

Lecture 2: Supervised Learning

Piazza

- Everyone needs to sign up for Piazza. This is how you get messages for the class.
- Please post most questions there, rather than sending email to me
- The TAs and I will answer questions posted to Piazza.

Python: Up and running

- **We’re using Python 3.6, not 2.7.**
- If you have not installed Python and tried a simple program yet, please do so as soon as possible.
- Try loading up the first Python Notebook for Homework 1. Even if you don’t have time to do the assignment yet, at least make sure the Python Notebook is working properly. This will make sure you don’t incur a major delay later.
- PYTHON Tutorial! See “Notes” tab of course web page.
(Show Notes page).

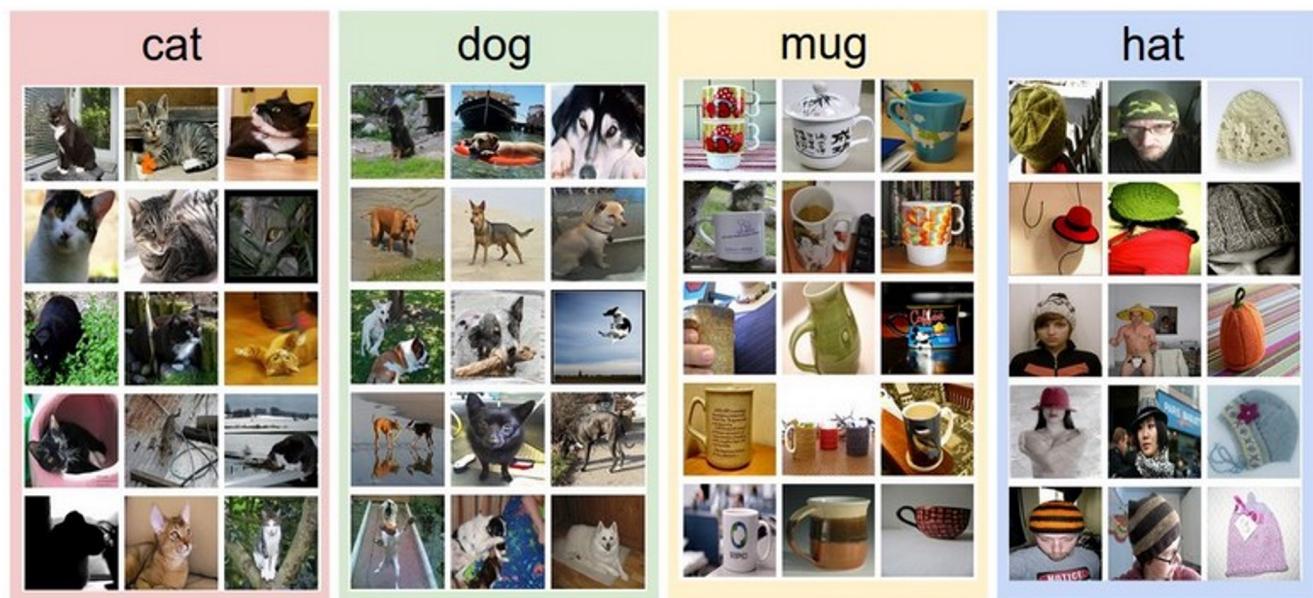
Back to classification...

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
```

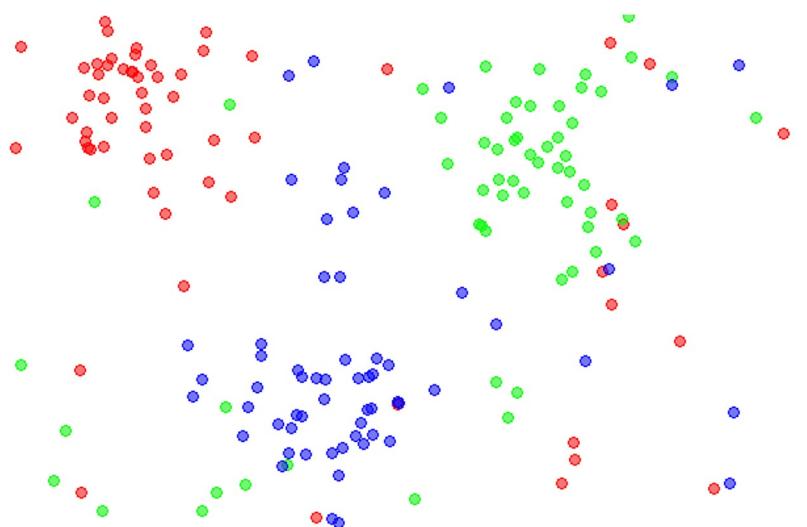
Example training set



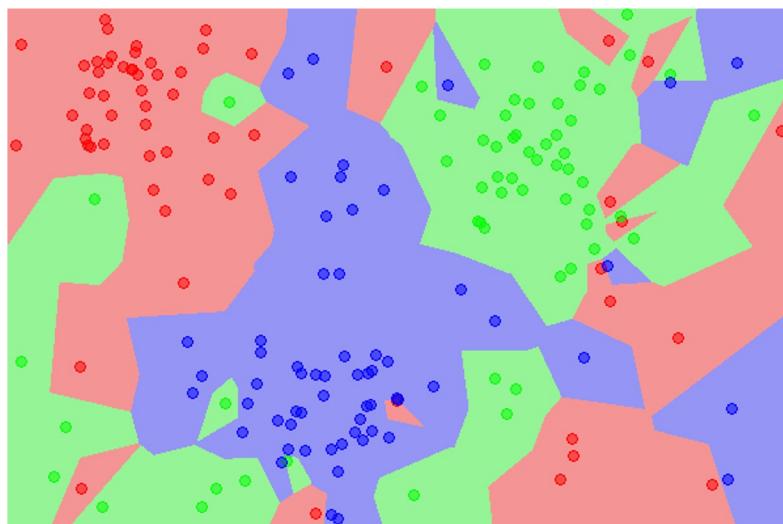
k-Nearest Neighbor

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

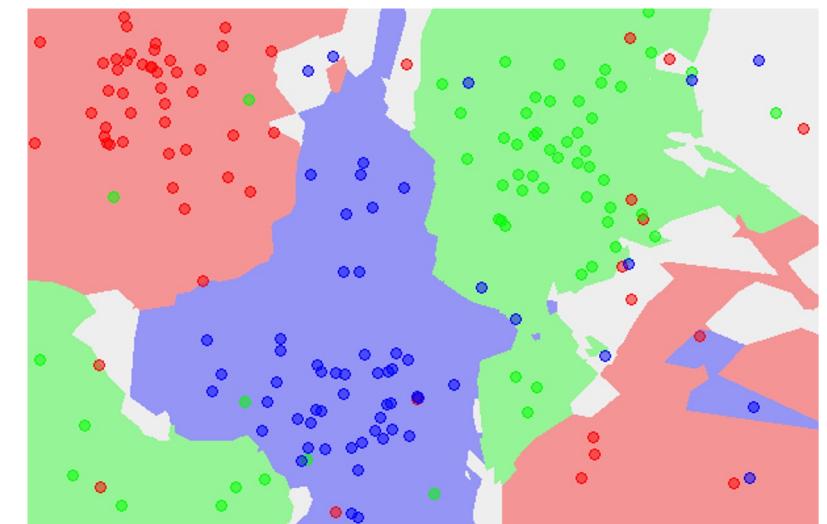
the data



NN classifier



5-NN classifier



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

Example dataset: CIFAR-10

10 labels

50,000 training images

10,000 test images.

airplane



automobile



bird



cat



deer



dog



frog



horse



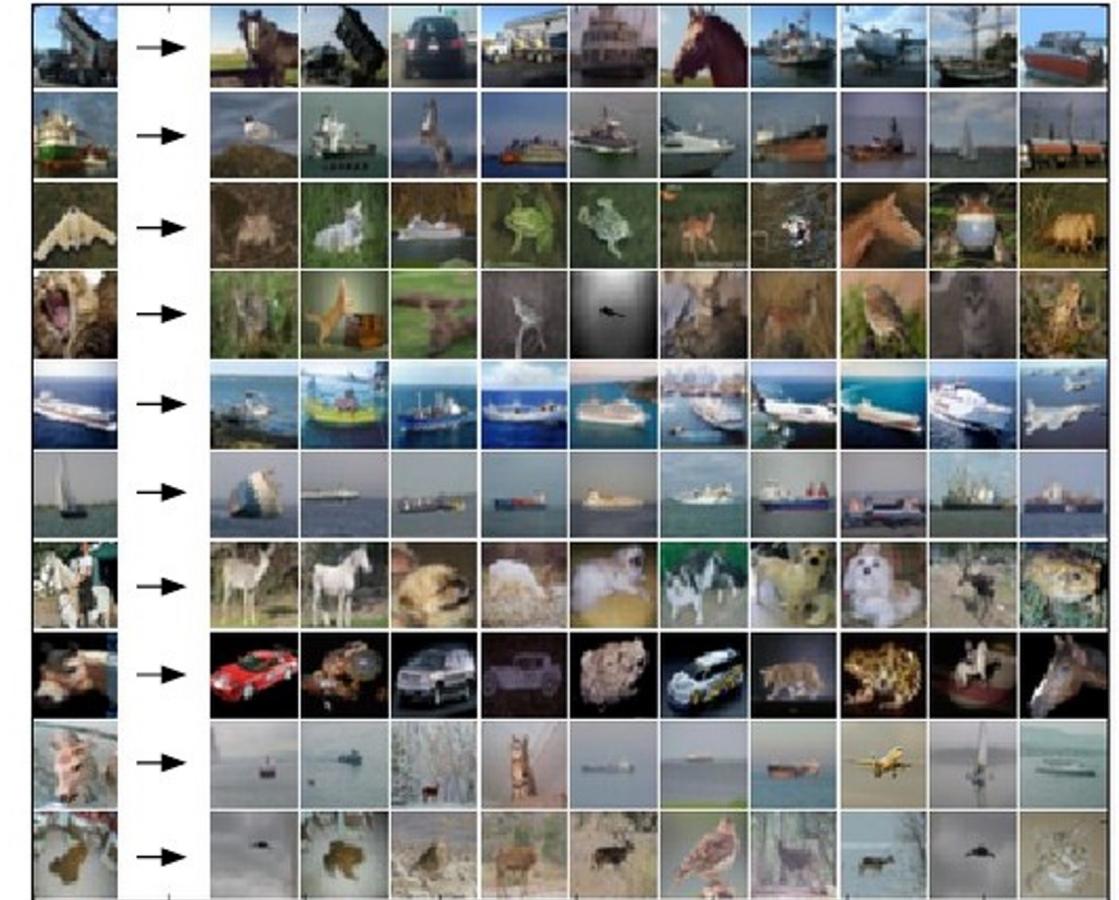
ship



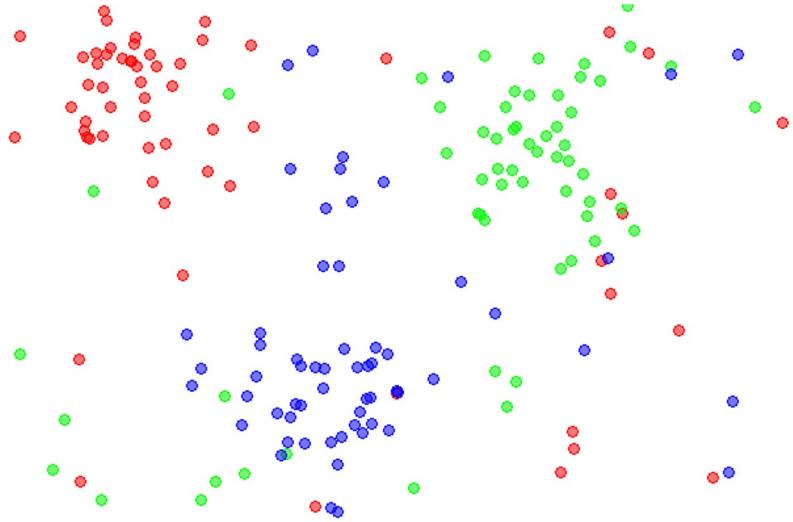
truck



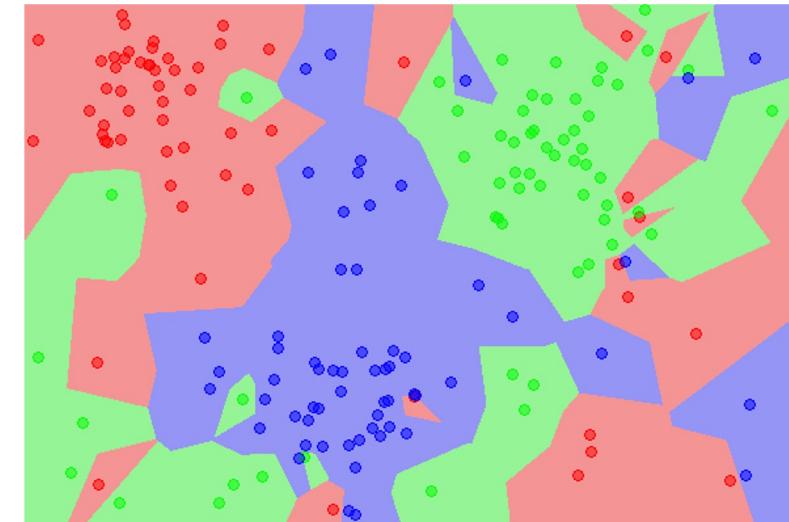
For every test image (first column),
examples of nearest neighbors in rows



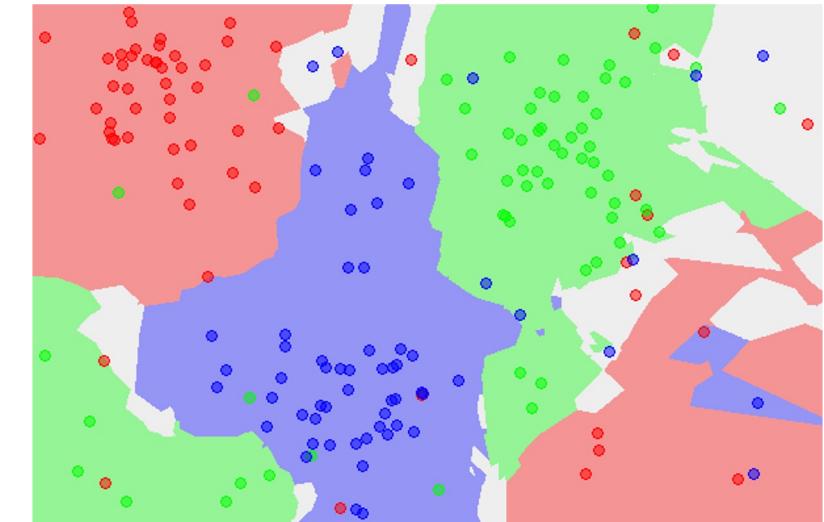
the data



NN classifier

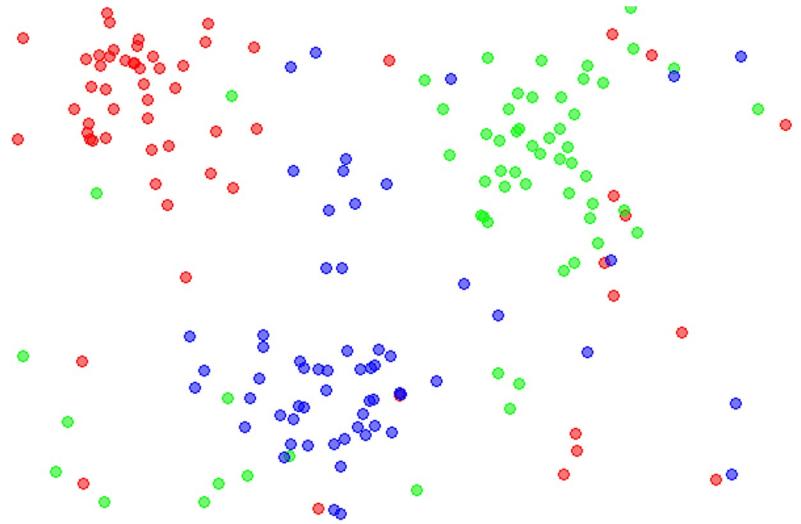


5-NN classifier

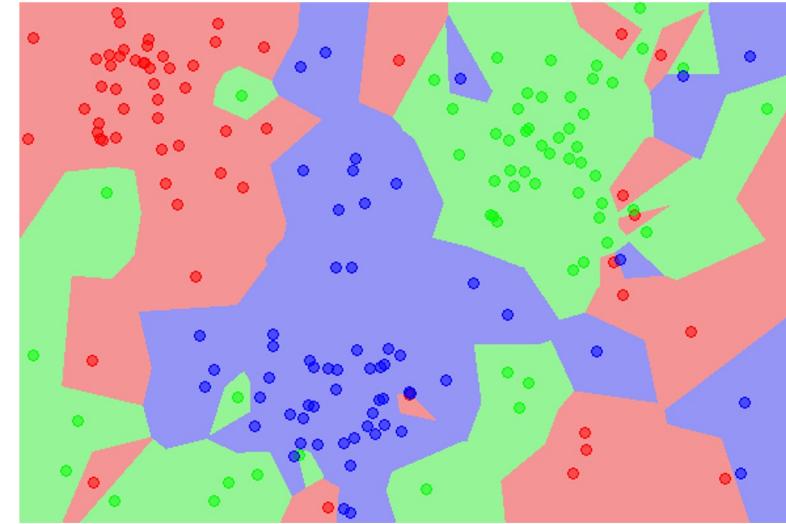


Q: what is the accuracy of the nearest neighbor classifier on the training data, when using the Euclidean distance?

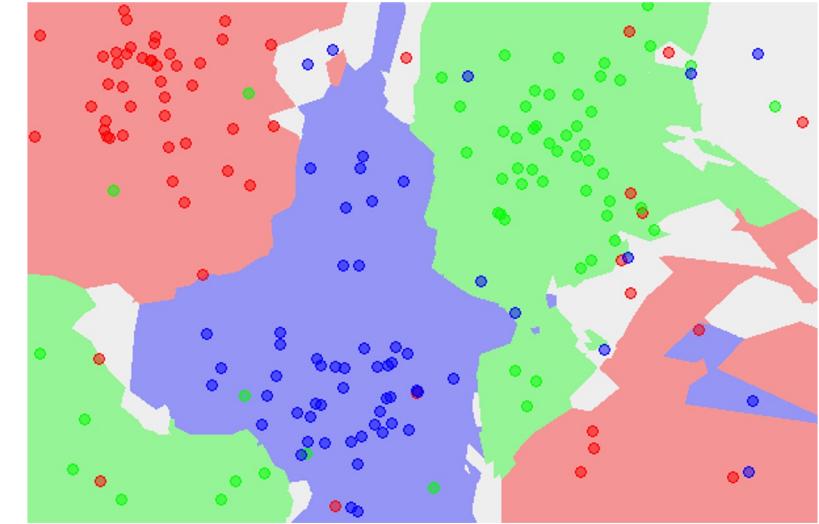
the data



NN classifier



5-NN classifier



Q2: what is the accuracy of the k -nearest neighbor classifier on the training data?

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

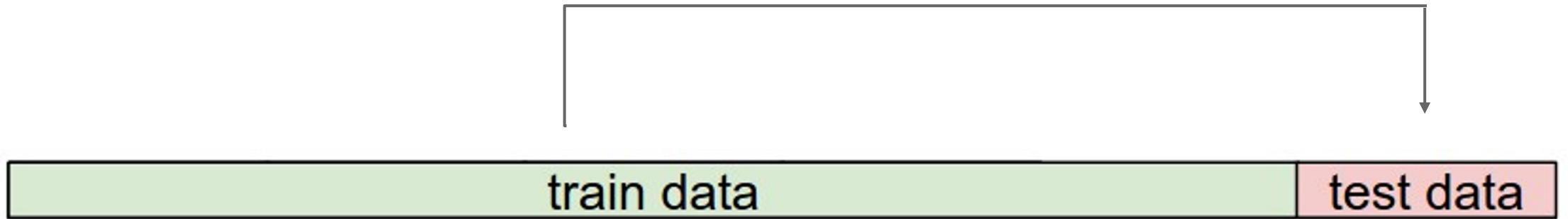
i.e. how do we set the hyperparameters?

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

i.e. how do we set the hyperparameters?

Very problem-dependent.
Must try them all out and see what works best.

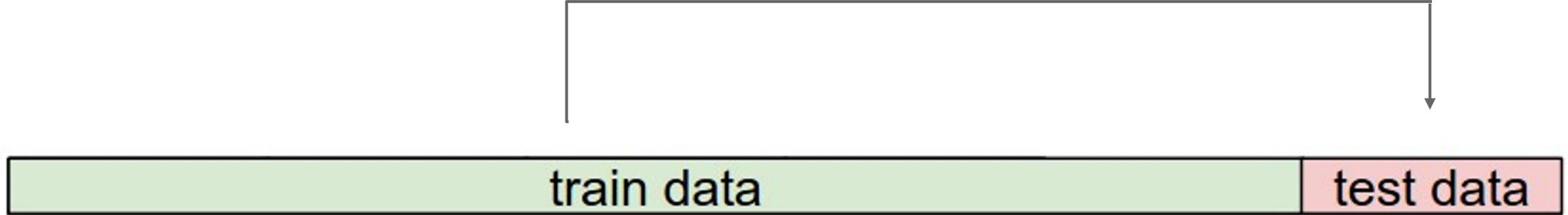
Try out what hyperparameters work best on test set.

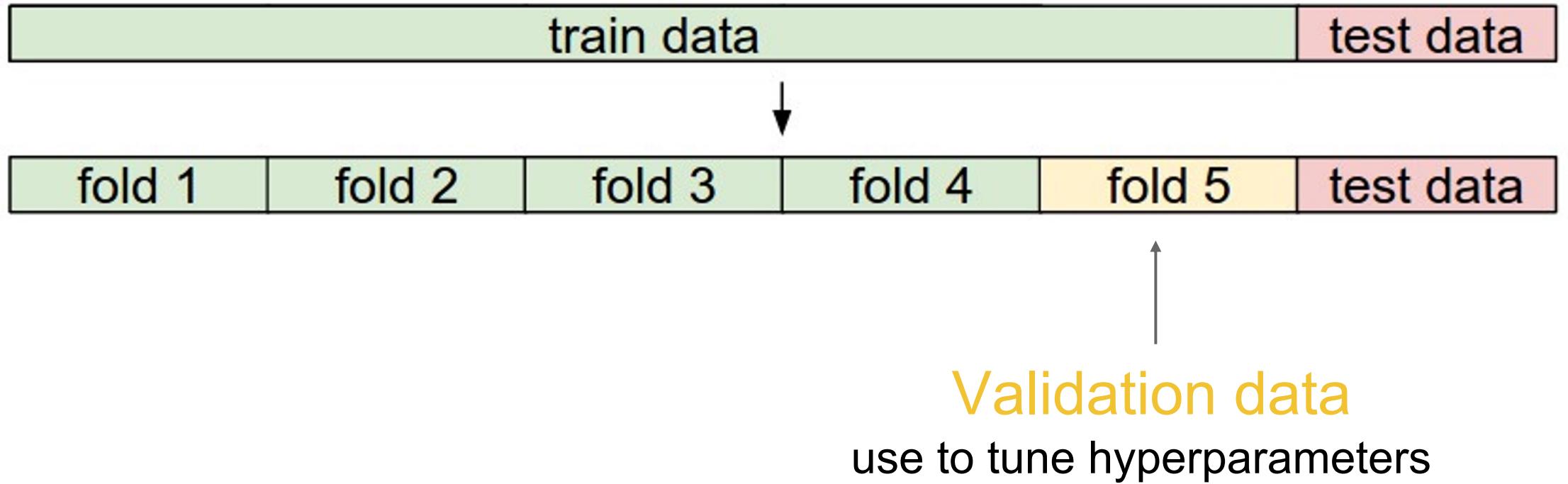


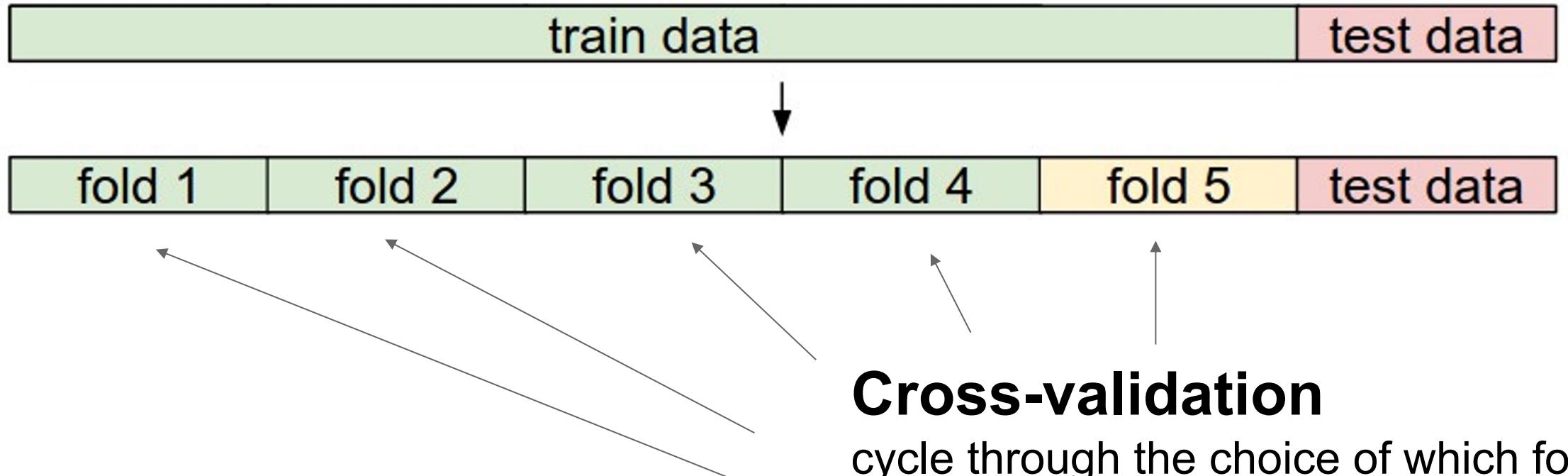
Trying out what hyperparameters work best on test set:

Very bad idea. The test set is a proxy for the generalization performance!

Use only **VERY SPARINGLY**, at the end.

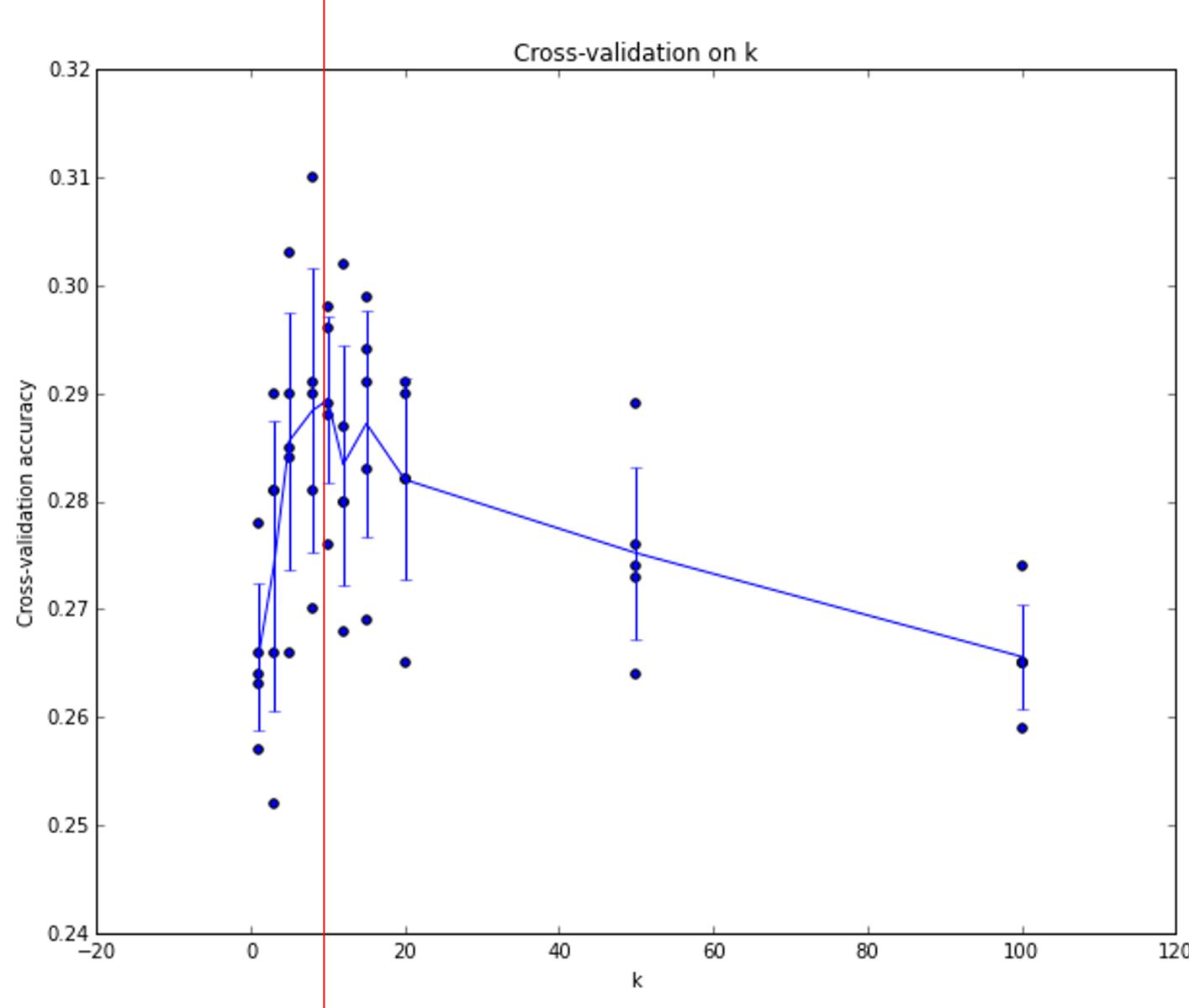






Cross-validation

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



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

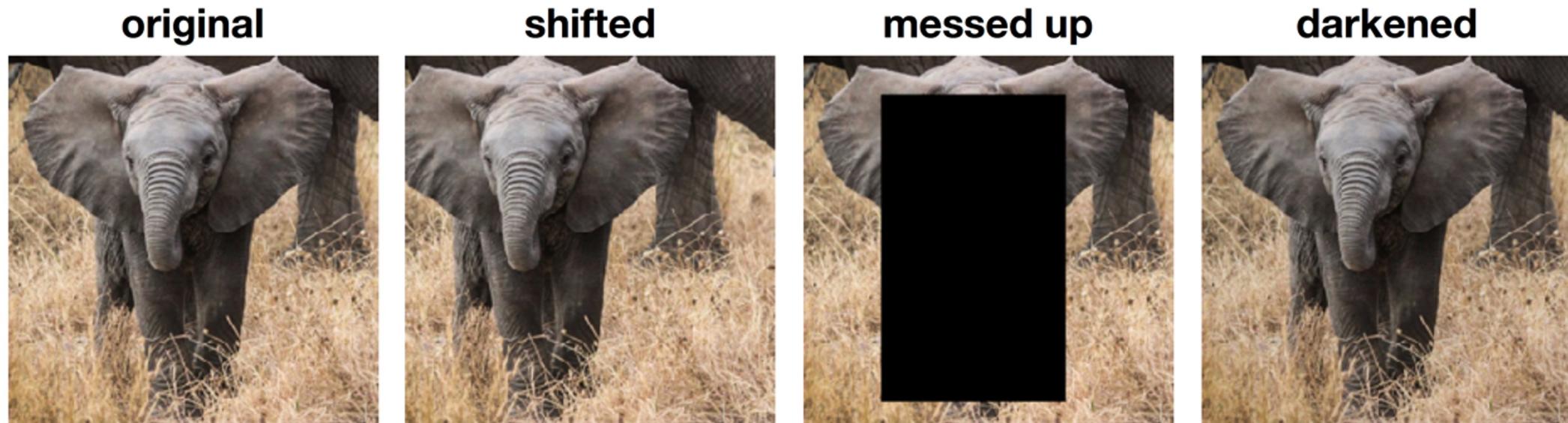
Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

(Seems that $k \approx 7$ works best
for this data)

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

- 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)

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

L1 distance:

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

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

pixel-wise absolute value differences

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

$\frac{\text{ad}}{d} \rightarrow 456$

```

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

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

```

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 every test image:
- find nearest train image with L1 distance
 - predict the label of nearest training image

```

import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

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

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

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

        return Ypred

```

Nearest Neighbor classifier

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

Do a poll...

```

import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

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

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

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

        return Ypred

```

Nearest Neighbor classifier

Q: 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

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

L1 (Manhattan) distance

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

L2 (Euclidean) distance

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

Linear Classification

airplane



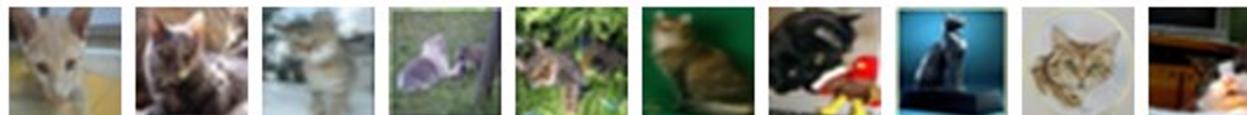
automobile



bird



cat



deer



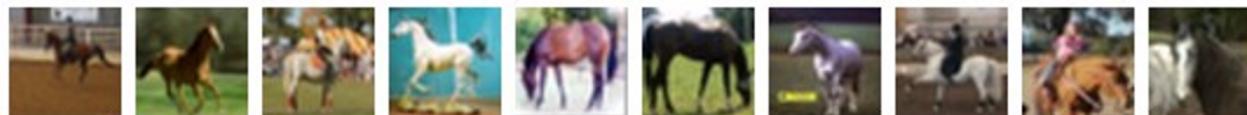
dog



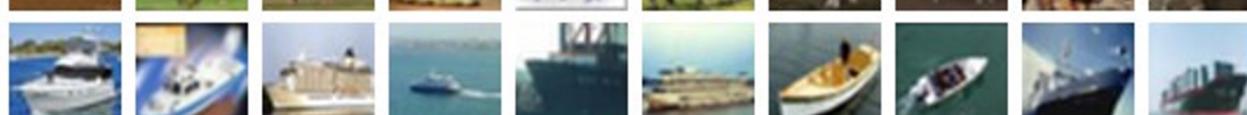
frog



horse



ship



truck



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

Parametric approach



image parameters

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

10 numbers,
indicating class
scores

[32x32x3]

array of numbers 0...1
(3072 numbers total)

Parametric approach: Linear classifier

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



10 numbers,
indicating class
scores

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

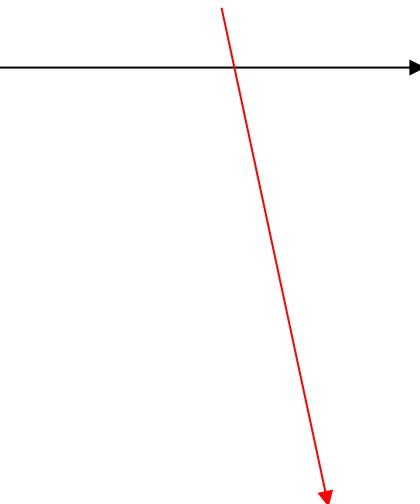
Parametric approach: Linear classifier



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

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

10x1 10x3072



10 numbers,
indicating class
scores

parameters, or “weights”

Parametric approach: Linear classifier



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

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

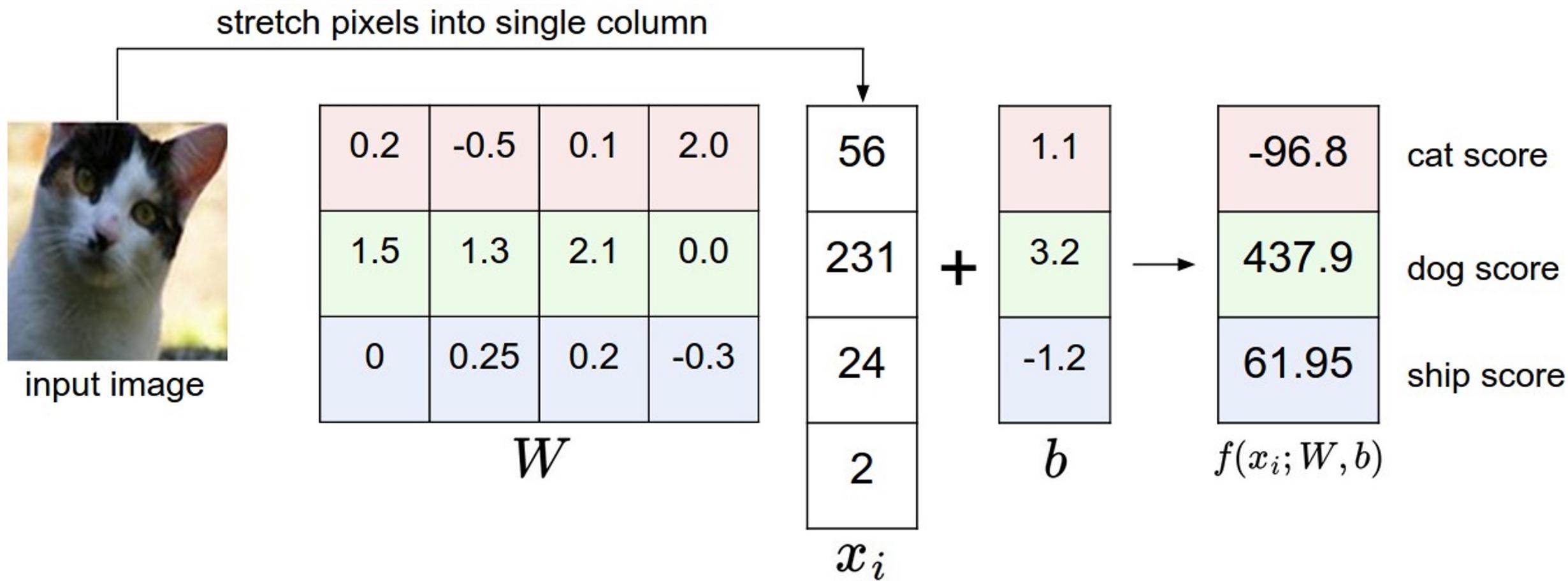
10x1 **10x3072** **3072x1**

(+b) **10x1**

10 numbers,
indicating class
scores

parameters, or “weights”

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



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

Coming up:

- Loss function
- Optimization
- Neural nets!

(quantifying what it means to have a “good” W)

(start with random W and find a W that minimizes the loss)

(tweak the functional form of f)

Suppose: 3 training examples, 3 classes.

With 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

Suppose: 3 training examples, 3 classes.

With 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: $s_i = f(x_i, W)$

the SVM loss has the form:

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

Suppose: 3 training examples, 3 classes.

With 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: $s_i = f(x_i, W)$

the SVM loss has the form:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \max(0, 5.1 - 3.2 + 1) \\ &\quad + \max(0, -1.7 - 3.2 + 1) \\ &= \max(0, 2.9) + \max(0, -3.9) \\ &= 2.9 + 0 \\ &= 2.9 \end{aligned}$$

Suppose: 3 training examples, 3 classes.

With 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		

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: $s_i = f(x_i, W)$

the SVM loss has the form:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \max(0, 1.3 - 4.9 + 1) \\ &\quad + \max(0, 2.0 - 4.9 + 1) \\ &= \max(0, -2.6) + \max(0, -1.9) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

Suppose: 3 training examples, 3 classes.

With 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	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 shorthand for the scores vector: $s_i = f(x_i, W)$

the SVM loss has the form:

$$\begin{aligned}L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\&= \max(0, 2.2 - (-3.1) + 1) \\&\quad + \max(0, 2.5 - (-3.1) + 1) \\&= \max(0, 6.3) + \max(0, 6.6) \\&= 6.3 + 6.6 \\&= 12.9\end{aligned}$$

Suppose: 3 training examples, 3 classes.

With 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	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 shorthand for the scores vector: $s_i = f(x_i, W)$

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 examples in the training data:

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

$$\begin{aligned} L &= (2.9 + 0 + 12.9)/3 \\ &= 5.3 \end{aligned}$$

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
```

There is a “bug” with the loss:

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

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



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

Suppose: 3 training examples, 3 classes.

With 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	
		12.9	

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

Before:

$$\begin{aligned} &= \max(0, 1.3 - 4.9 + 1) \\ &\quad + \max(0, 2.0 - 4.9 + 1) \\ &= \max(0, -2.6) + \max(0, -1.9) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

With W twice as large:

$$\begin{aligned} &= \max(0, 2.6 - 9.8 + 1) \\ &\quad + \max(0, 4.0 - 9.8 + 1) \\ &= \max(0, -6.2) + \max(0, -4.8) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

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



An example:

What is the loss? (POLL)

cat	1.3
car	2.5
frog	2.0

Loss:

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



An example:

What is the loss?

cat	1.3
car	2.5
frog	2.0
Loss:	0.5

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



An example:
What is the loss?

How could we change W to eliminate
the loss? (POLL)

cat	1.3
car	2.5
frog	2.0
Loss:	0.5

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



cat	1.3	2.6
car	2.5	5.0
frog	2.0	4.0
Loss:	0.5	0

An example:

What is the loss?

How could we change W to eliminate
the loss? (POLL)

Multiply W (and b) by 2!

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



cat	1.3	2.6
car	2.5	5.0
frog	2.0	4.0
Loss:	0.5	0

An example:

What is the loss?

How could we change W to eliminate the loss? (POLL)

Multiply W (and b) by 2!

Wait a minute! Have we done anything useful???

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

An example:

What is the loss?



cat	1.3	2.6
car	2.5	5.0
frog	2.0	4.0
Loss:	0.5	0

How could we change W to eliminate the loss? (POLL)

Multiply W (and b) by 2!

Wait a minute! Have we done anything useful???

No! Any example that used to be wrong is still wrong (on the wrong side of the boundary). Any example that is right is still right (on the correct side of the boundary).

Regularization

λ = regularization strength
(hyperparameter)

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

Data loss: Model predictions should match training data

Regularization: Prevent the model from having too much flexibility.

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

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

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

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

L2 Regularization

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

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

Softmax Classifier (Multinomial Logistic Regression)



cat	3.2
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)



scores = unnormalized log probabilities of the classes.

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

cat	3.2
car	5.1
frog	-1.7

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

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

cat	3.2
car	5.1
frog	-1.7

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 $s = f(x_i; W)$

cat	3.2	Softmax function
car	5.1	
frog	-1.7	

Softmax Classifier (Multinomial Logistic Regression)



cat	3.2
car	5.1
frog	-1.7

scores = unnormalized log probabilities of the classes.

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

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

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

Softmax Classifier (Multinomial Logistic Regression)



cat	3.2
car	5.1
frog	-1.7

scores = unnormalized log probabilities of the classes.

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

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

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

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

Softmax Classifier (Multinomial Logistic Regression)



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

cat
car
frog

3.2

5.1

-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
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
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
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

probabilities
>0, sum to 1

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
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

$$\rightarrow L_i = -\log(0.13) = 0.89$$

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

$$\rightarrow L_i = -\log(0.13) = 0.89$$

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

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

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

$$\rightarrow L_i = -\log(0.13) = 0.89$$

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

Q2: usually at initialization W are small numbers, so all s ~ 0. What is the loss?

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

$$\rightarrow L_i = -\log(0.13) = 0.89$$

unnormalized log probabilities

probabilities

hinge loss (SVM)

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

-2.85
0.86
0.28

$$\begin{aligned} & \max(0, -2.85 - 0.28 + 1) + \\ & \max(0, 0.86 - 0.28 + 1) \\ & = \\ & \mathbf{1.58} \end{aligned}$$

cross-entropy loss (Softmax)

-2.85
0.86
0.28

\exp

0.058
2.36
1.32

$\xrightarrow{\text{normalize}}$
(to sum to one)

0.016
0.631
0.353

$$\begin{aligned} & -\log(0.353) \\ & = \\ & \mathbf{0.452} \end{aligned}$$

Softmax vs. SVM

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

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