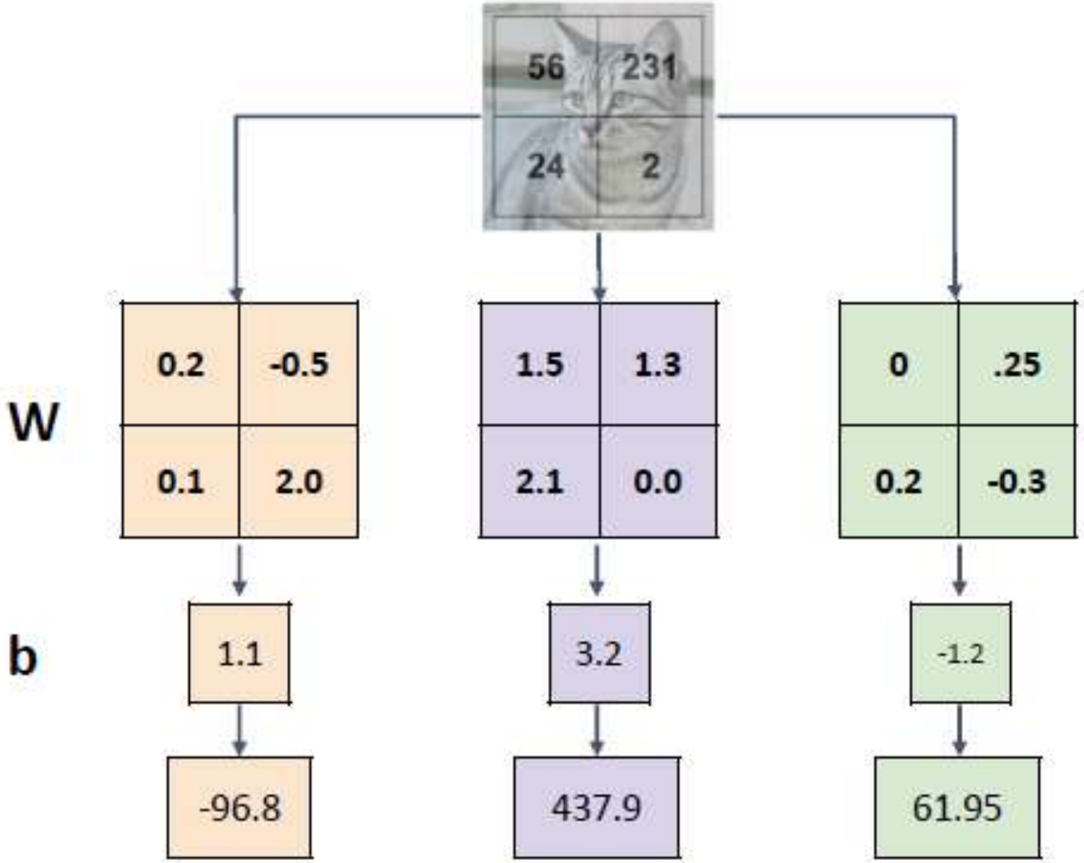
Q: what does the linear classifier do?

Interpreting a Linear Classifier

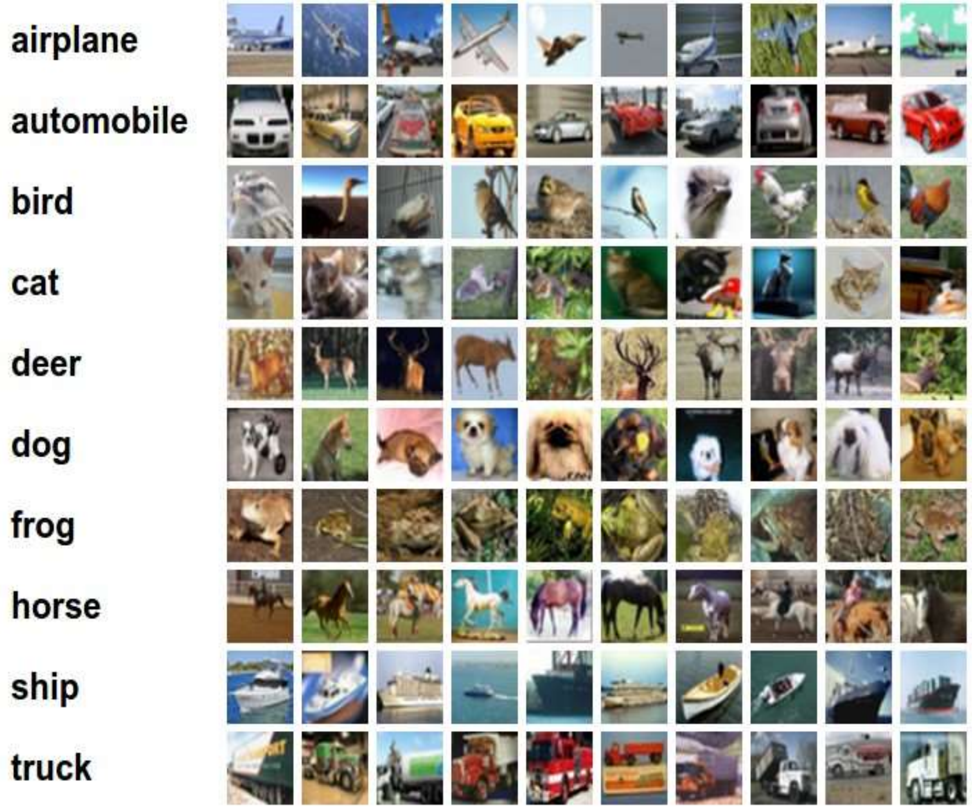
Algebraic Viewpoint

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





Interpreting a Linear Classifier(Visual Viewpoint)



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

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

Linear classifier has one "template" per category

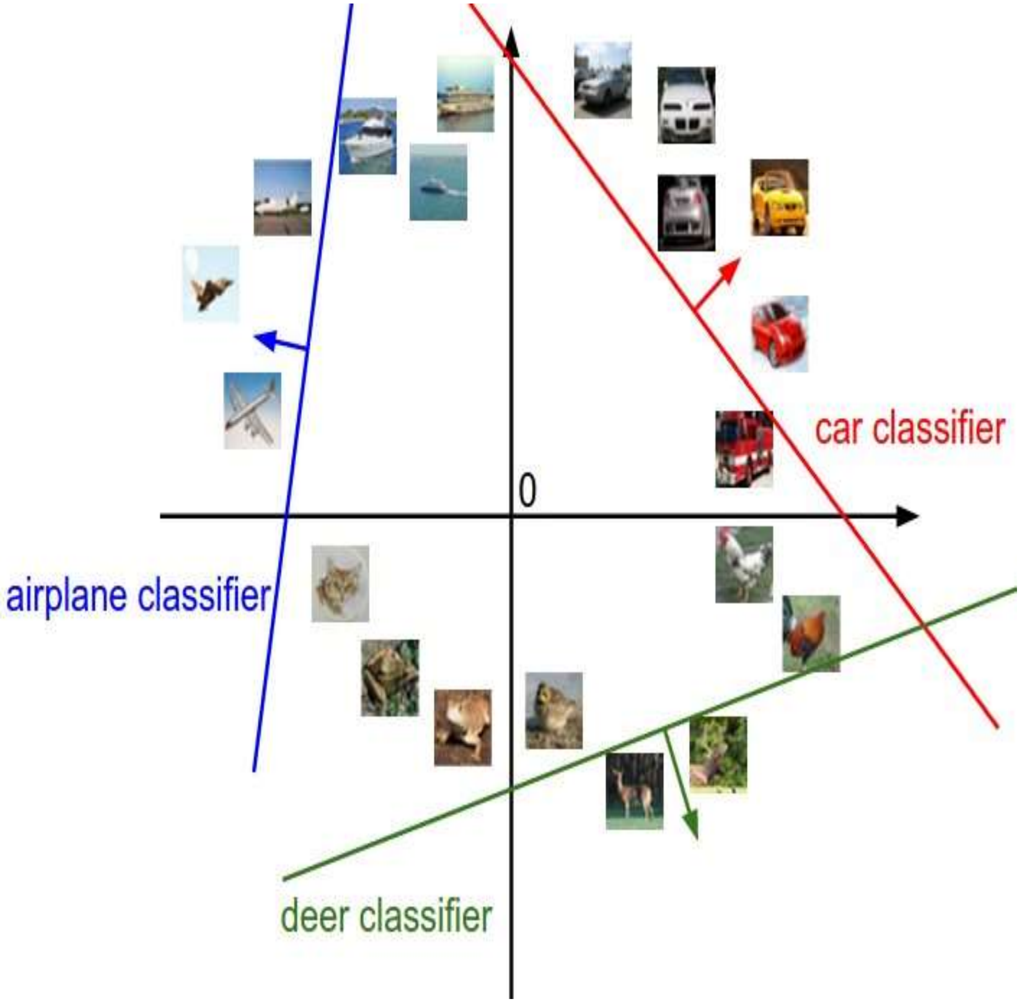
Interpreting a Linear Classifier(Visual Viewpoint)

horse template has 2 heads!

Matrix W of the horse（left）and the ship（right）

A single template cannot capture multiple modes of the data

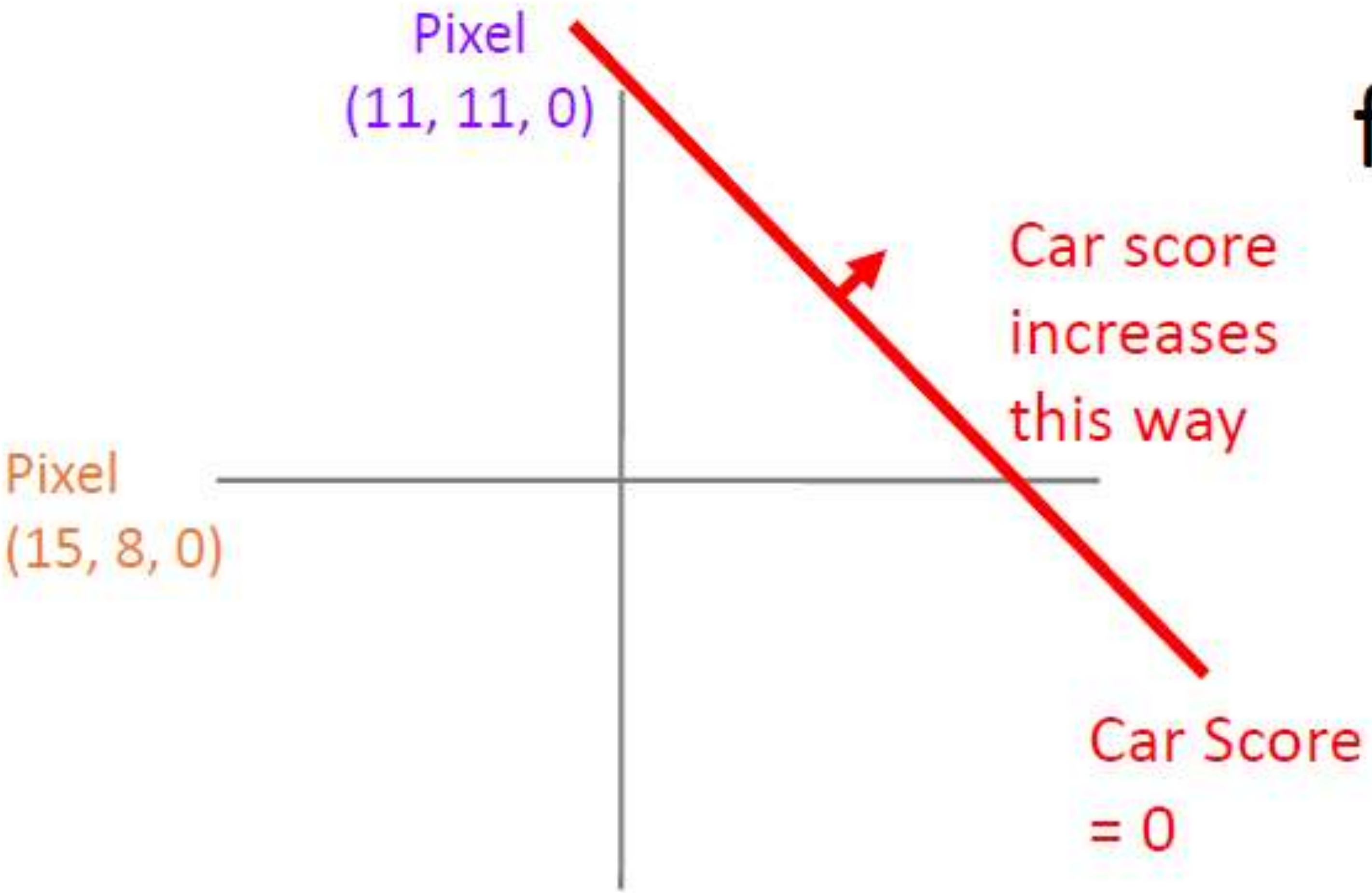
Interpreting a Linear Classifier(Geometric Viewpoint)



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

[32x32x3]
array of numbers 0...1
(3072 numbers total)

Interpreting a Linear Classifier(Geometric Viewpoint)



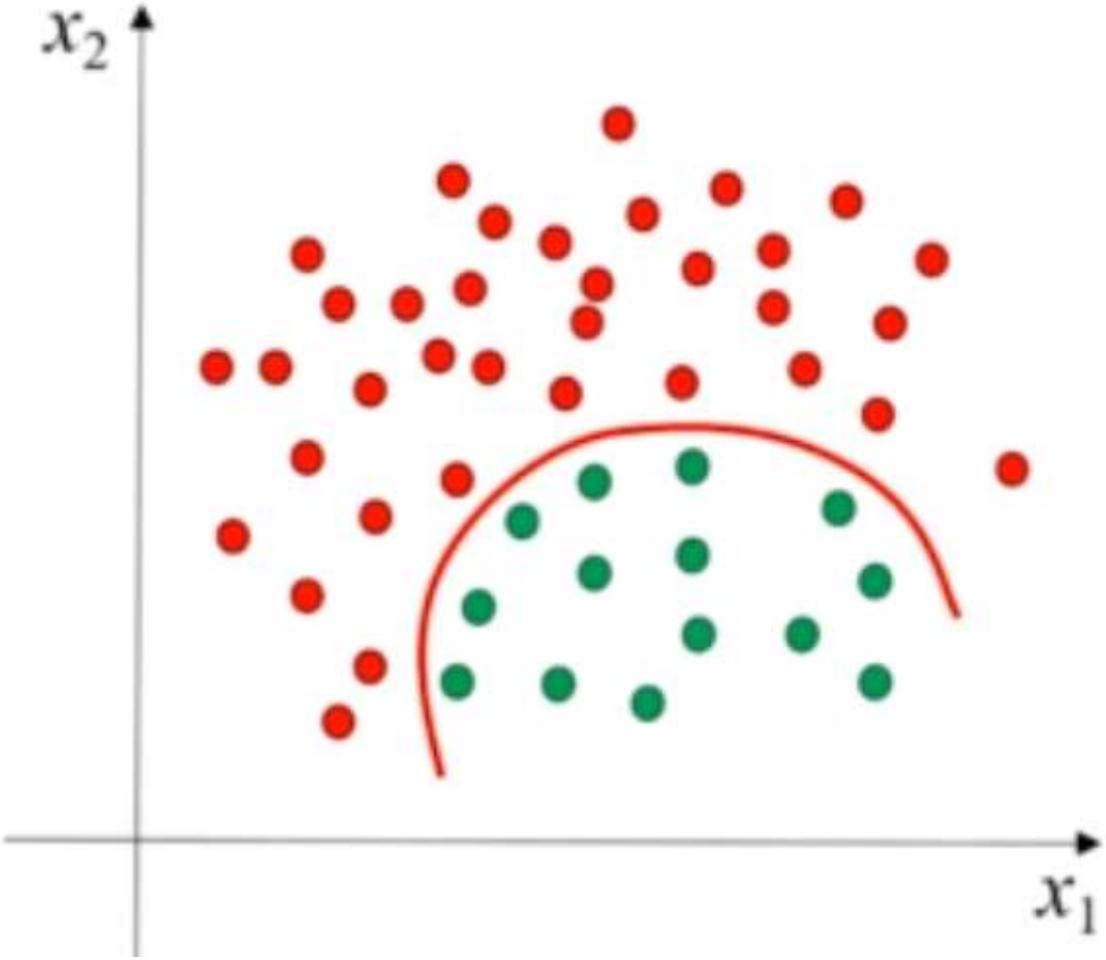
$$f(x,W) = Wx + b$$

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

可视化展示：A Neural Network Playground (tensorflow.org)

Linear Classifier

Drawbacks:



Going forward: Loss functions/optimization

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

- Given a W, we can compute class scores for an image x.
- But how can we actually choose a good W?

- Define a **loss function** that quantifies our unhappiness with the scores across the training data.
- Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

Loss functions

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

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

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

Low loss = good classifier

High loss = bad classifier

Loss functions

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

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

Given a dataset of examples

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

Where x_i is image and
y_i is (integer) label

Loss for a single example is

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

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

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

Loss functions

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



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

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector:

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

the SVM loss has the form:

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

Δ=1

Loss functions

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

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

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector:

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

the SVM loss has the form:

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

= 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

Loss functions

Multiclass SVM loss:

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



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

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:

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

= 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

Loss functions

Multiclass SVM loss:

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

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

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:

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

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 5.3) + max(0, 5.6)
= 5.3 + 5.6
= 10.9

Loss functions

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

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

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,
and using the shorthand for the scores vector:

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

the SVM loss has the form:

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

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

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

L = (2.9 + 0 + 10.9)/3 = **4.6**

Loss functions

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



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

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector:

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

the SVM loss has the form:

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

Q: what if the sum was instead over all classes? (including j = y_i)

Loss functions

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

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

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector:

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

the SVM loss has the form:

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

Q2: what if we used a mean instead of a sum here?

Loss functions

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



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

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector:

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

the SVM loss has the form:

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

Q3: what if we used a squared loss:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

Example numpy code:

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

```
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
```

Loss functions

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

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

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector:

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

the SVM loss has the form:

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

Q4: what is the min/max possible loss?

Loss functions

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

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

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector:

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

the SVM loss has the form:

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

Q5: usually at initialization W are small numbers, so all $s \approx 0$ What is the loss?

Loss functions

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

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

There is a bug with the loss:

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

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

Multiclass SVM loss:

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

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

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

Original Car Image Loss:
= 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

W twice as large:
= max(0, 2.6 - 9.8 + 1)
+max(0, 4.0 - 9.8 + 1)
= max(0, -6.2) + max(0, -4.8)
= 0 + 0
= 0

There is a bug with the loss:

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

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

E.g. Suppose that we found a W such that L = 0.
Is this W unique?

Choose W or 2W?

Weight Regularization

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

λ= regularization strength (hyperparameter)

损失函数=数据损失(Data loss)+正则化惩罚项(regularization)

- **Simple:**

✓ 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)

$R(W) = \beta \sum_k \sum_l W_{k,l}^2 + (1-\beta) \sum_k \sum_l |W_{k,l}|$

- **More complex:(will see later)**

✓ Dropout

✓ Batch normalization

✓ Cutout, Mixup, Stochastic depth, etc…

L2 regularization: motivation

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

$w_1 = [1,0,0,0]$ $\quad$ $w_2 = [0.25,0.25,0.25,0.25]$

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

Same predictions, so data loss will always be the same

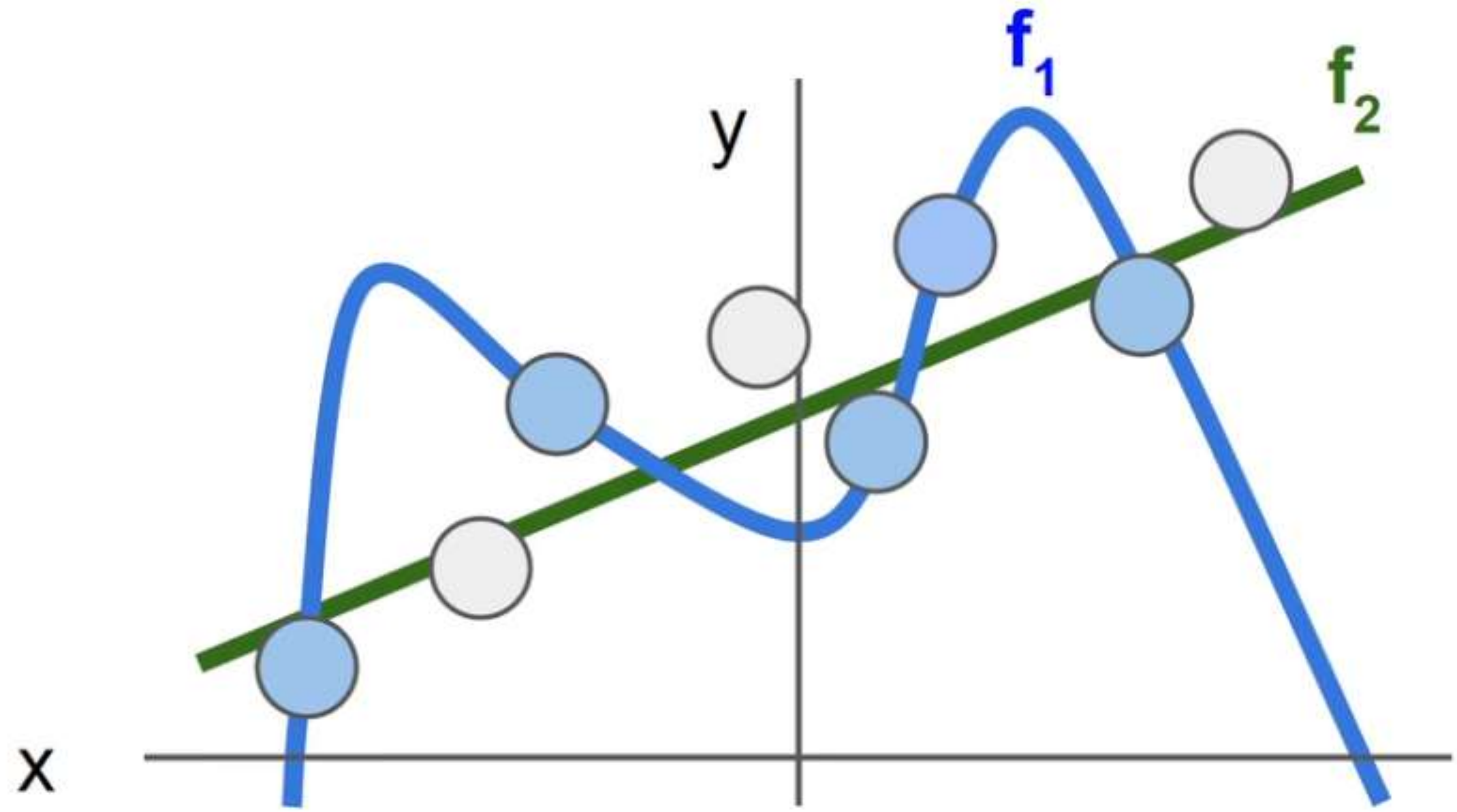
What will L2 regularization do?

What will L1 regularization do?

L2 regularization prefers weights to be “spread out”

Loss function consists of **data loss** to fit the training

data and **regularization** to prevent overfitting



Regularization pushes against fitting the data too well so we don’t fit noise in the data.

Multiclass SVM loss:

Note：Deep Learning using Linear Support Vector Machines

可视化展示： Multiclass SVM optimization demo (stanford.edu)

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

Scores = unnormalized log prob. of the classes

$$\boxed{s = f(x, W)}$$

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

Scores = unnormalized log prob. of the classes

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

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

Scores = unnormalized log prob. of the classes

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

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

Scores = unnormalized log prob. of the classes

$$\boxed{P(Y=k|X=x_i)=\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}}$$ where $$\boxed{s=f(x,W)}$$

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

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

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

Scores = unnormalized log prob. of the classes

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

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
交叉熵损失

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**		**24.5**
car	5.1	$\xrightarrow{\text{exp}}$	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

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

$L_i = -\log(0.13)$
= **2.04**

unnormalized log probabilities

probabilities

Softmax Classifier (Multinomial Logistic Regression)

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

Q: What is the min/max possible L_i ?

$0, \quad \infty$

unnormalized probabilities

	unnormalized log probabilities	$\xrightarrow{\text{exp}}$	$\xrightarrow{\text{normalize}}$ probabilities
cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

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

Softmax Classifier (Multinomial Logistic Regression)

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

Q2: At initialization all s will be approximate equal.What is the loss?

-log(1/C)
log(10) ≈ 2.3

unnormalized probabilities>=0

	unnormalized log probabilities	$\xrightarrow{\text{exp}}$	unnormalized probabilities>=0	$\xrightarrow{\text{normalize}}$	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

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

unnormalized log probabilities

probabilities

Softmax vs. SVM

单个损失：

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

总损失（交叉熵损失）：

$$\mathrm{P_i} = \frac{e^{f(x_i,W)_{y_i}}}{\sum_j e^{f(x_i,W)_j}}$$

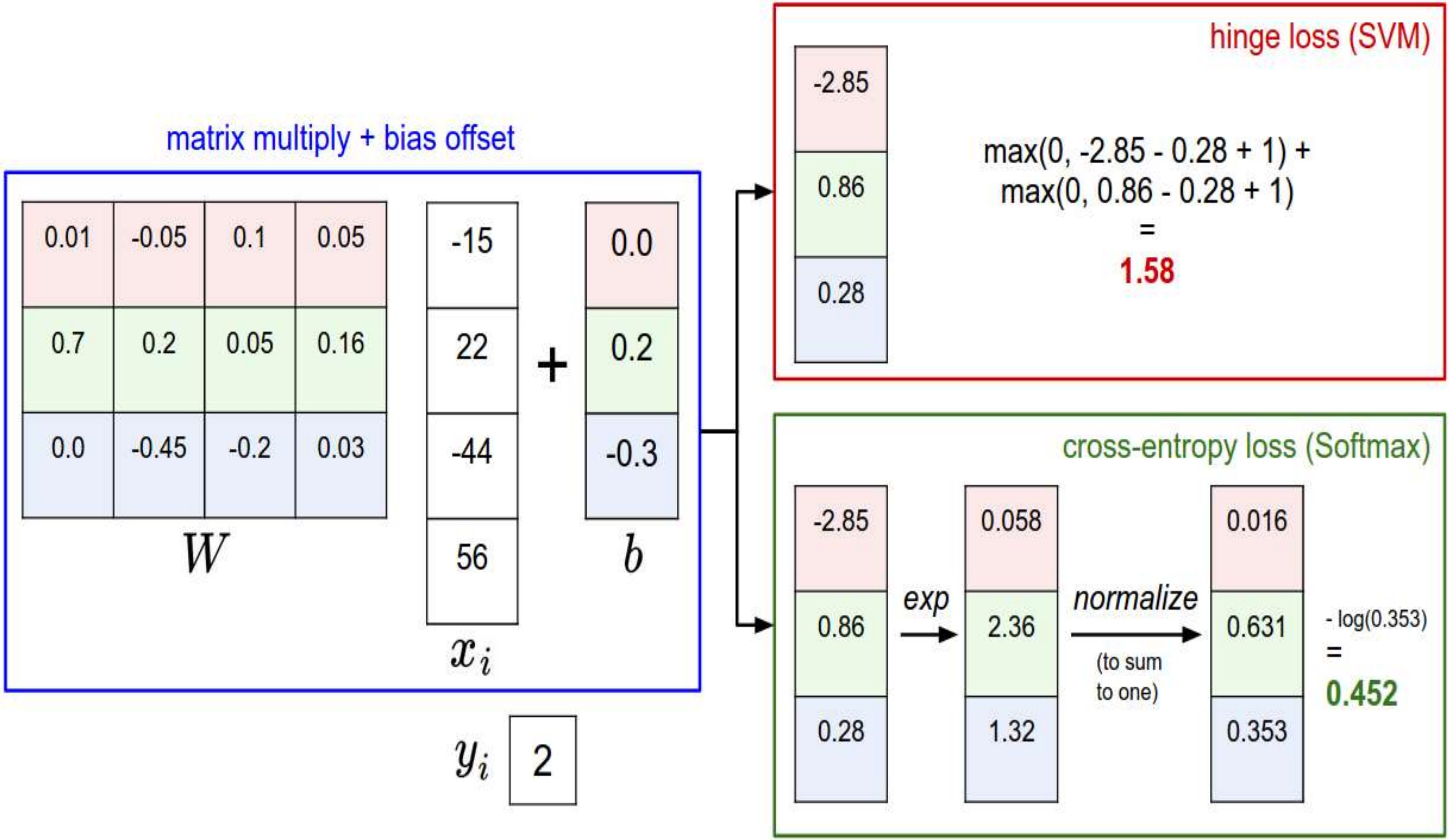
$$L = -\sum_{i=1}^{N} \mathrm{P_i}\log(\mathrm{P_i}) + \lambda R(W)$$

单个损失：

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

总损失：

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



matrix multiply + bias offset
0.01
-0.05
0.1
0.05
0.7
0.2
0.05
0.16
0.0
-0.45
-0.2
0.03
W
-15
22
-44
56
x_i
+
0.0
0.2
-0.3
b
y_i
2
hinge loss (SVM)
-2.85
0.86
0.28
max(0, -2.85 - 0.28 + 1) +
max(0, 0.86 - 0.28 + 1)
=
1.58
cross-entropy loss (Softmax)
-2.85
0.86
0.28
exp
0.058
2.36
1.32
normalize
(to sum
to one)
0.016
0.631
0.353
- log(0.353)
=
0.452

Softmax vs. SVM

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

---

assume scores:
[10, -2, 3]
[10, 9, 9]
[10, -100, -100]
and

$$\boxed{y_i = 0}$$

Q: Suppose I take a datapoint and I jiggle a bit (changing its score slightly). What happens to the loss in both cases?

Summary

- **Linear classifiers:** The simplest ***parametric*** classifier.
- You must define a loss function in order to optimize the weights of a linear classifier. We first described **SVM loss**.
- Certain loss functions (esp. SVM loss) underconstrain the model parameters. **Regularization** can fix this.
- To produce a continuous prediction (probability of class membership) we described the **softmax loss**.
- Finally we explored these methods with an **interactive visualization**.