Lecture 4: Supervised Classification

Announcements:

- HW #1 is due in a week, Monday Jan 17, uploaded to Gradescope. To submit your Jupyter Notebook, print the notebook to a pdf with your solutions and plots filled in. The graders will grade your work from these notebooks. In the future, you will also have to print out code from any .py files that were modified.

- No lecture this Monday due to the MLK Holiday.

Maximum-likelihood introduction

Data: H T H H T T H T

Model: $x^{(i)} = \begin{cases} 0, & \text{w.p. } 1-\theta \\ 1, & \text{w.p. } \theta \end{cases}$

Likelihood:

Model 1: $\theta = 1$

Model 2: $\theta = 0.75$

Model 3: $\theta = 0.5$

Model 1: $1 \cdot 0 \cdot 1 \cdot 1 \cdot 0 \cdot 0 \cdot 1 \cdot 0 = 0$

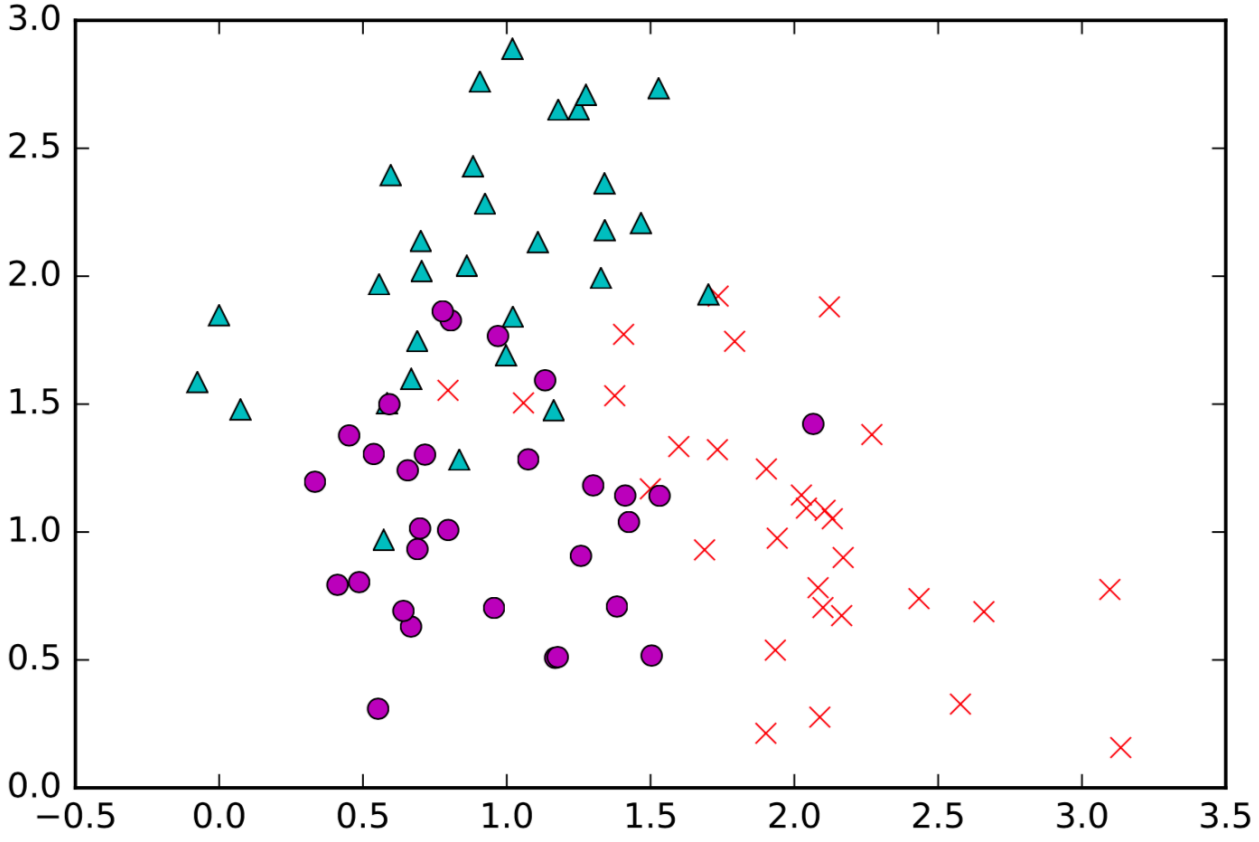
Model 2: $(0.75)^4 \cdot (0.25)^4 = 0.00124$

Model 3: $(0.5)^4 \cdot (0.5)^4 = 0.0039$

Maximum-likelihood estimation

Example of ML estimation

Say we receive paired data $\{\mathbf{x}^{(i)}, y^{(i)}\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^2$ is a data point that belongs to one of three classes, $y^{(i)} \in \{1, 2, 3\}$. Classes $1, 2, 3$ denote the three possible classes (or labels) that a data point could belong to. The drawing below (with appropriate labels) represents this:



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.

For example:

$\Pr(y=1|x) = 0.2$

$\Pr(y=2|x) = 0.7 \longrightarrow$ Class 2

$\Pr(y=3|x) = 0.1$

PROBLEM SETUP : We know $p(x|j)$ and $p(j) = 1/3$

$$\arg\max_j P_n(y=j|x) = \arg\max_j \frac{P_n(y=j, x)}{p(x)}$$

$$= \arg\max_j \frac{p(j)\, p(x|j)}{p(x)}$$

$$= \arg\max_j p(j)\, p(x|j) \quad [p(x) \text{ is common for all } j]$$

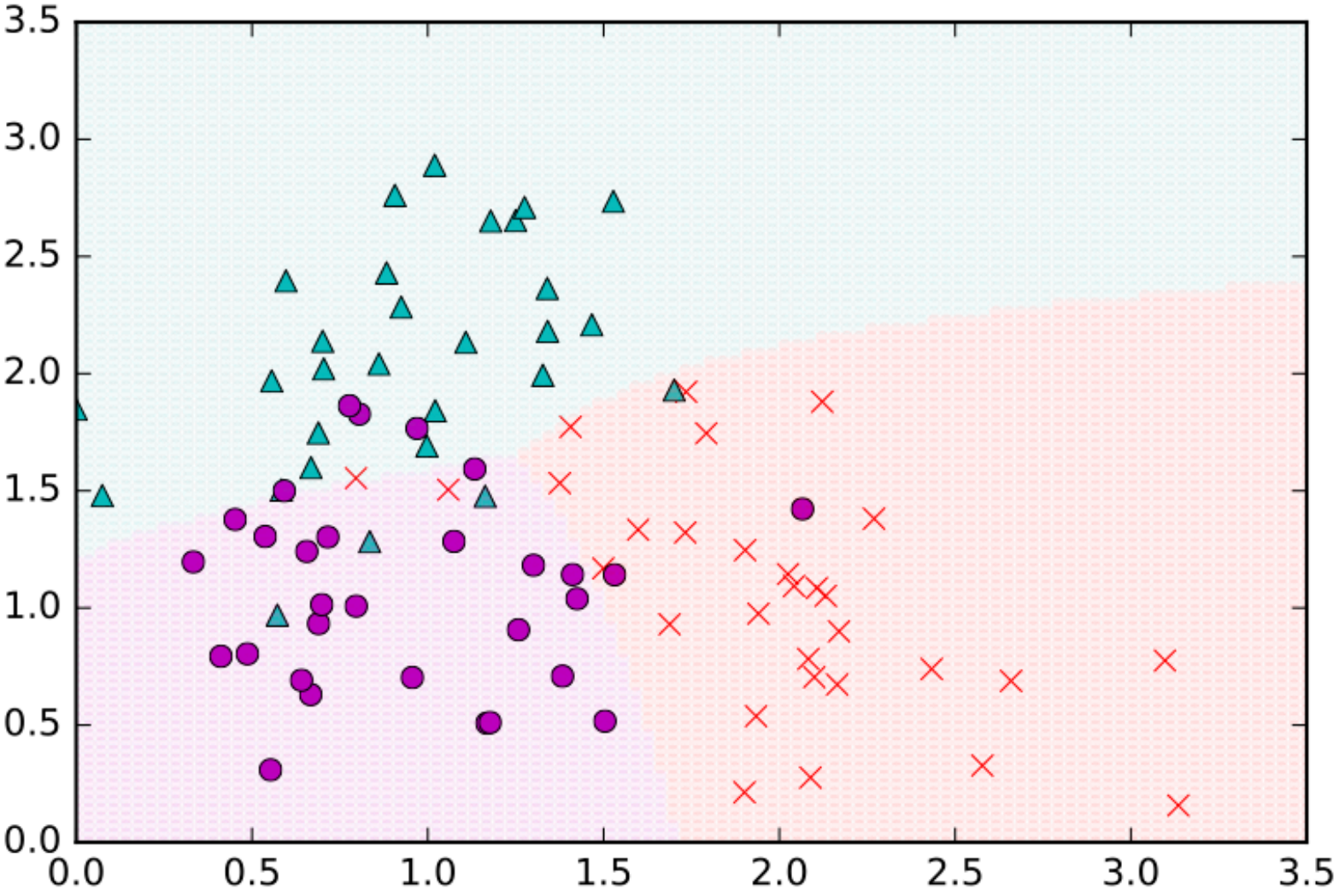
$$= \arg\max_j p(x|j) \quad [p(j) \text{ are all equal}]$$

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



Supervised classification & gradient descent principles

↳ softmax classifier

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

Reading

Reading:

Deep Learning, 5.7, 5.9, 5.10, 5.11.1.

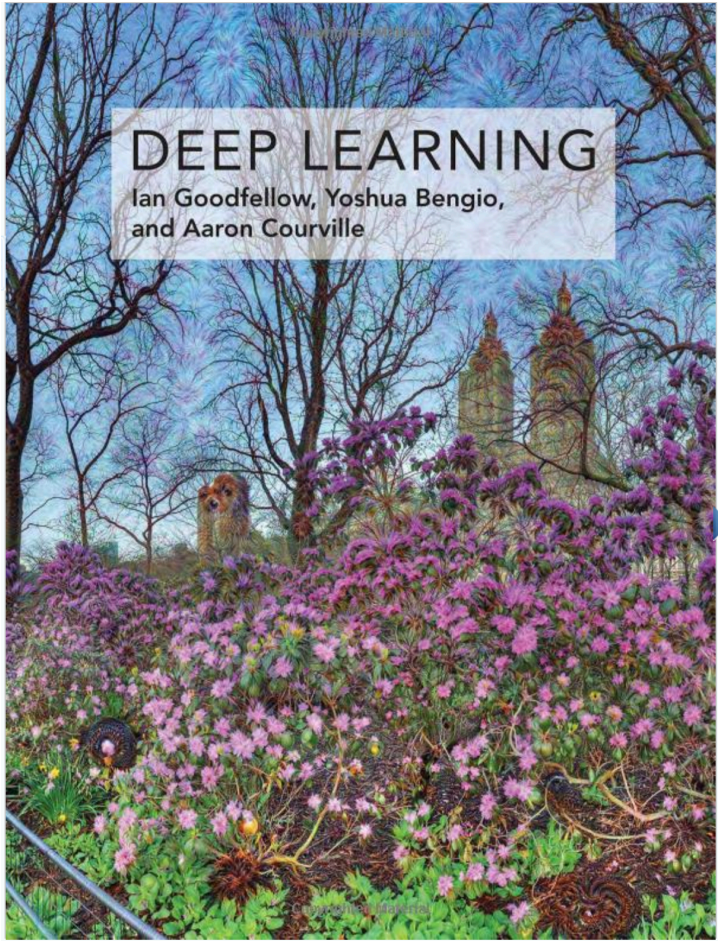
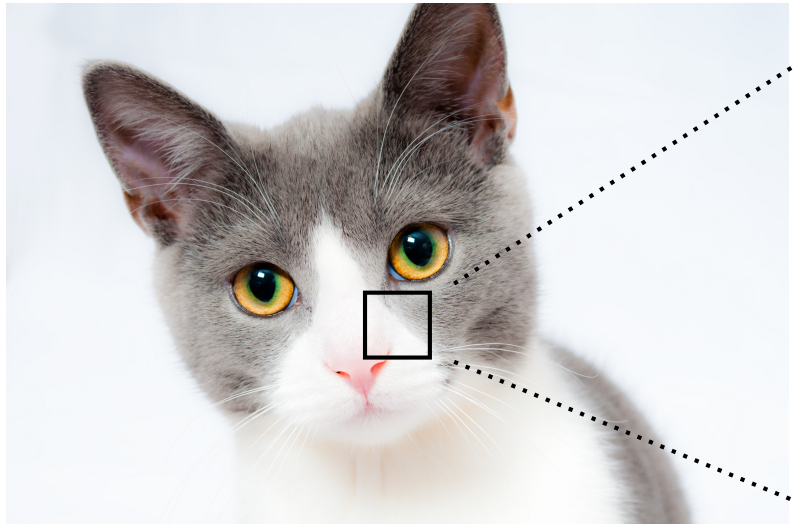


Image classification

Image classification



This image is licensed under CC0.

R, G, B

[0, 255]

```
[[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, 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]]
```

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.

Image classification

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

Image classification

Viewpoint variation:

Image classification

Viewpoint variation

ZOOMED IN

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 → model parameters

(Deep NN's, these parameters result in learning features that are optimized to classify an image).

TEST: Model → prediction

new image

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).
- 10 classes, with 6,000 images per class.
- 50,000 images are for training and 10,000 are for testing.

$32 \times 32 \times 3$

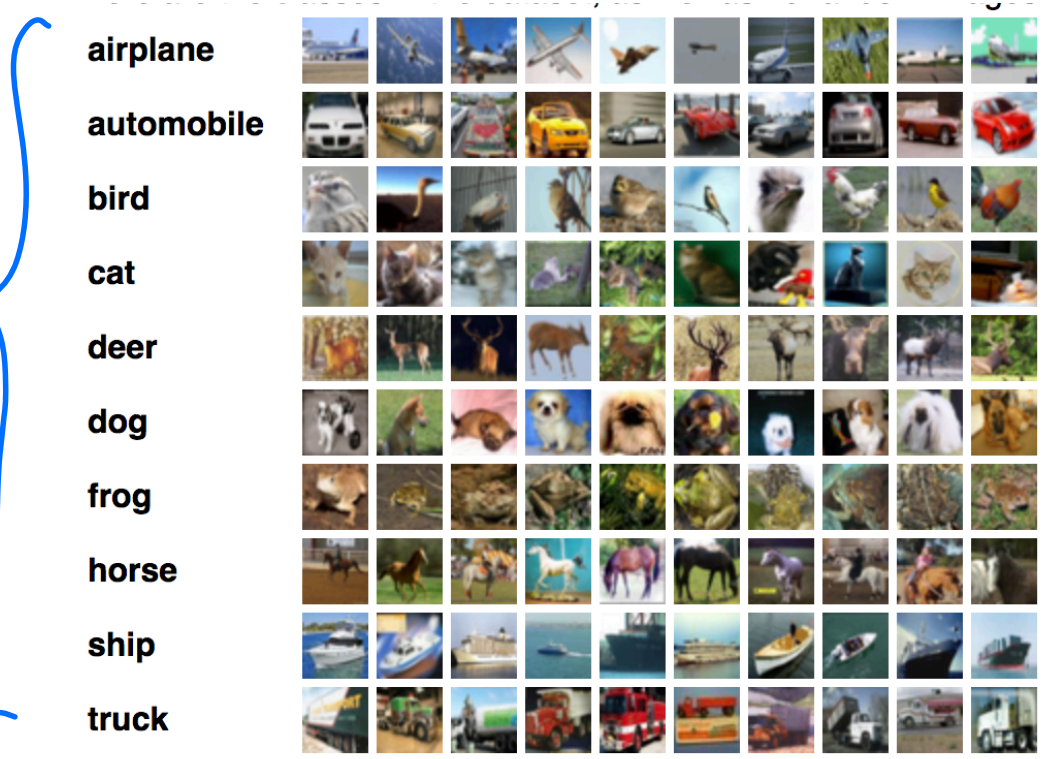
↓ reshape

3072

$x \in \mathbb{R}$

$y^{(i)}$

$y^{(i)} \in [1, 10]$



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, $\{y^{(1)}, y^{(2)}, \ldots, y^{(m)}\}$.

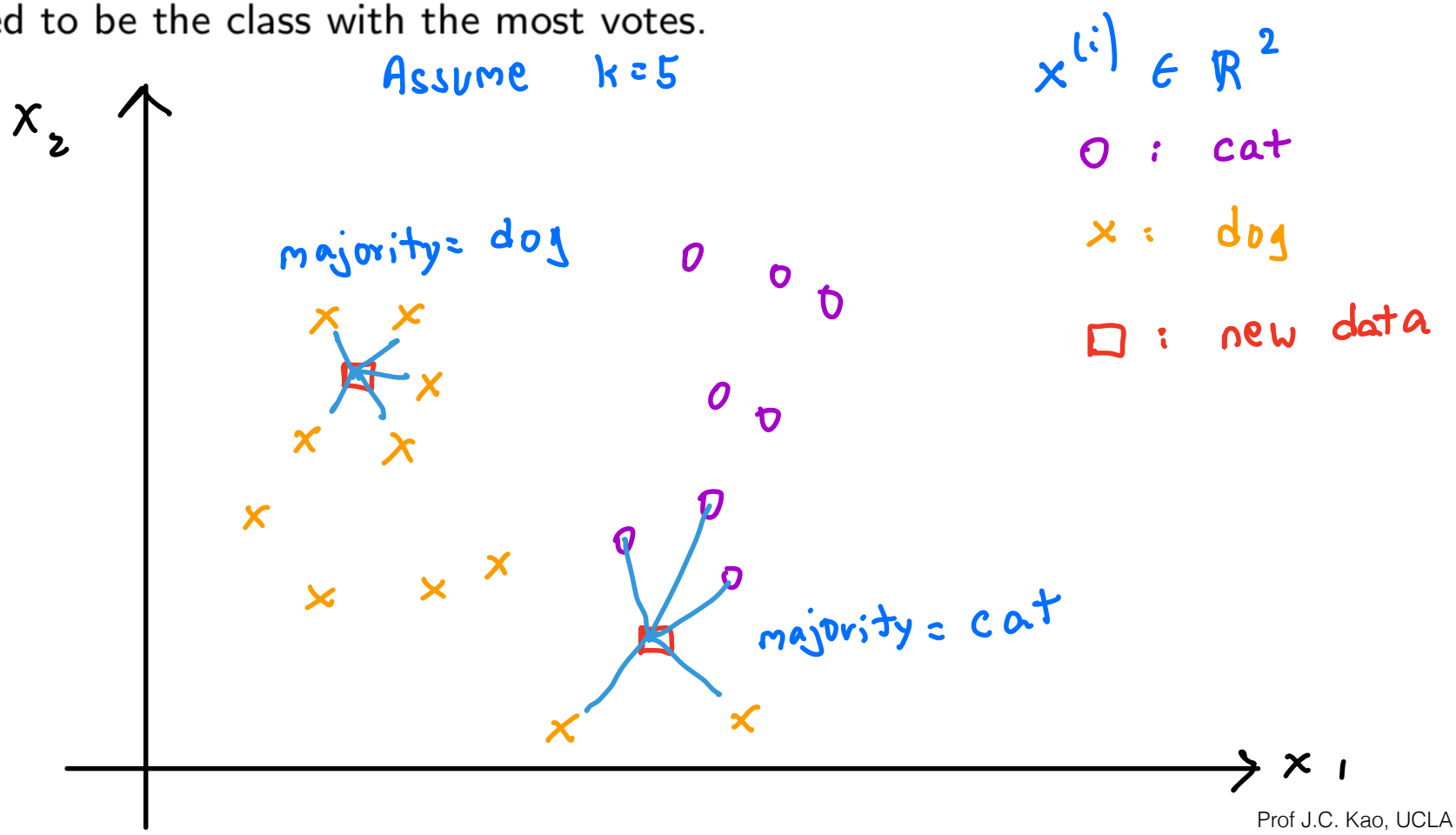
↳ $\mathbb{R}^{3072}$

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

hyperparameter

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.

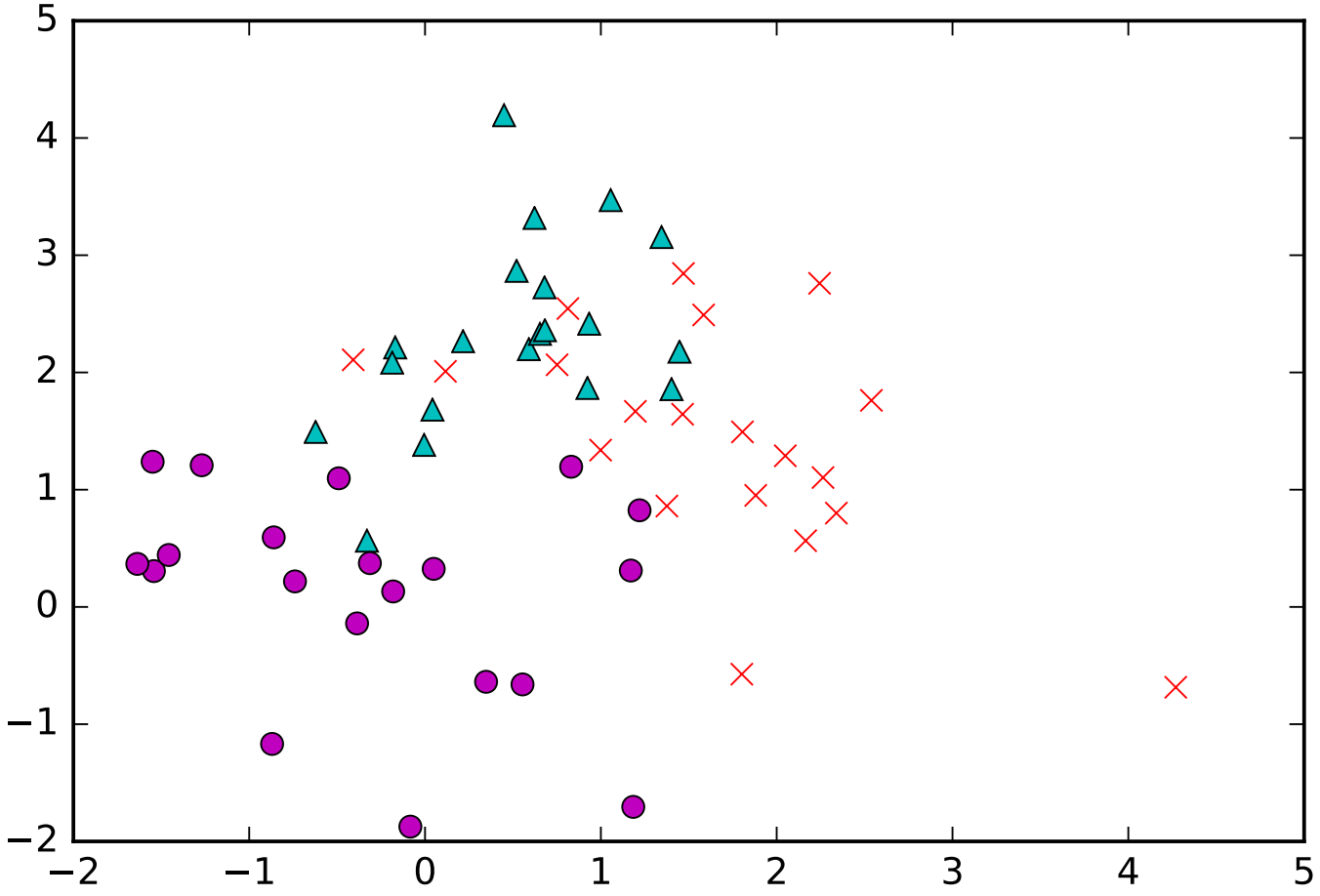


k-nearest neighbors, more formally

- 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., $d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \left\| \mathbf{x}^{(i)} - \mathbf{x}^{(j)} \right\|$. $= \sqrt{\sum_{k=1}^{n} (x_k^{(i)} - x_k^{(j)})^2}$
- 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, fast

Cons? Memory intensive because we need to store all the input data.

Example: Multivariate Classification

x_1, y_1 $x \rightarrow$ 2 #s

x_2, y_2 $y \rightarrow$ 4 #s

x_3, y_3

} 18 numbers totally

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
```

BROADCASTING

Pros? Simple

Cons? Scale with amount of training data

x_train : (2, N) array

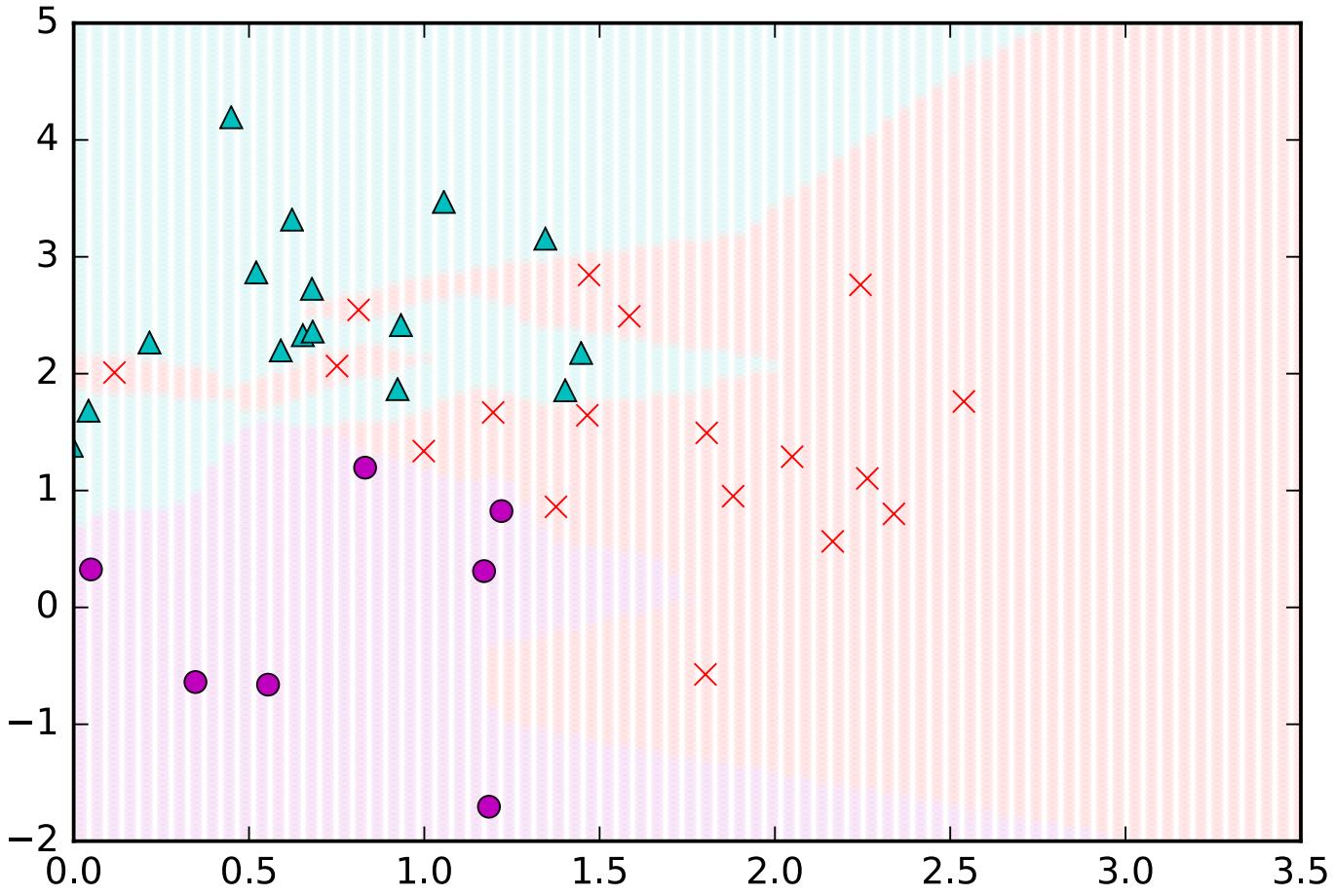
$x \in \mathbb{R}^2$

N examples

x_test : (2,) array

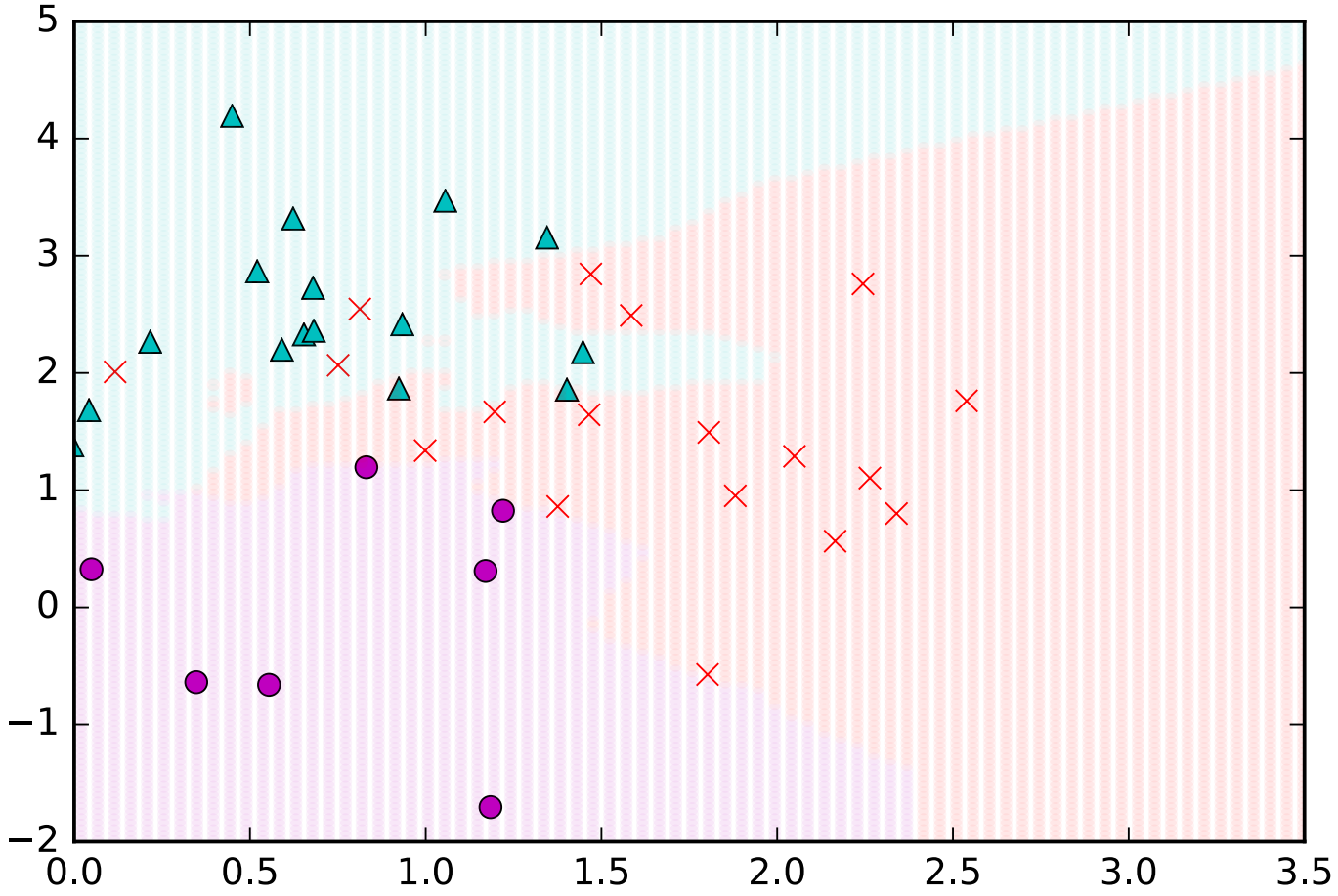
k-nearest neighbors

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, distance metric

(L_2, L_1, L_∞ ... norm)

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.

$x \in \mathbb{R}^2$

$x \in \mathbb{R}^{3072}$ → some dimensions might be more important.

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

Distance doesn't understand the perturbation.

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

$x \in \mathbb{R}^n$ → dimensionality of data

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 $\mathbf{y}$.

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

$c \times n$ $\quad n \times 1$ $\quad c \times 1$

10×3072 $\quad 3072 \times 1$ $\quad 10 \times 1$

score of class 1

score of class c

10×1

CIFAR-10

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

$y \in \mathbb{R}^{10}$ – correspond to scores for 10 classes

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

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

Classifiers based on linear classification

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

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

assume norm = 1

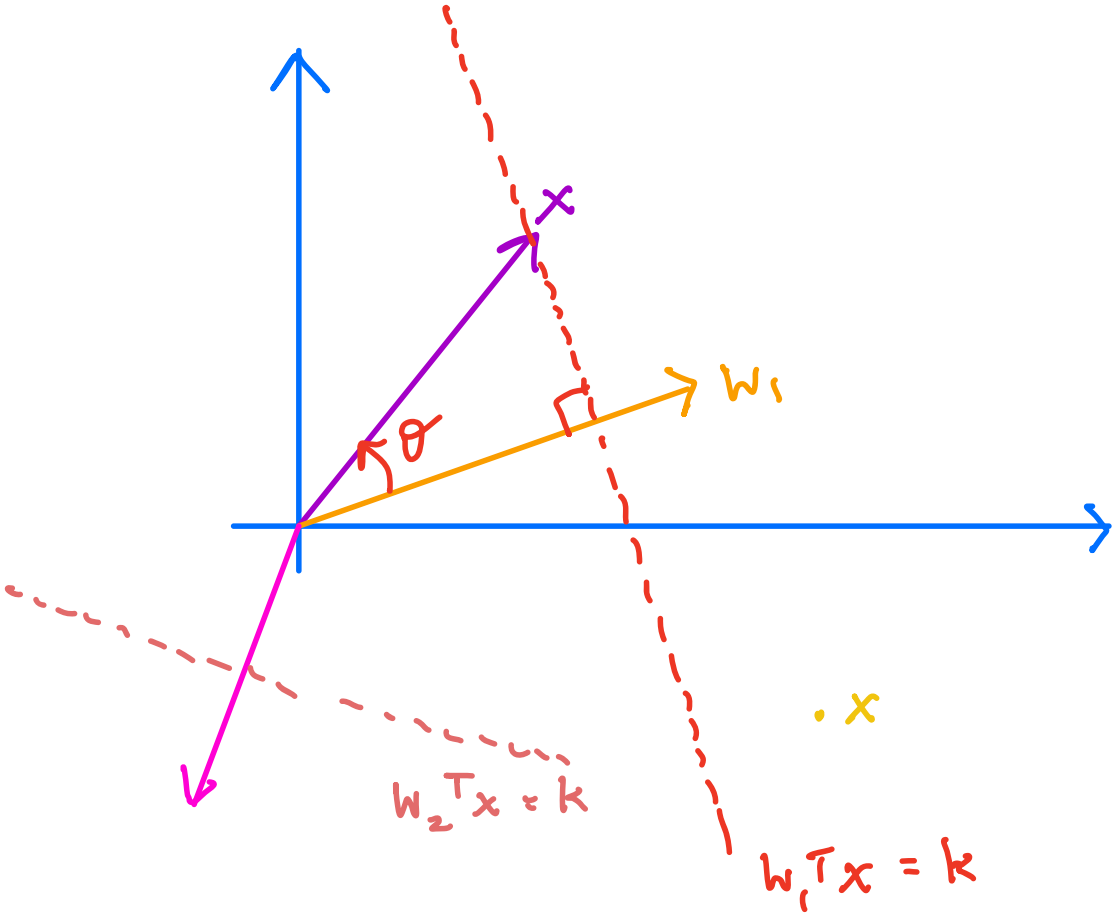
to maximize $w^{(i)}$ and x should overlap

weight of the class of x.

Why b?

Classifiers based on linear classification

What is a linear classifier doing?



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

$y = w_1^T x$

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

Assuming

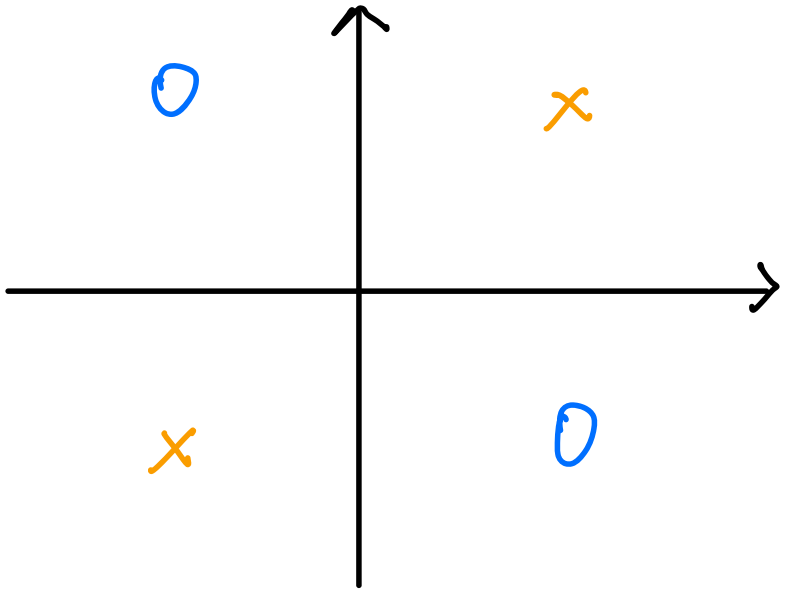
$\|w_1\| = 1$

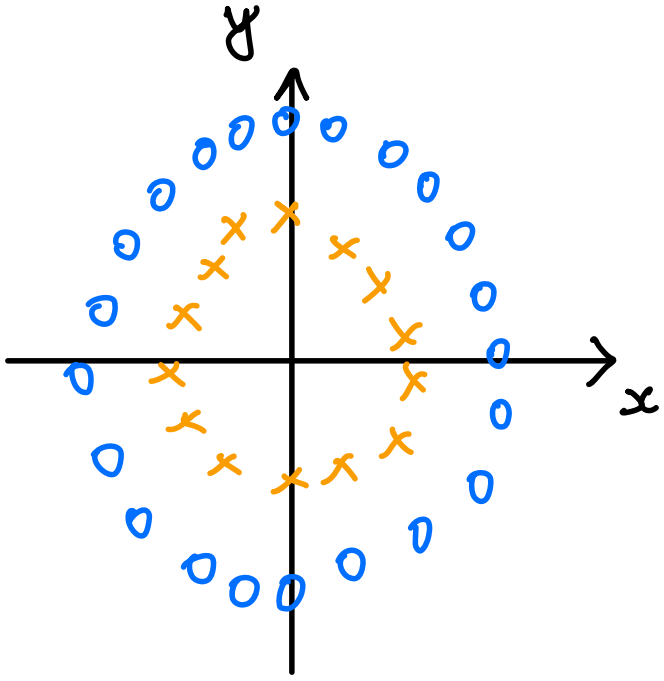
$w_1^T x = \|x\| \cos\theta$

if $w_1^T x > w_2^T x$, x is classified in class 1.

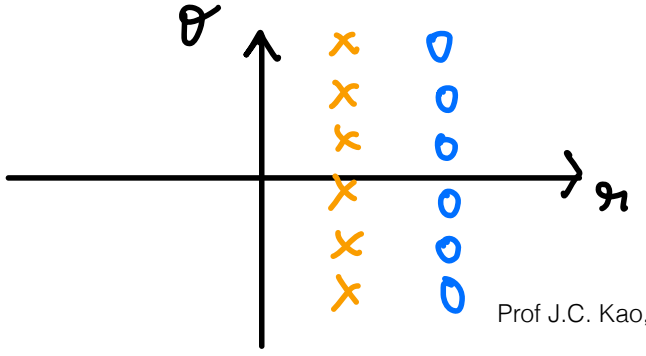
Classifiers based on linear classification

Where might linear classifiers fail?





we can convert to polar coordinates



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?

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, $\text{softmax}_i(\mathbf{x})$, so that:

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

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

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

Assume $b = 0$

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

$$\text{softmax}_1(x) = \frac{e^{w_1^T x}}{e^{w_1^T x} + e^{w_2^T x}} \qquad \text{softmax}_2(x) = \frac{e^{w_2^T x}}{e^{w_1^T x} + e^{w_2^T x}}.$$

Softmax classifier

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

for $a_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + \cancel{w_i}$ b_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,

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

↓

Probability that data point $x^{(j)}$ comes from class i.