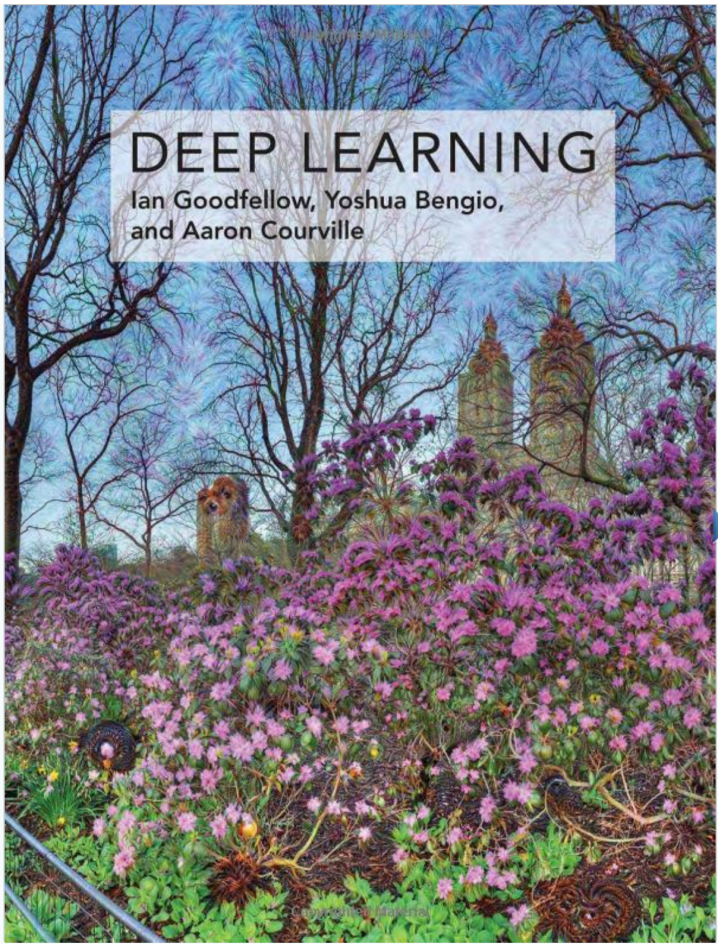
Announcements, 2018-01-22

- HW #1 is due tonight at 11:59pm.
- HW #2 will be uploaded tonight after lecture. It contains three coding notebooks.
- It is due in a week, Monday, 29 Jan 2018, at 11:59pm.

Announcements, 2018-01-22

Reading:

Deep Learning, 5.7, 5.9, 5.10, 5.11.1.



Example of ML estimation (cont.)

We'd like to maximize the likelihood of having seen the dataset. This can be written as

$$\mathcal{L} = p\left(\{\mathbf{x}^{(1)}, y^{(1)}\}, \{\mathbf{x}^{(2)}, y^{(2)}\}, \ldots, \{\mathbf{x}^{(N)}, y^{(N)}\}\right)$$

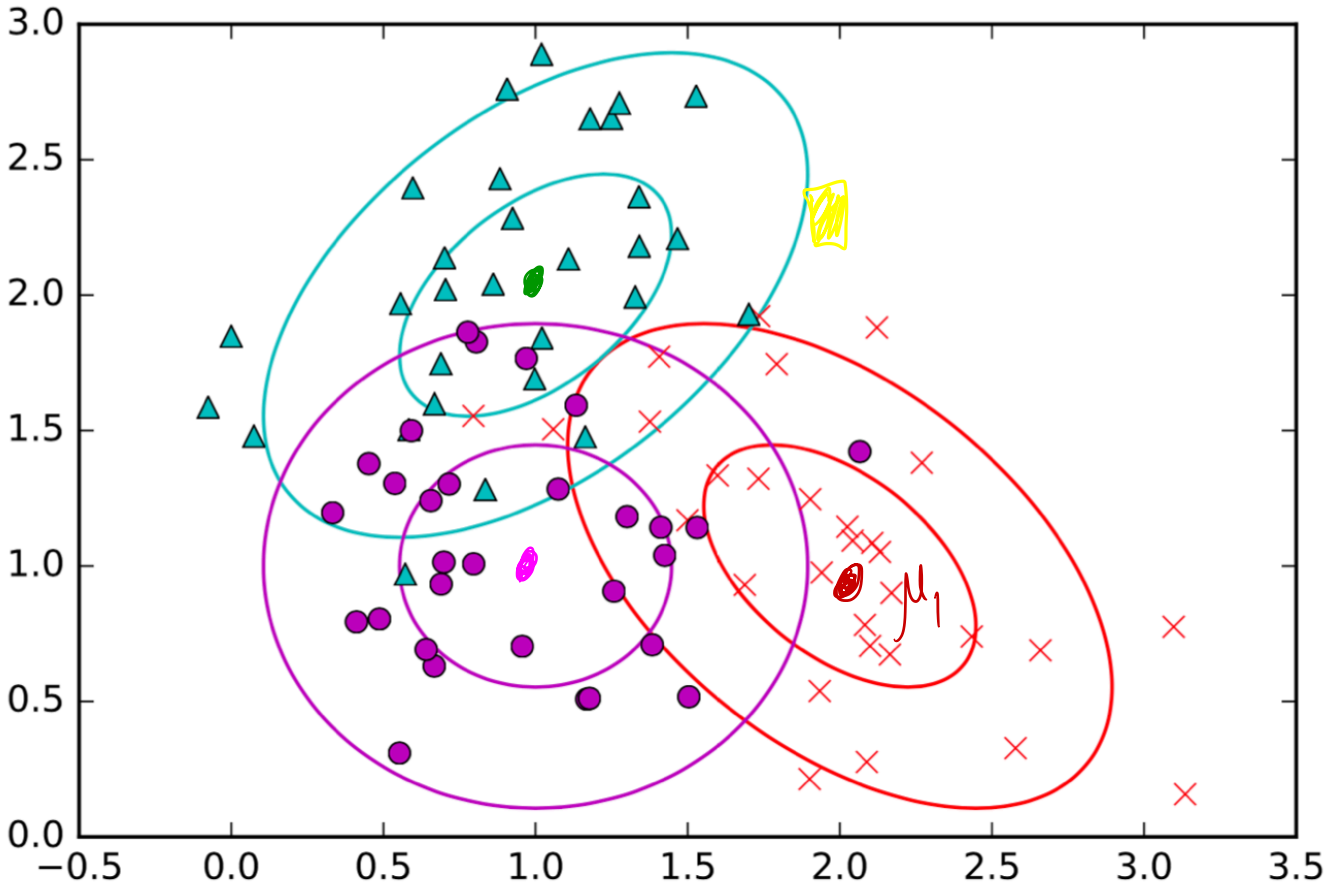
$x^{(i)} \in \mathbb{R}^2$

$y^{(i)}$

X

Δ

O



Maximum-likelihood estimation

$$\log \mathcal{L} = \sum_{i=1}^{N} \left[\log p(y^{(i)}) + \log p(\mathbf{x}^{(i)} | y^{(i)}) \right]$$

$$\frac{\partial \log \mathcal{L}}{\partial \mu_j} = 0 \Rightarrow \mu_j = \text{sample mean of all the points in class } j$$

$$\frac{\partial \log \mathcal{L}}{\partial \Sigma_j} = 0 \Rightarrow \Sigma_j$$

Maximum-likelihood estimation

$p(j) = \frac{1}{3}$

$p(j|x)\,p(x) = p(j,x) = p(j)\,p(x|j)$

Example of ML estimation (cont.)

Imagine now a new point $\mathbf{x}$ comes in that we'd like to classify as being in one of three classes. To do so, we'd like to calculate: $\Pr(y = j|\mathbf{x})$ and pick the j that maximizes this probability. We'll denote this probability $p(j|\mathbf{x})$. We'll also assume, as earlier, that the classes are equally probable, i.e., $\Pr(y = j)$ is the same for all j.

X's → j=1 , O's → j=2 Δ's → j=3

$$\Pr\{y = j \mid x\} = \frac{p(j)\,p(x|j)}{p(x)}$$

$\Pr\{y=1|x\} = 0.2$

$\Pr\{y=2|x\} = 0.7$

$\Pr\{y=3|x\} = 0.1$

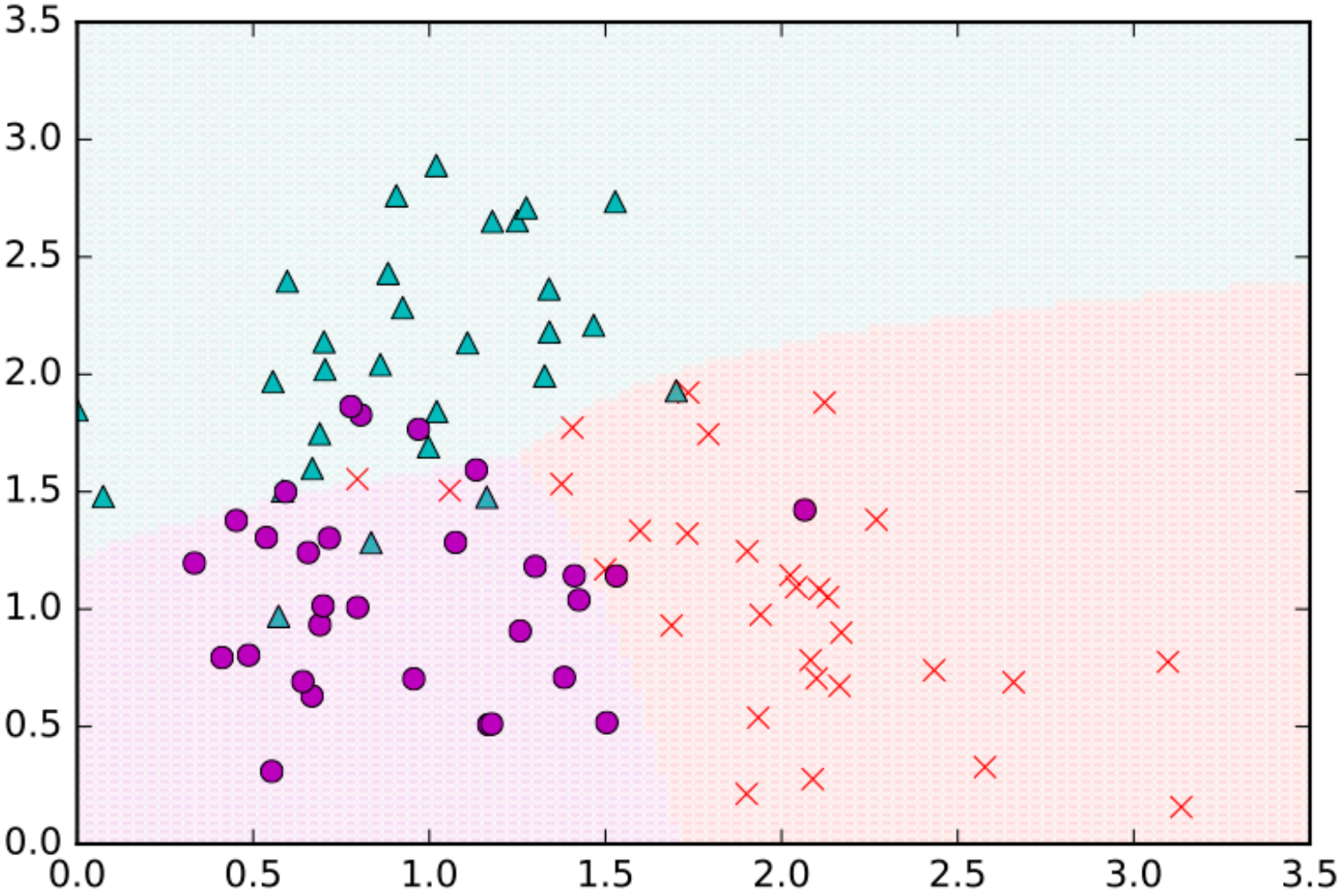
$$\underset{j}{\text{argmax}} \frac{p(j)\,p(x|j)}{p(x)} = \underset{j}{\text{argmax}}\; p(j)\,p(x|j) = \underset{j}{\text{argmax}}\; p(x|j)$$

Maximum-likelihood estimation

```
f = plt.figure()
ax = f.gca()
ax.plot(data1[0,:], data1[1,:], marker='x', linestyle='', color=class_colors[0])
ax.plot(data2[0,:], data2[1,:], marker='^', linestyle='', color=class_colors[1])
ax.plot(data3[0,:], data3[1,:], marker='o', linestyle='', color=class_colors[2])

for (i,xx) in enumerate(np.linspace(0, 3.5, 100)):
    for yy in np.linspace(0, 3.5, 100):
        xn = np.array((xx,yy))
        pmax = -np.Inf
        decoded_class = np.nan
        for c in np.arange(len(class_idxs)):
            class_prob = -np.log(np.linalg.det(covs[c])) - (xn-means[c]).T.dot(np.linalg.inv(covs[c])).dot(xn-means[c])
            if class_prob > pmax:
                pmax = class_prob
                decoded_class = c
        ax.plot(xx, yy, marker='.', color=class_colors[decoded_class], alpha=0.05)
ax.set_xlim(0,3.5)
f.savefig('ml-basics_maxlik-data_fit-ellipsoids_colorized.pdf')
```

Maximum-likelihood estimation



Where we go from here

Even more cost functions

There are even more cost functions that we could use. We'll encounter some others in this class. Others include:

- MAP estimation.
- KL divergence.
- Maximize an approximation of the distribution.

In machine learning, it is important to arrive at an appropriate model and cost function. After that, it's important to know how to optimize it. In our examples here, the models were simple enough that we could differentiate and set the derivative equal to zero. In future lectures, we'll discuss more general ways to learn parameters of models when they are not so simple.

Lecture 3: Supervised classification & gradient descent principles

In this lecture, we'll cover some supervised learning techniques that are relevant for classification, and helpful for later lectures in neural networks.

inputs: $x^{(i)}$ image

outputs: labels cat, dog, airplane

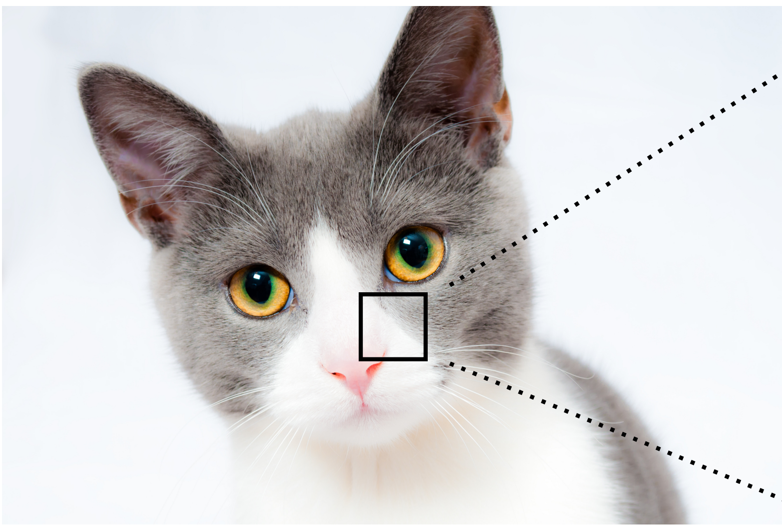
label $\leftarrow f(\text{input})$

Image classification

This image is licensed under CC0. https://www.pexels.com/photo/grey-and-white-short-fur-cat-104827/

Image classification

600



800

This image is licensed under CC0.

```
[[217, 215, 213, 213, 211, 210, 209, 207, 205, 203, 202, 201, 209],
 [218, 216, 215, 214, 211, 210, 209, 207, 205, 203, 202, 201, 209],
 [220, 218, 217, 216, 211, 210, 209, 207, 205, 203, 202, 201, 209],
 [220, 219, 217, 216, 211, 210, 209, 207, 205, 203, 202, 201, 209],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214]]
```

800 x 600 x 3   R, G, B

Images are stored in computers as a width x height x 3 array with numbers from [0, 255].

This is the input data for a computer vision algorithm.

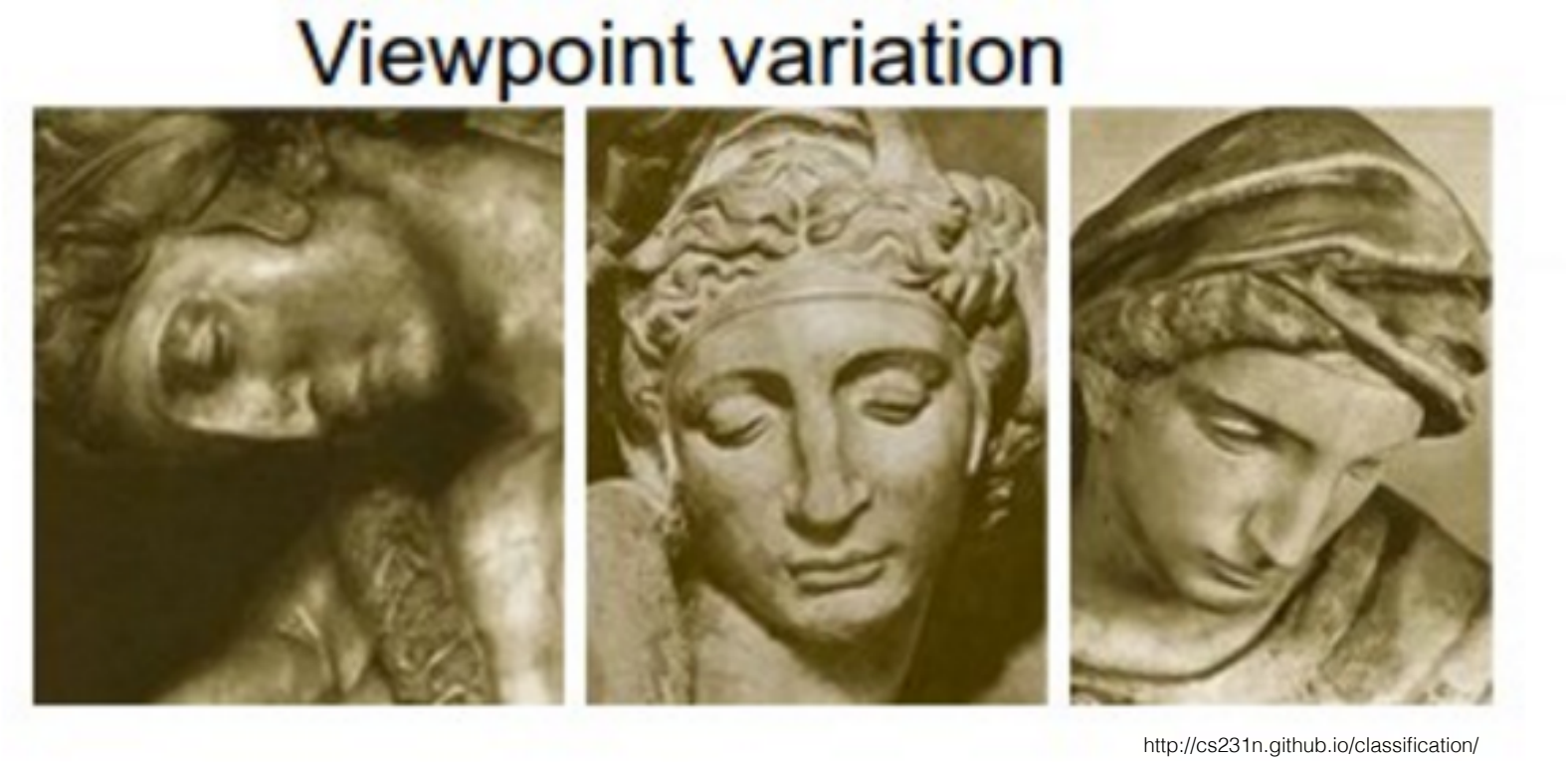
255	0	0
0	0	255
255	0	255

Image classification

As the image is stored as an array of numbers, there are several challenges associated with image classification.

Image classification

Viewpoint variation:



http://cs231n.github.io/classification/

Image classification

Viewpoint variation

https://kevinzakka.github.io/2017/01/18/stn-part2/

Image classification

Illumination.

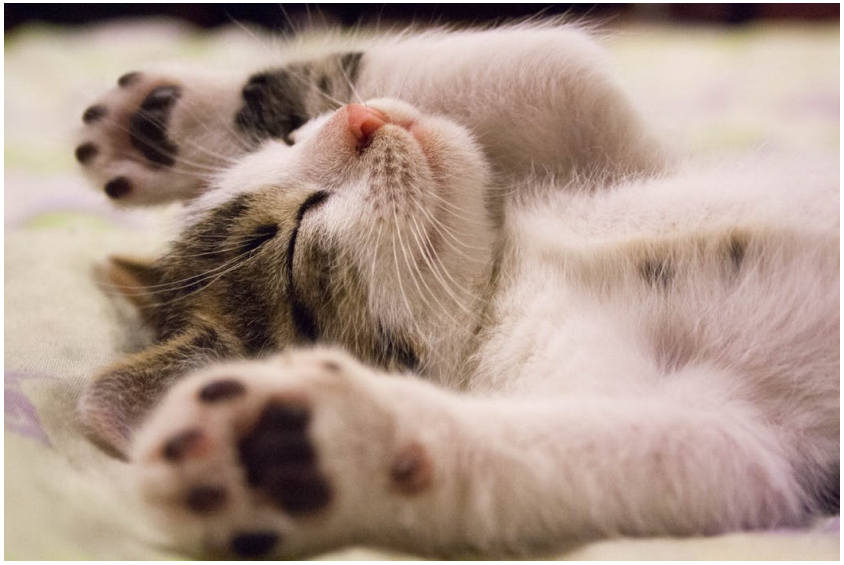
This image is licensed under CC0.

This image is licensed under CC0.

Image classification

Deformation.

This image is licensed under CC0.



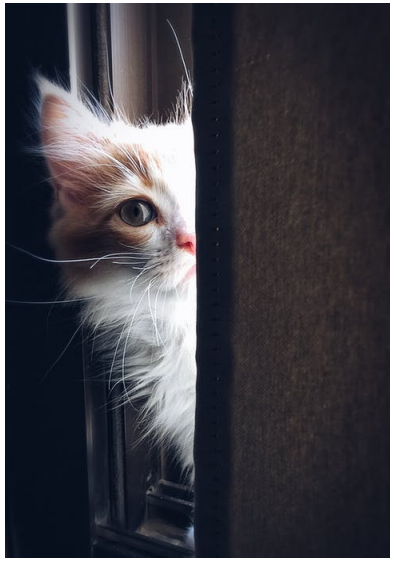
This image is licensed under CC0.

This image is licensed under CC0.

Image classification

Occlusion.

This image is licensed under CC0.



This image is licensed under CC0.

This image is licensed under CC0.

Image classification

Background clutter.

This image is licensed under CC0.

This image is licensed under CC0.

Image classification

Intraclass variation.

Classifying image data?

When trying to classify an image, there are several approaches one could think of taking.

In this class, we use the data driven approach, where we let a machine learning algorithm see a lot of data and learn a function mapping the image to class.

Train : data $\longrightarrow$ model's params.

Test : model $\longrightarrow$ prediction.

Classifying image data

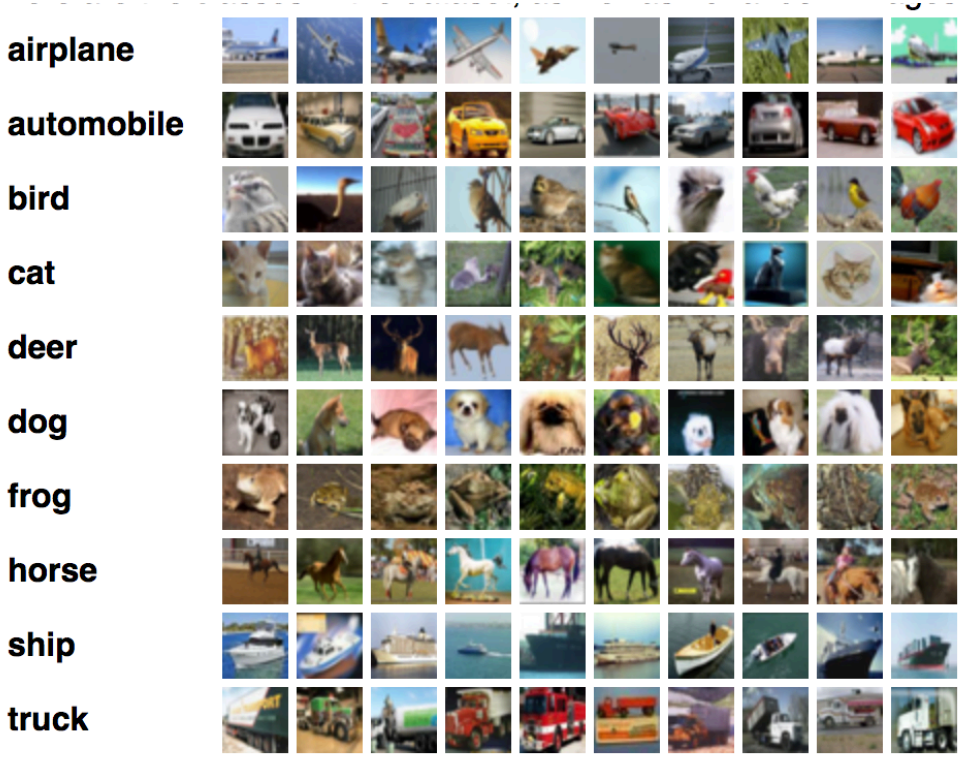
For data, we will use the CIFAR-10 dataset: http://www.cs.toronto.edu/~kriz/cifar.html

The dataset comprises:

- 60,000 images that are 32 x 32 pixels in color (hence 3 channels). $32 \times 32 \times 3$
- 10 classes, with 6,000 images per class.
- 50,000 images are for training and 10,000 are for testing.

$x^{(i)}$

$y^{(i)}$



$32 \cdot 32 \cdot 3$

$= 3072$

Classifying image data

```
# Loading CIFAR-10
import os
import pickle

def load_CIFAR10(ROOT):
  """ load all of cifar """
  xs = []
  ys = []
  for b in range(1,6):
    f = os.path.join(ROOT, 'data_batch_%d' % (b, ))
    X, Y = load_CIFAR_batch(f)
    xs.append(X)
    ys.append(Y)
  Xtr = np.concatenate(xs)
  Ytr = np.concatenate(ys)
  del X, Y
  Xte, Yte = load_CIFAR_batch(os.path.join(ROOT, 'test_batch'))
  return Xtr, Ytr, Xte, Yte

def load_CIFAR_batch(filename):
  """ load single batch of cifar """
  with open(filename, 'rb') as f:
    datadict = pickle.load(f)
    X = datadict['data']
    Y = datadict['labels']
    X = X.reshape(10000, 3, 32, 32).transpose(0,2,3,1).astype("float")
    Y = np.array(Y)
    return X, Y
```

Classifying image data

```
X_train, y_train, X_test, y_test = load_CIFAR10('cifar-10-batches-py')

print 'Training data shape: ', X_train.shape
print 'Training labels shape: ', y_train.shape
print 'Test data shape: ', X_test.shape
print 'Test labels shape: ', y_test.shape
```

```
Training data shape:  (50000, 32, 32, 3)
Training labels shape:  (50000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```

We will reshape these 32x32x3 arrays into a 3072-dimensional vector.

These constitute the “input” data.

Classifying image data?

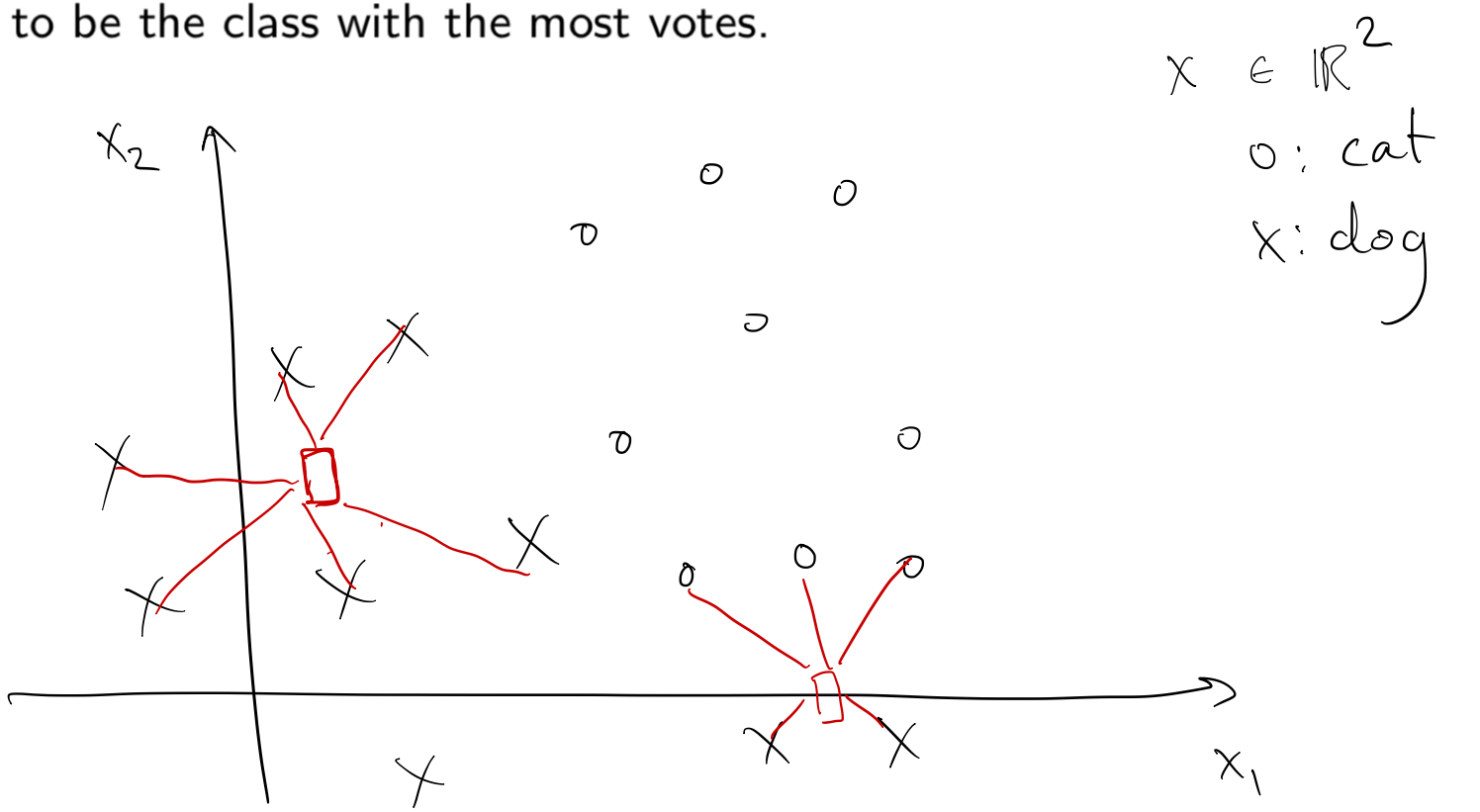
Supervised classification (cont.)

Consider a setup that is the same as the maximum-likelihood classifier of the previous lecture. Imagine you were given a training set of input vectors, $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}\}$ and their corresponding classes, $\{y^{(1)}, y^{(2)}, \ldots, y^{(m)}\}$.

Now imagine that you were also given a new data point, $\mathbf{x}^{\text{new}}$. We found a way to classify through a probabilistic model, where we had to learn parameters, but isn't there a simpler way to classify that doesn't involve very much "machine learning" machinery?

k-nearest neighbors

Intuitively, k-nearest neighbors says to find the k closest points (or nearest neighbors) in the training set, according to an appropriate metric. Each of its k nearest neighbors then vote according to what class it is in, and $\mathbf{x}^{\text{new}}$ is assigned to be the class with the most votes.

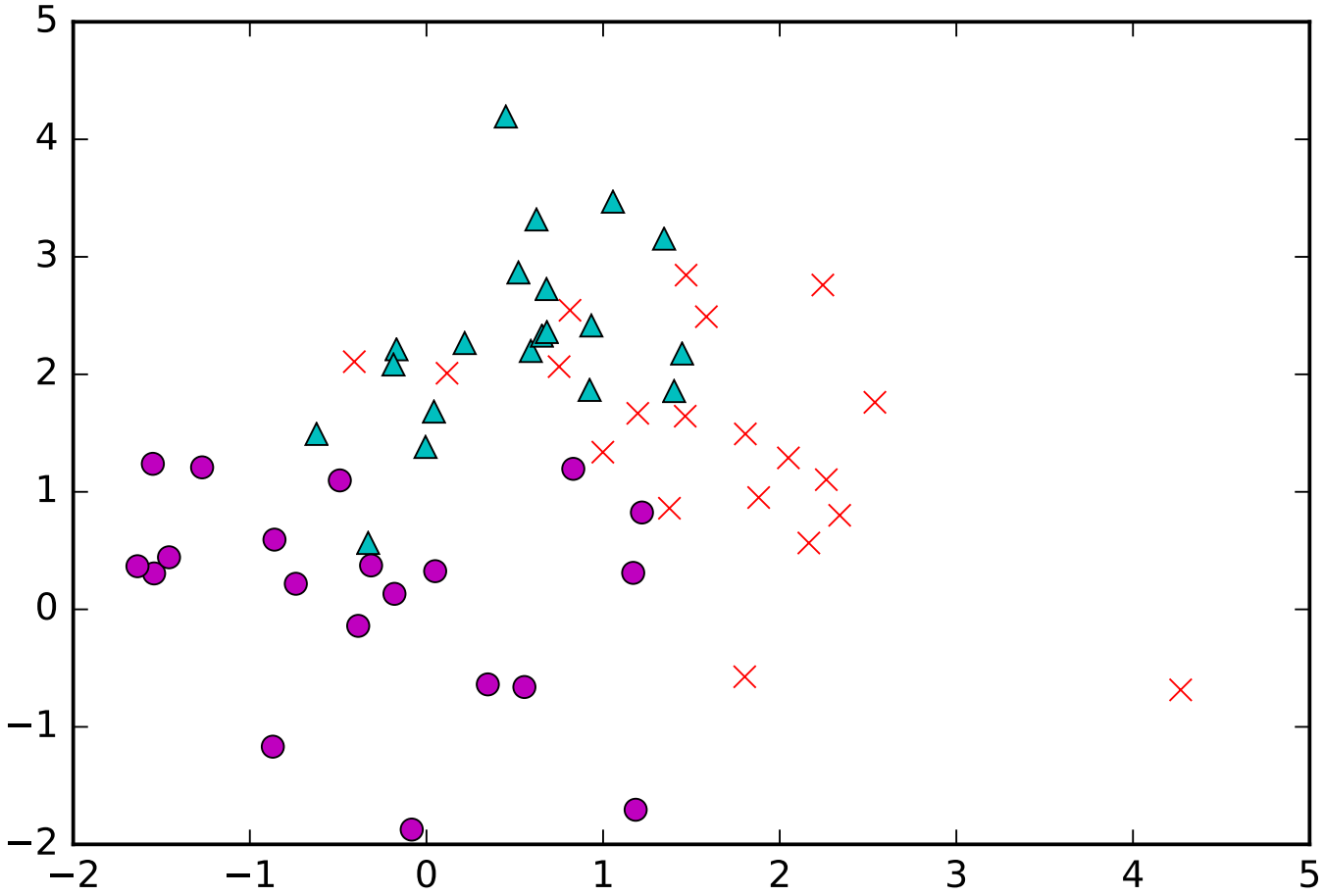


k-nearest neighbors, more formally

- Choose an appropriate distance metric, $d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$, returning the distance between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$. E.g., $d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \left\| \mathbf{x}^{(i)} - \mathbf{x}^{(j)} \right\|_2 = \sqrt{\sum_{k=1}^{n} \left(x_k^{(i)} - x_k^{(j)} \right)^2}$
- Choose the number of nearest neighbors, k.
- Take desired test data point, $\mathbf{x}^{\text{new}}$, and calculate $d(\mathbf{x}^{\text{new}}, \mathbf{x}^{(i)})$ for $i = 1, \ldots, m$.
- With $\{c_1, \ldots, c_k\}$ denoting the k indices corresponding to the k smallest $d(\mathbf{x}^{\text{new}}, \mathbf{x}^{(i)})$, classify $\mathbf{x}^{\text{new}}$ as the class that occurs most frequently amongst $\{y^{c_1}, \ldots, y^{c_k}\}$. If there is a tie, any of the tying classes may be selected.

k-nearest neighbors

Example data:



k-nearest neighbors

How do we train the classifier?

k-nearest neighbors

How do we train the classifier?

```
class KNearestNeighbor(object):

  def __init__(self):
    pass

  def train(self, X, y):
    self.X_train = X
    self.y_train = y
```

Pros? Simple

Cons? Memory.

$x^{(i)}$: 2 #'s

$y^{(i)}$: 1 #

} 3 #'s

18 #'s

2 #'s

4 #'s

μ_1, Σ_1

μ_2, Σ_2

μ_3, Σ_3

k-nearest neighbors

How do we test a new data point?

```
class KNearestNeighbor(object):

  def __init__(self):
    pass

  def train(self, X, y):
    self.X_train = X
    self.y_train = y

  def test(self, x_test, k=1):

    dists = np.linalg.norm(self.X_train.T - x_test, axis=1).T
    sortedIdxs = np.argsort(dists)
    closest_y = self.y_train[sortedIdxs[:k]]
    y_pred = np.argmax(np.bincount(closest_y));

    return y_pred
```

X_train : 2 x N array

X_test : 2 array

X_train.T : N x 2

bsxfun

Pros? Simple

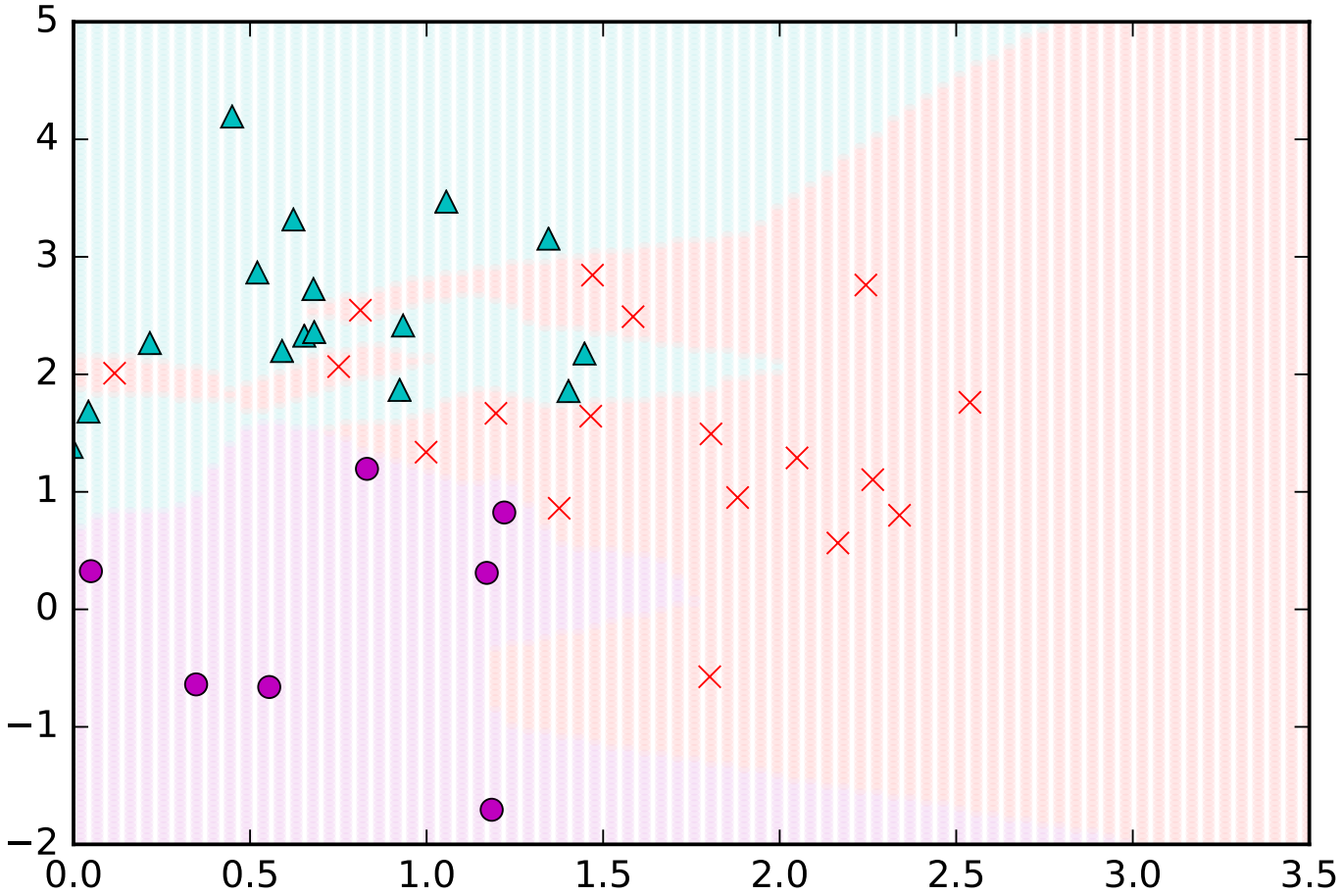
Cons? O(N)

Train: O(1)

Test: O(N)

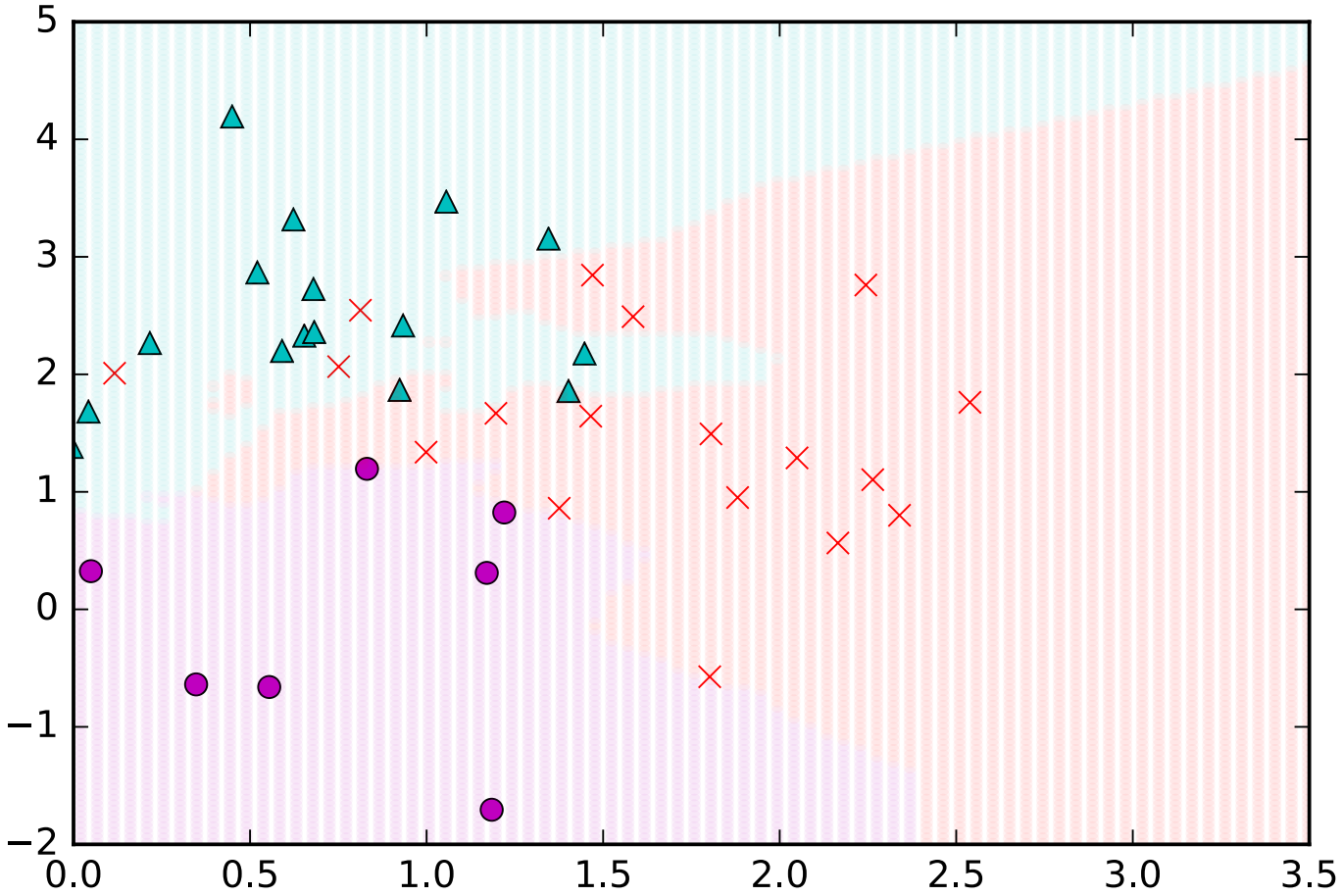
k-nearest neighbors

What a solution looks like for k=1 neighbor:



k-nearest neighbors

What a solution looks like for k=3 neighbors:

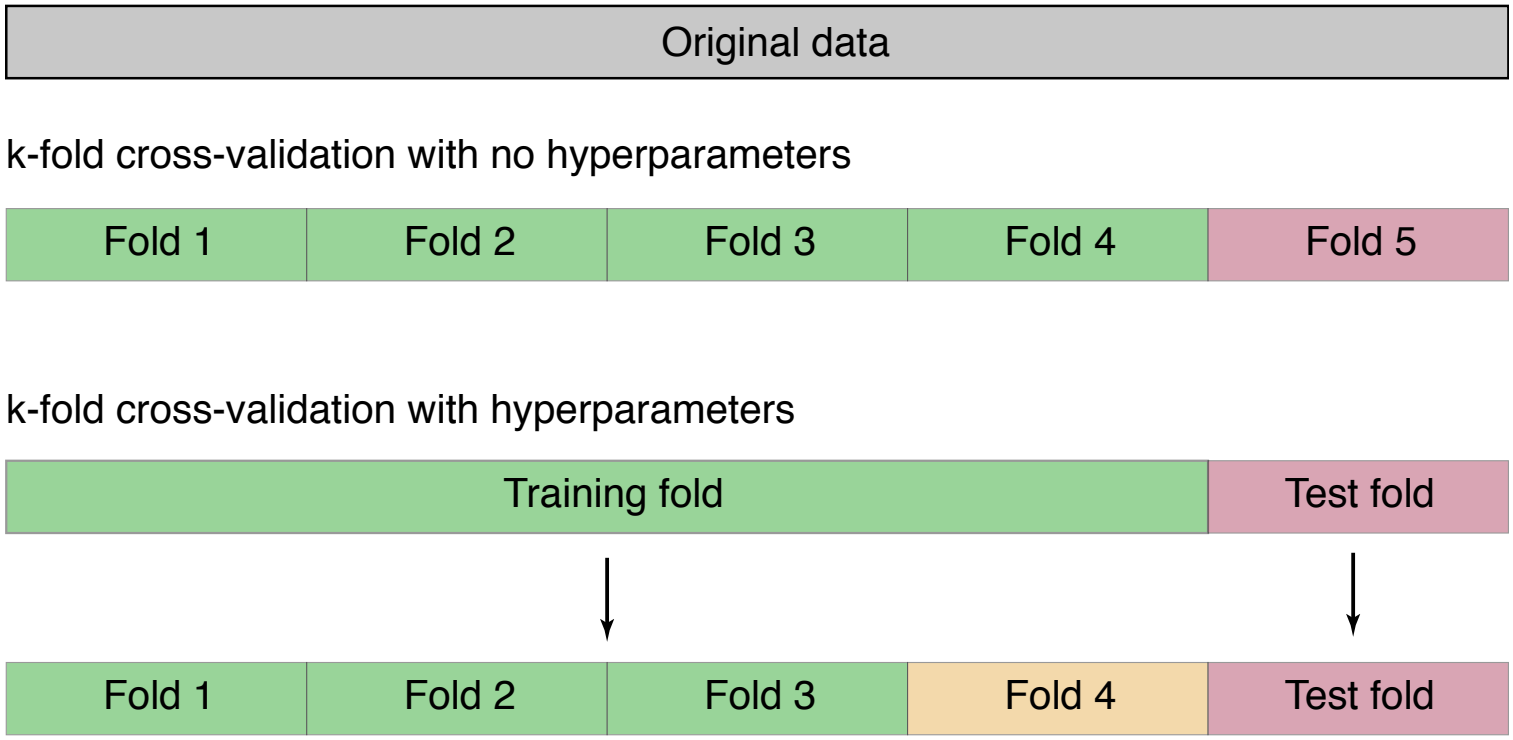


k-nearest neighbors

What are the hyperparameters of k-nearest neighbors?

k-nearest neighbors

How do we choose what the best *k* and distance metric is to use?

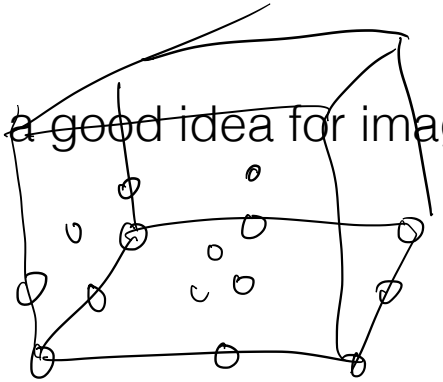


k-nearest neighbors

Why might k-nearest neighbors not be a good idea for image classification?

k-nearest neighbors

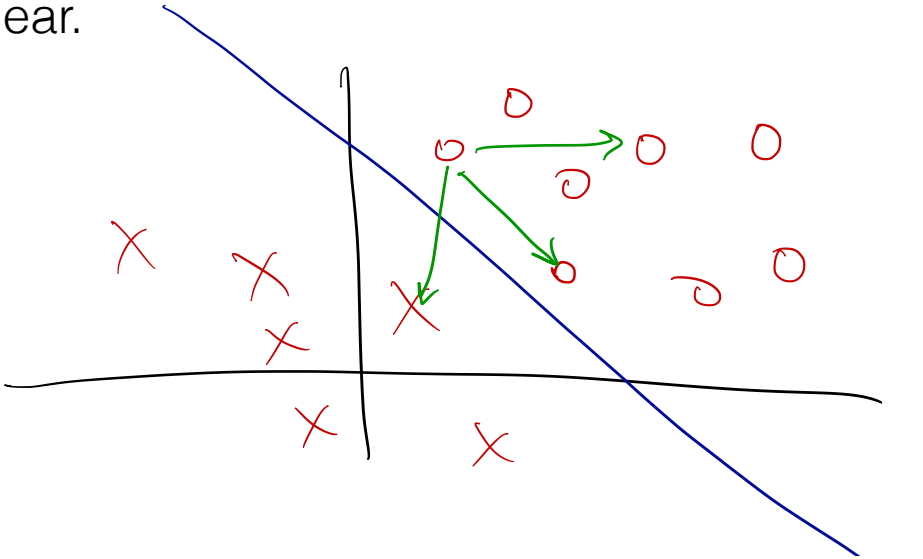
Why might k-nearest neighbors not be a good idea for image classification?



Curse of dimensionality:

- Images are very high-dimensional vectors, e.g., each CIFAR-10 image is a 3072 dimensional vector (and these are small images).

- Notions of "distance" become less intuitive in higher dimensions.
  - In higher-dimensional space, the volume increases exponentially.
  - This leaves a lot of empty space — and so the nearest neighbors may not be so near.



k-nearest neighbors

Why might k-nearest neighbors not be a good idea for image classification?

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

Credit: CS231n, Stanford University

Classifiers based on linear classification

Perhaps a better way would be to develop a “score” for an image coming from each class, and then pick the class that achieves the highest score.

Linear classifiers may seem simple, but they compose a major building block for neural networks.

In particular, each layer of a neural network is composed of a linear classifier that is then passed through a nonlinearity.

Classifiers based on linear classification

Example 2: Consider a matrix, $\mathbf{W}$, defined as:

$$\begin{bmatrix} - \mathbf{w}_1^T - \\ \vdots \\ - \mathbf{w}_c^T - \end{bmatrix}$$

Then, $\mathbf{W} \in \mathbb{R}^{c \times N}$. Let $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{b}$ is a vector of bias terms. Then $\mathbf{y} \in \mathbb{R}^c$ is a vector of scores, with its ith element corresponding to the score of $\mathbf{x}$ being in class i. The chosen class corresponds to the index of the highest score in $\mathbf{y}$.

$$\begin{bmatrix} w_1^T x + b_1 \\ w_2^T x + b_2 \\ \vdots \\ w_c^T x + b_c \end{bmatrix} = \begin{bmatrix} - w_1^T - \\ - w_2^T - \\ \vdots \\ - w_c^T - \end{bmatrix} \begin{bmatrix} x \end{bmatrix} + \begin{bmatrix} b \end{bmatrix}$$

CIFAR-10

$x \in \mathbb{R}^{3072}$

$y \in \mathbb{R}^{10}$

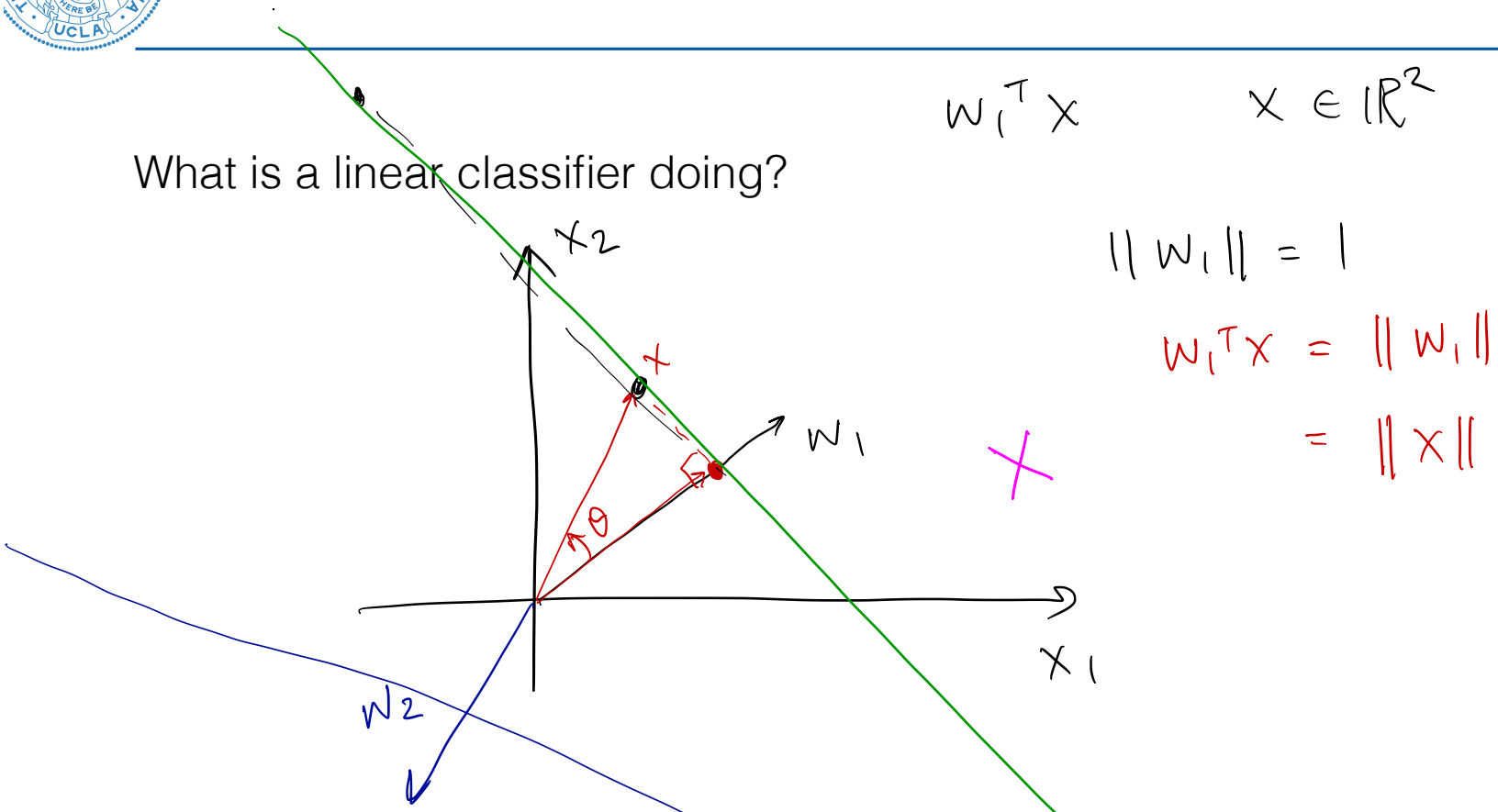
$W \in \mathbb{R}^{10 \times 3072}$

Classifiers based on linear classification

Each row of **W** can be thought of as a template.

Classifiers based on linear classification

What is a linear classifier doing?



$w_1^T x \qquad x \in \mathbb{R}^2 \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$\|w_1\| = 1$

$w_1^T x = \|w_1\| \|x\| \cos\theta$

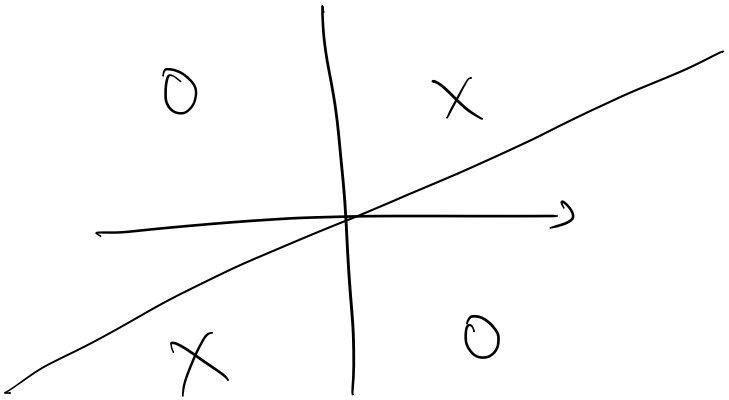
$= \|x\| \cos\theta$

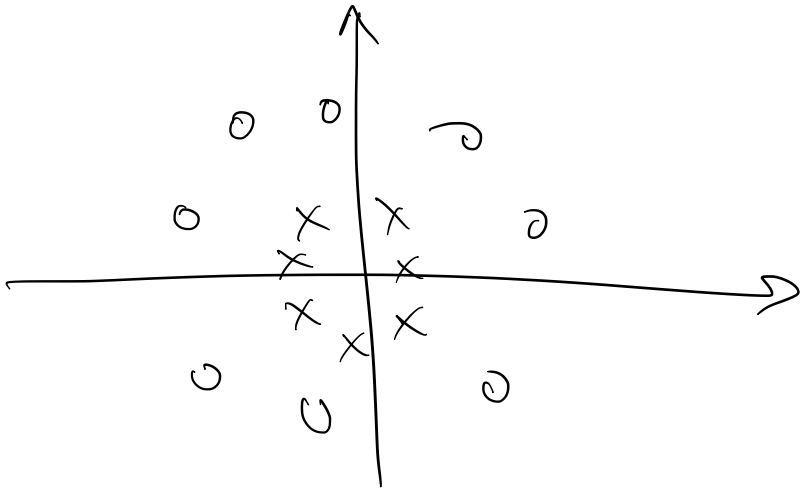
$w_1^T x = k$

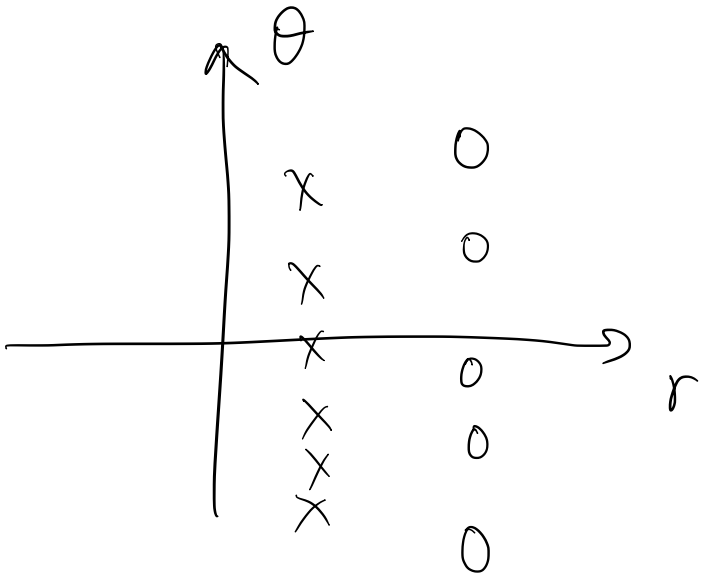
$w_2^T x = k$

Classifiers based on linear classification

Where might linear classifiers fail?







Classifiers based on linear classification

Next, how do we take the scores we receive (which are analog in value) and turn them into an appropriate **loss function** for us to optimize, so we can learn **W** and **b** appropriately?

$$\hat{y} = Wx + b$$

Turn the scores into a probability

A first thought is to turn the scores into probabilities.

Softmax function

There are several instances when the scores should be normalized. This occurs, for example, in instances where the scores should be interpreted as probabilities. In this scenario, it is appropriate to apply the *softmax* function to the scores.

The softmax function transforms the class score, $\mathrm{softmax}_i(\mathbf{x})$, so that:

$$\mathrm{softmax}_i(\mathbf{x}) = \frac{e^{a_i(\mathbf{x})}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x})}}$$

for $a_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + w_i$ and c being the number of classes.

$a_i(x) = w_i^T x + b_i$

$0 \le \mathrm{softmax}_i(x) \le 1$

2 $a_1(x) = w_1^T x \qquad a_2(x) = w_2^T x$

$$\mathrm{Softmax}_1(x) = \frac{e^{w_1^T x}}{e^{w_1^T x} + e^{w_2^T x}}$$

$$\mathrm{Softmax}_2(x) = \frac{e^{w_2^T x}}{\quad}$$

Softmax classifier

softmax$_1(x)$ ... softmax$_{10}(x)$

Pr(label is class 1 given x)

Pr(label is class 10 given x)

$$\text{softmax}_i(\mathbf{x}) = \frac{e^{a_i(\mathbf{x})}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x})}}$$

for $a_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + w_i$ and c being the number of classes.

If we let $\theta = \{\mathbf{w}_j, w_j\}_{j=1,\ldots,c}$, then $\text{softmax}_i(\mathbf{x})$ can be interpreted as the probability that $\mathbf{x}$ belongs to class i. That is,

$$\Pr(y^{(j)} = i | \mathbf{x}^{(j)}, \theta) = \text{softmax}_i(\mathbf{x}^{(j)})$$

Softmax classifier

Although we know the softmax function, how do we specify the *objective* to be optimized with respect to θ?

$= \{W, b\}$

One intuitive heuristic is that we should choose the parameters, θ, so as to maximize the likelihood of having seen the data. Assuming the samples, $(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})$ are iid, this corresponds to maximizing:

$$\begin{aligned} p(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}, y^{(1)}, \ldots, y^{(m)} | \theta) &= \prod_{i=1}^{m} p(\mathbf{x}^{(i)}, y^{(i)} | \theta) \\ &= \prod_{i=1}^{m} p(\mathbf{x}^{(i)} | \theta) p(y^{(i)} | \mathbf{x}^{(i)}, \theta) \end{aligned}$$

Softmax classifier

$$\arg\max_\theta \prod_{i=1}^{m} p(\mathbf{x}^{(i)}|\theta)p(y^{(i)}|\mathbf{x}^{(i)},\theta) \quad = \quad \arg\max_\theta \prod_{i=1}^{m} p(y^{(i)}|\mathbf{x}^{(i)},\theta)$$

$\text{softmax}_i(x)$

$$= \arg\max_\theta \frac{1}{m}\sum_{i=1}^{m} \log \text{softmax}_{y^{(i)}}(x^{(i)})$$

$$= \arg\max_\theta \frac{1}{m}\sum_{i=1}^{m} \left[\log \frac{e^{a_{y^{(i)}}(x^{(i)})}}{\sum_j e^{a_j(x^{(i)})}} \right]$$

$$= \arg\max_\theta \frac{1}{m}\sum_{i=1}^{m} \left[a_{y^{(i)}}(x^{(i)}) - \log \sum_{j=1}^{c} e^{a_j(x^{(i)})} \right]$$

$$= \arg\min_\theta \frac{1}{m}\sum_{i=1}^{m} \left[\log \sum_{j=1}^{c} e^{a_j(x^{(i)})} - a_{y^{(i)}}(x^{(i)}) \right]$$

Softmax classifier

Now we have our softmax loss function.

$$\arg\min_{\theta} \sum_{i=1}^{m} \left(\log \sum_{j=1}^{c} e^{a_j(\mathbf{x})} - a_{y^{(i)}}(\mathbf{x}^{(i)}) \right)$$

Note, we haven't figured out yet how to get the optimal parameters.

(That'll be later.)

Softmax classifier

Simple sanity check:



Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2

Score check: $-a_{y^{(i)}}(\mathbf{x}^{(i)}) + \log \sum_{j=1}^{c} \exp(a_j(\mathbf{x}^{(i)}))$

Softmax classifier

Simple sanity check:



Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2

Cat: -2.1 + log(exp(2.1) + exp(3.4) + exp(-2.0)) = 1.54

Softmax classifier

Simple sanity check:



Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	1.54		

Car: -5.1 + log(exp(0.2) + exp(5.1) + exp(1.7)) = 0.04

Softmax classifier

Simple sanity check:



Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	1.54	0.04	

Bird: 1.2 + log(exp(2.3) + exp(3.1) + exp(-1.2)) = 4.68

Softmax classifier

Simple sanity check:



Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	1.54	0.04	4.68

$$-a_{y^{(i)}}(\mathbf{x}^{(i)}) + \log \sum_{j=1}^{c} \exp(a_j(\mathbf{x}^{(i)}))$$

Softmax classifier

A few additional notes on the softmax classifier:

Softmax classifier: intuition

When optimizing likelihoods, we typically work with the "log likelihood." When applying the softmax, we interpret its output as the probability of a class.

$$\begin{aligned} \log \Pr(y = i|\mathbf{x}) &= \log \text{softmax}_i(\mathbf{x}) \\ &= a_i(\mathbf{x}) - \log \sum_{j=1}^{c} \exp(a_j(\mathbf{x})) \end{aligned}$$

When maximizing this, the term $a_i(\mathbf{x})$ is made larger, and the term $\log \sum_j \exp(a_j(\mathbf{x}))$ is made smaller. The latter term can be approximated by $\max_j a_j(\mathbf{x})$. (Why?)

We consider two scenarios:

- If $a_i(\mathbf{x})$ produces the largest score, then the log likelihood is approximately 0.
- If $a_j(\mathbf{x})$ produces the largest score for $j \neq i$, then $a_i(\mathbf{x}) - a_j(\mathbf{x})$ is negative, and thus the log likelihood is negative.

Softmax classifier

A few additional notes on the softmax classifier:

Overflow of softmax

A potential problem when implementing a softmax classifier is overflow.

- If $a_i(\mathbf{x}) \gg 0$, then $e^{a_i(\mathbf{x})}$ may be very large, and numerically overflow and / or result to numerical impression.
- Thus, it is standard practice to normalize the softmax function as follows:

$$\begin{aligned}
\text{softmax}_i(\mathbf{x}) &= \frac{e^{a_i(\mathbf{x})}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x})}} \\
&= \frac{k e^{a_i(\mathbf{x})}}{k \sum_{j=1}^{c} e^{a_j(\mathbf{x})}} \\
&= \frac{e^{a_i(\mathbf{x}) + \log k}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x}) + \log k}}
\end{aligned}$$

$\tilde{a}_i(x) = a_i(x) + \log k$

$e^{a_i(x)} e^{\log k} = e^{a_i(x) + \log k}$

- A sensible choice of k is so that $\log k = -\max_i a_i(\mathbf{x})$, making the maximal argument of the exponent 0.

Support vector machine

Before getting to parameter fitting, we'll want to introduce one more classifier that is commonly used: the support vector machine.

Support vector machine: introduction

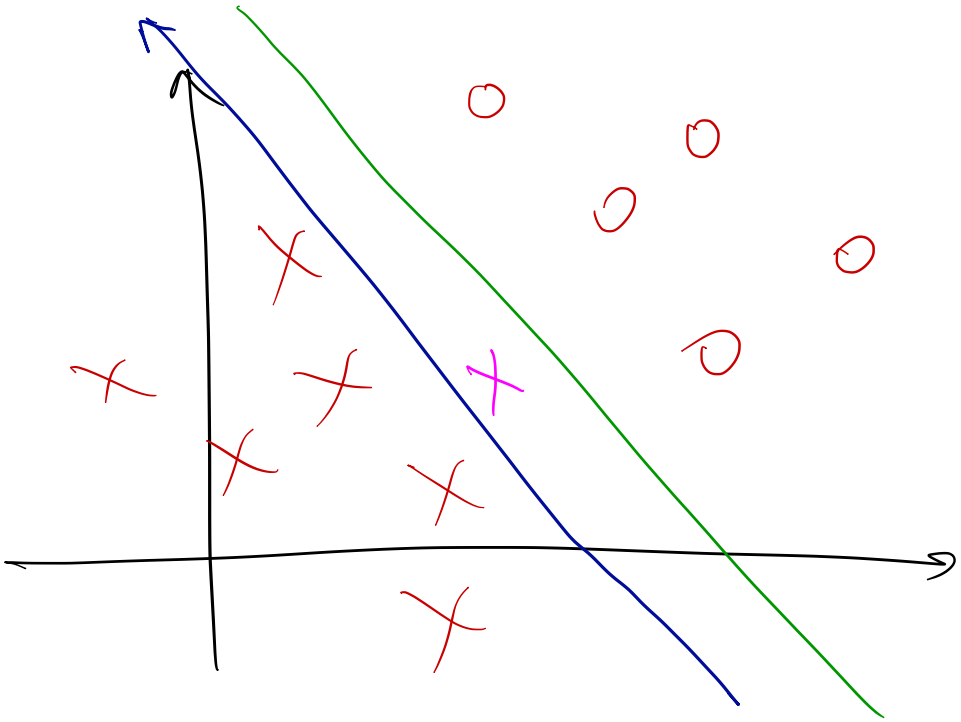
The SVM is a commonly used and has much theory behind it. A typical machine learning class will formulate the SVM as a convex optimization problem. However, this is beyond the scope of this class.

Instead, we'll talk about the SVM at a very high-level using a soft-margin "hinge loss" and provide appropriate intuitions. We'll focus on linear SVMs and will *not* touch on kernels. Please look into a machine learning class for more information about the SVM.

Support vector machine: introduction

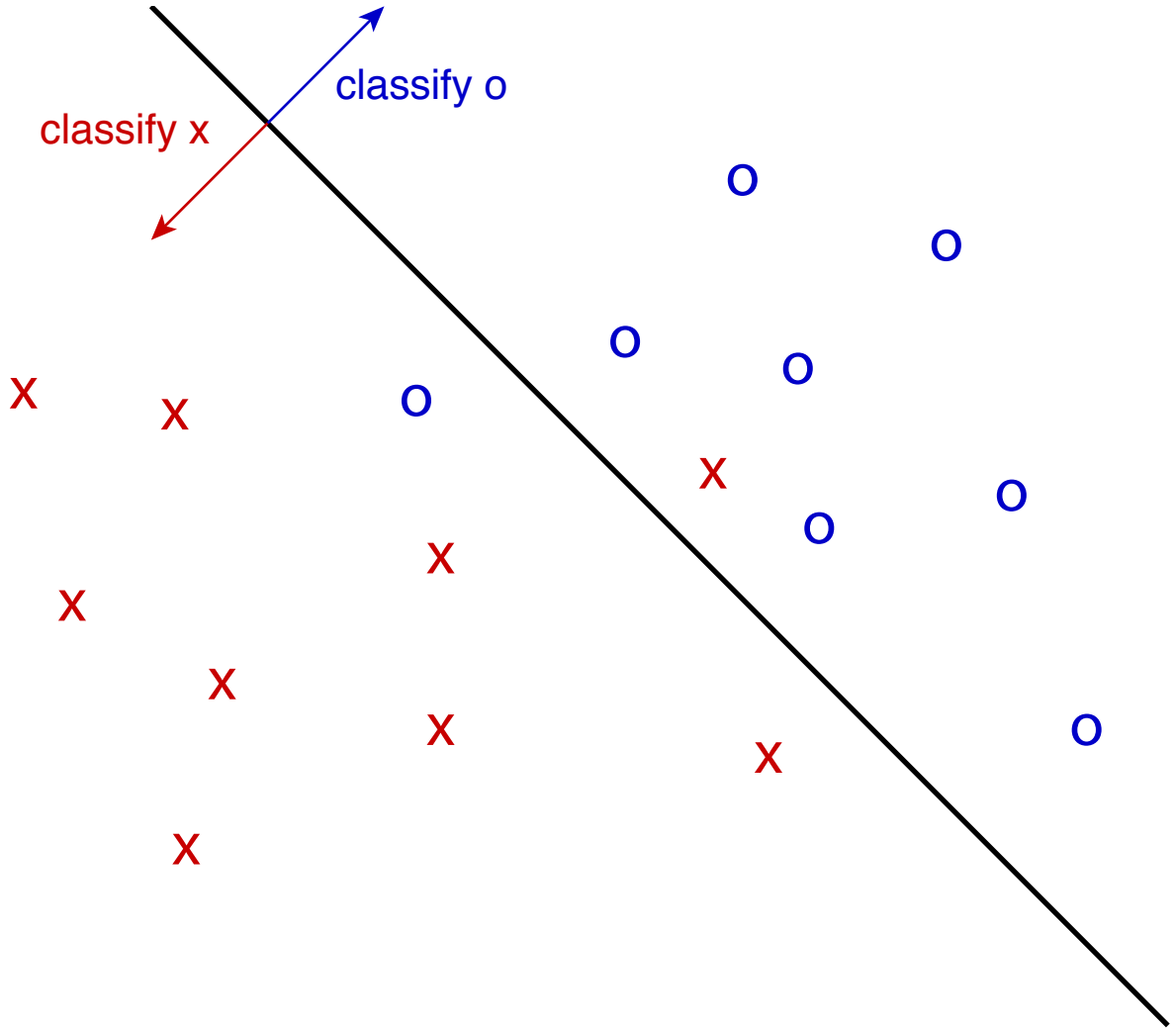
Another common decision boundary classifier is the support vector machine (SVM).

Informally, the SVM finds a boundary that maximizes the *margin*, or intuitively the "gap" between the boundary and the data points. The fundamental idea here is that if a point is further away from the decision boundary, there ought to be greater *confidence* in classifying that point.



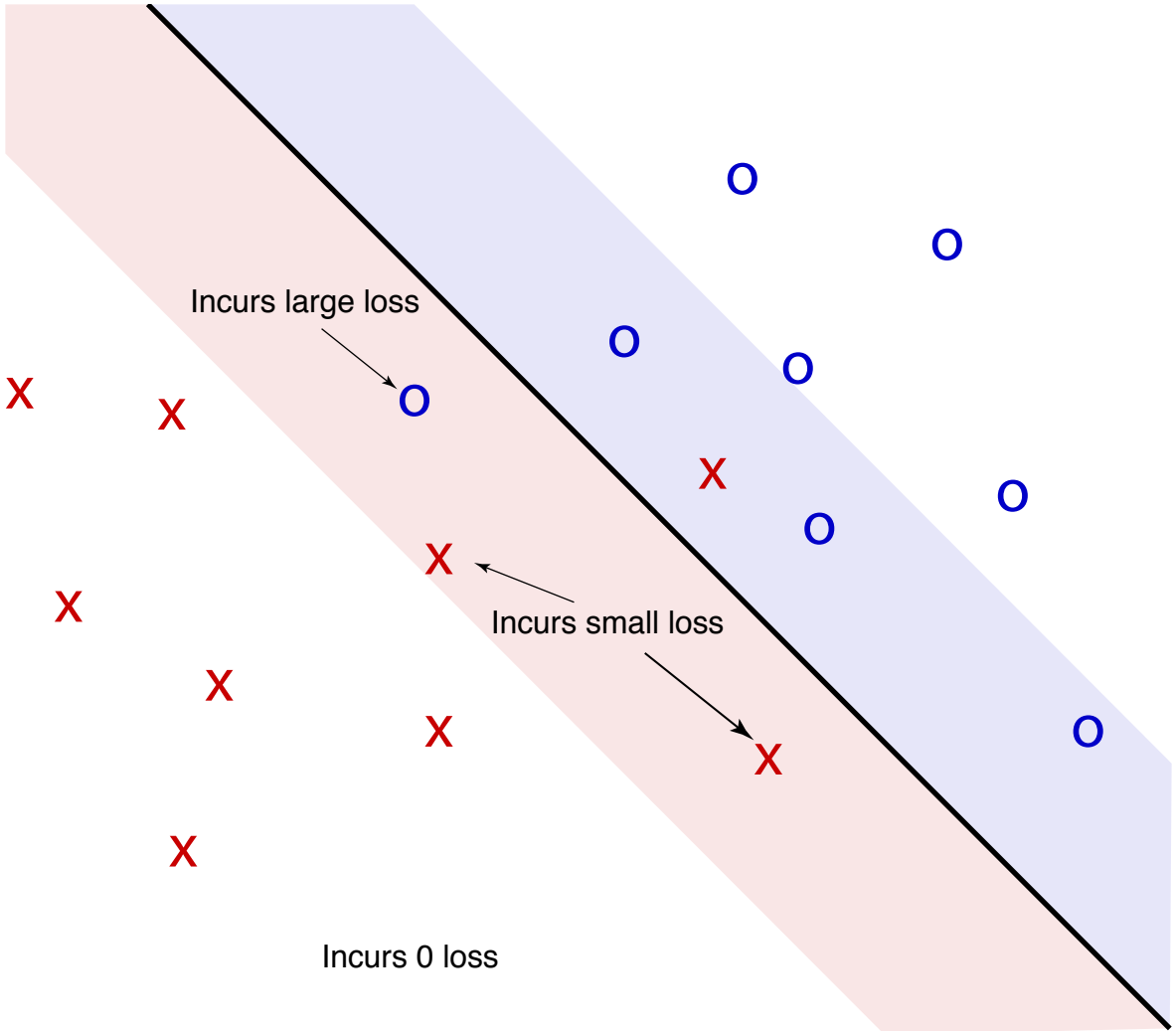
Support vector machine

This is the picture we should have in mind:



Support vector machine

This is the picture we should have in mind:



The hinge loss function

The hinge loss is standardly defined for a binary output $y^{(i)} \in \{-1, 1\}$. If $y^{(i)} = 1$, then we would like $\mathbf{w}^T\mathbf{x}^{(i)} + b$ to be large and positive. If $y^{(i)} = -1$, then we would like $\mathbf{w}^T\mathbf{x}^{(i)} + b$ to be large and negative. The larger $a_i(\mathbf{x}^{(j)})$ is in the right direction, the larger the margin.

In this scenario, the hinge loss is for $x^{(i)}$ being in class $y^{(i)}$ is:

$$\text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) = \max(0, 1 - y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b))$$

$y^{(i)} = 1$ $\quad w^T x^{(i)} + b \gg 0 \quad \Rightarrow$ 1 − large pos

$y^{(i)} = 1$ $\quad w^T x^{(i)} + b = 0.3 \quad \Rightarrow \max(0, 1 - 0.3) = 0.7$

$y^{(i)} = 1$ $\quad w^T x^{(i)} + b = -1 \quad \Rightarrow \max(0, 1 - (-1)) = 2$

Support vector machine

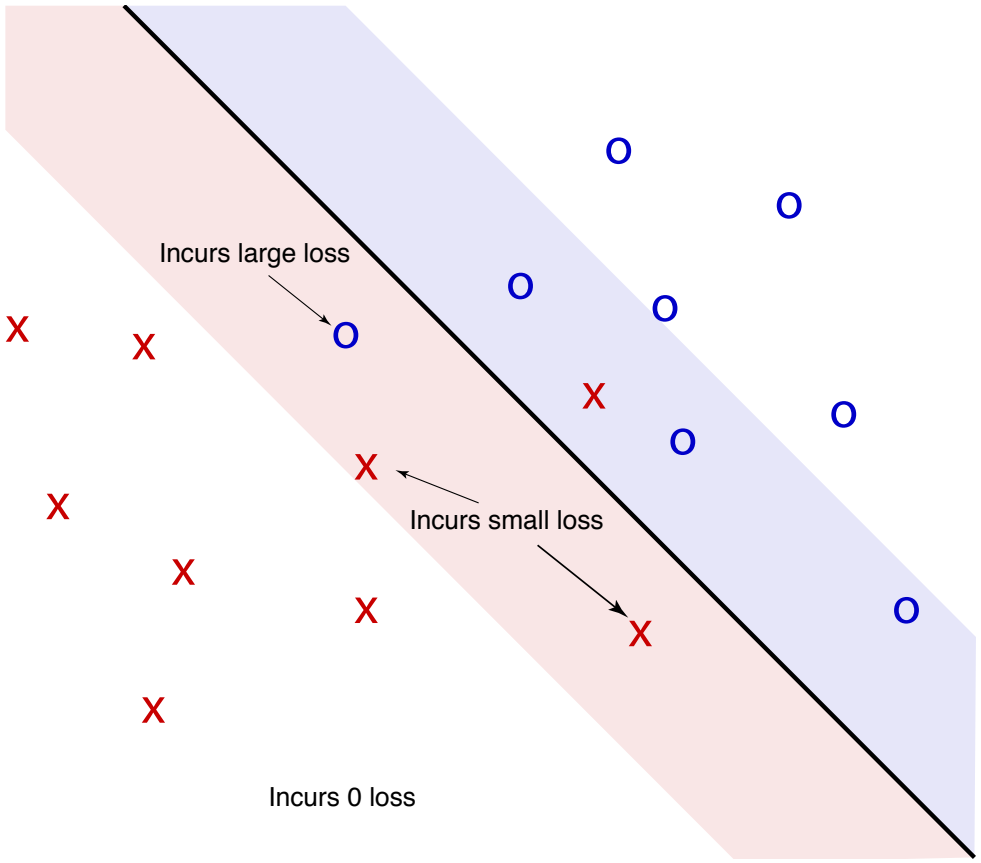
$$\text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) = \max(0, 1 - y^{(j)}(\mathbf{w}^T\mathbf{x}^{(i)} + b))$$

This is a loss, and hence something we wish to minimize. There are a few things to notice about the form of this function.

- If $\mathbf{w}^T\mathbf{x}^{(i)} + b$ and $y^{(i)}$ have the same sign, indicating a correct classification, then $0 \leq \text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) \leq 1$.
  - The error will be zero if $\mathbf{w}^T\mathbf{x}^{(i)} + b$ is large, corresponding to a large margin.
  - The error will be nonzero if $\mathbf{w}^T\mathbf{x}^{(i)} + b$ is small, corresponding to a small margin.
- If $\mathbf{w}^T\mathbf{x}^{(i)} + b$ and $y^{(i)}$ have opposite signs, then the hinge loss is non-negative, i.e., $\text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) = 1 + |\mathbf{w}^T\mathbf{x}^{(i)} + b|$.

Hinge loss intuition

The intuition of the prior slide is that the hinge loss is greatest for misclassifications, and the greater the error in misclassification, the worse the loss. For correct classifications, the loss will be zero only if there is a large enough margin.



Hinge loss extension

An extension of the hinge loss to multiple potential outputs is the following loss:

$$\text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) = \sum_{j \neq y^{(i)}} \max(0, 1 + a_j(\mathbf{x}^{(i)}) - a_{y^{(i)}}(\mathbf{x}^{(i)}))$$

for $a_j(\mathbf{x}^{(i)}) = \mathbf{w}_j^T \mathbf{x}^{(i)}$. Some intuitions, for the scenario that there are c classes:

- When the correct class achieves the highest score, $a_{y^{(i)}}(\mathbf{x}^{(i)}) \geq a_j(\mathbf{x}^{(i)})$ for all $j \neq y^{(j)}$, then $a_j(\mathbf{x}^{(i)}) - a_{y_{(i)}}(\mathbf{x}^{(i)}) \leq 0$ and

$$0 \leq \text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) \leq c - 1$$

- When an incorrect class, class i, achieves the highest score, then $a_j(\mathbf{x}^{(i)}) - a_{y^{(i)}}(\mathbf{x}^{(i)}) \geq 0$ and has the potential to be large.
- In both scenarios, it is still desirable to make the correct margins larger and the incorrect margins smaller.

The SVM cost function

If we let $\theta = \{\mathbf{w}_j\}_{j=1,\ldots,c}$, where there are c classes, we can now formulate the SVM optimization function. In particular, we want to minimize the hinge loss across all training examples. Then, to optimize θ for a linear kernel and hinge loss, we solve the following minimization problem:

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)})$$

which, for the sake of completeness, can be writen as:

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j \neq y^{(i)}} \max(0, 1 + a_j(\mathbf{x}^{(i)}) - a_{y^{(i)}}(\mathbf{x}^{(i)}))$$

Support vector machine

Is there a closed-form solution?

Support vector machine

Simple sanity check:



Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2

Score check: $\sum_{i \neq y^{(j)}} \max(0, 1 + a_i(\mathbf{x}^{(j)}) - a_{y^{(j)}}(\mathbf{x}^{(j)}))$

Support vector machine

Simple sanity check:



Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2

Cat: max(0, 1 - 2.1 + 3.4) + max(0, 1 - 2.1 - 2.0) = max(0, 2.3) + max(0, -3.1) = 2.3

Support vector machine

Simple sanity check:



Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	2.3		

Car: max(0, 1 - 5.1 + 0.2) + max(0, 1 - 5.1 +1.7) = max(0, -3.9) + max(0, -2.4) = 0

Support vector machine

Simple sanity check:



Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	0.3	0	

Bird: max(0, 1 + 1.2 + 2.3) + max(0, 1 + 1.2 + 3.1) = max(0, 4.5) + max(0, 5.3) = 9.8

Support vector machine

Simple sanity check:



Cat:	2.1	0.2	2.3
Car:	3.4	5.1	3.1
Bird:	-2.0	1.7	-1.2
Loss:	0.3	0	9.8

$$\sum_{i \neq y^{(j)}} \max(0, 1 + a_i(\mathbf{x}^{(j)}) - a_{y^{(j)}}(\mathbf{x}^{(j)}))$$