

Classification with Perceptrons

Perceptrons as simplified “neurons”

w_0 is called the “**bias**”

$-w_0$ is called the “**threshold**”

Input is $(x_1, x_2, \dots x_n)$

Weights are $(w_1, w_2, \dots w_n)$

Output y is +1 (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:

If $w_1x_1 + w_2x_2 + \dots + w_nx_n > \text{threshold}$

then $y = 1$, else $y = -1$

If $w_1x_1 + w_2x_2 + \dots + w_nx_n > -w_0$

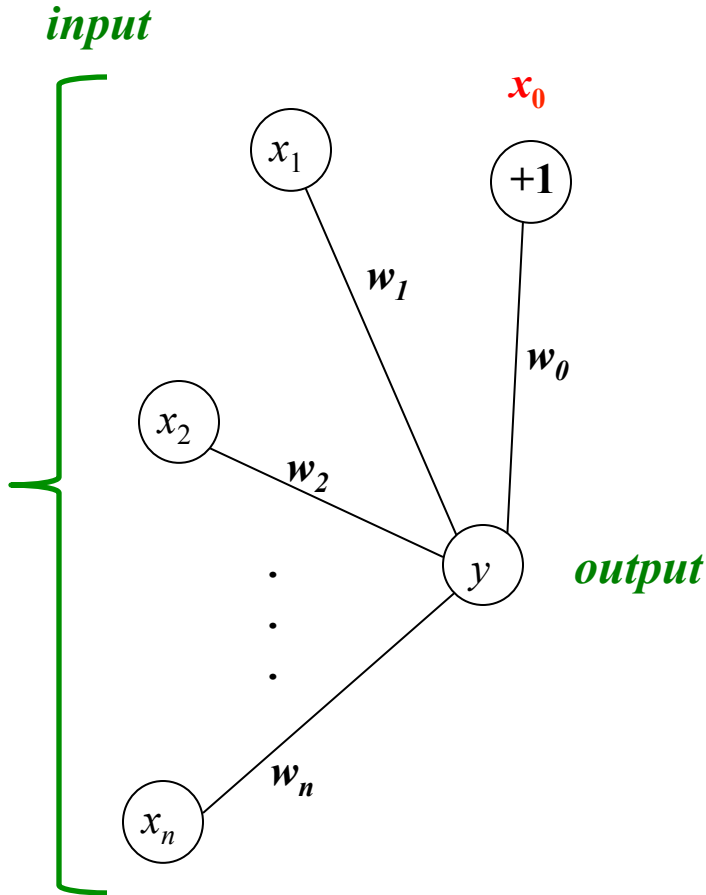
then $y = 1$, else $y = -1$

If $w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n > 0$

then $y = 1$, else $y = -1$

If $w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n > 0$

then $y = 1$, else $y = -1$

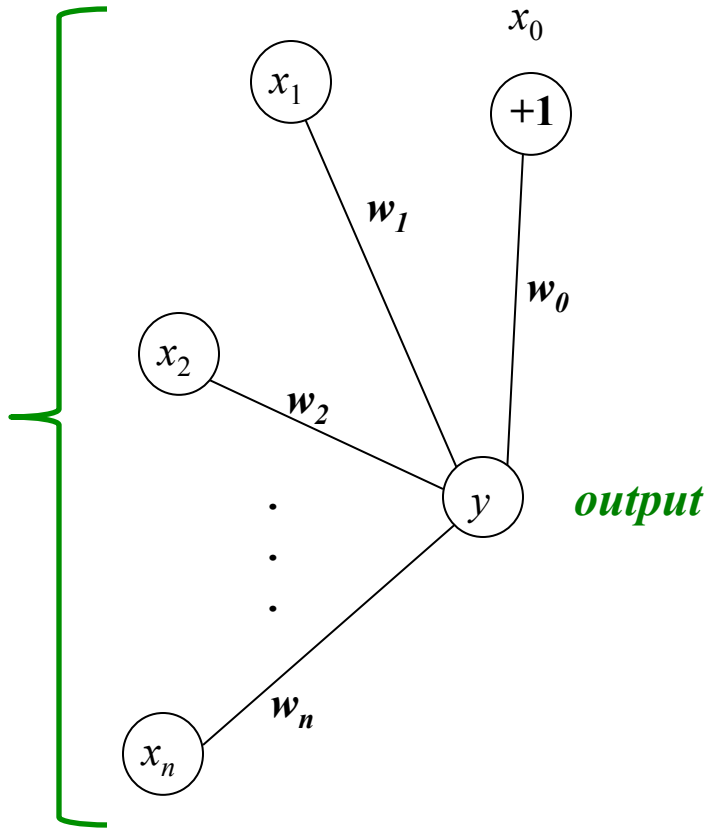


Perceptrons as simplified “neurons”

w_0 is called the “**bias**”

$-w_0$ is called the “**threshold**”

input



Input is $(x_1, x_2, \dots x_n)$

Weights are $(w_1, w_2, \dots w_n)$

Output y is $+1$ (“the neuron fires”) if the sum of the inputs times the weights is greater or equal to the threshold:

$$\text{output} = y(\mathbf{x}) = \text{sgn}(w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n)$$

$$\text{where } \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ +1 & \text{if } z \geq 0 \end{cases}$$

Let $\mathbf{x} = (x_0, x_1, x_2, \dots x_n)$
 $\mathbf{w} = (w_0, w_1, w_2, \dots w_n)$

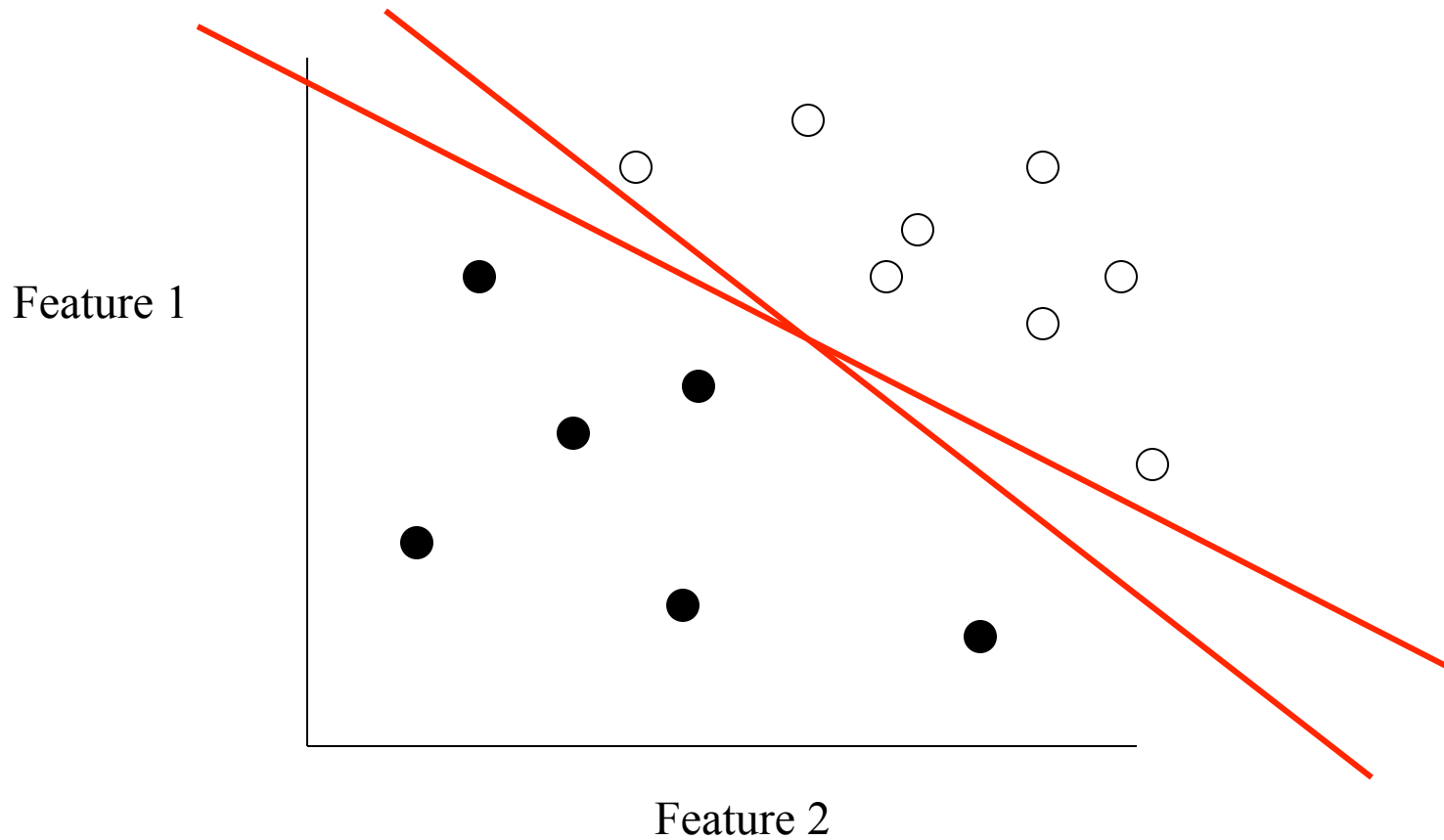
Then:

$$y = \text{sgn}(\mathbf{w} \cdot \mathbf{x})$$

Decision Surfaces

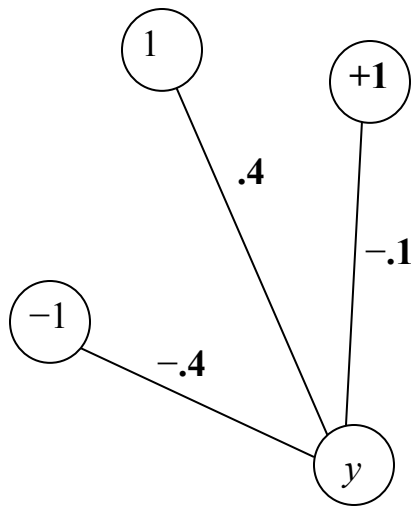
- Assume data can be separated into two classes, positive and negative, by a linear *decision surface*.
- Assuming data is n -dimensional, a perception represents a $(n-1)$ -dimensional hyperplane that separates the data into two classes, *positive* (+1) and *negative* (-1).

Example where line won't work?

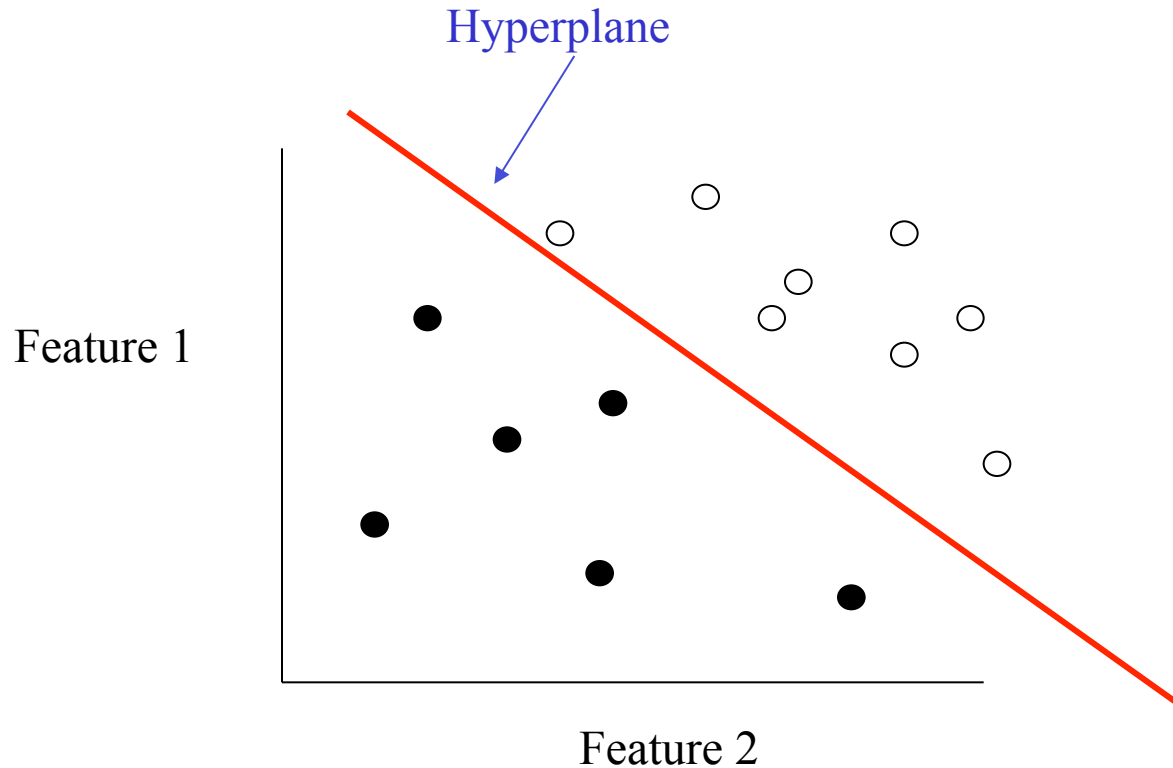


Example

- What is the predicted class y ?



Geometry of the perceptron

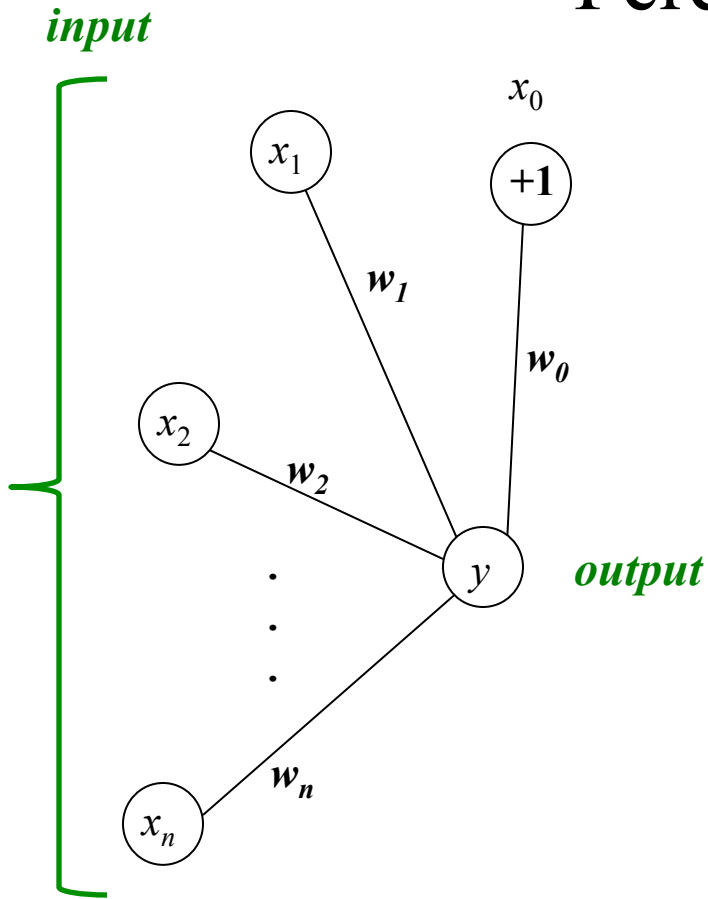


In 2D:

$$w_1x_1 + w_2x_2 + w_0 = 0$$

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$$

Perceptron Learning



Goal is to use the training data to learn a set of weights that will:

- (1) correctly classify the training data
- (2) generalize to unseen data

Input instance: $\mathbf{x}^k = (x_1, x_2, \dots, x_n)$,
with target class $t^k \in \{-1, 1\}$

Perceptron Learning

Learning is often framed as an optimization problem:

- Find \mathbf{w} that minimizes a “loss” function:

$$J(\mathbf{w}) = \frac{1}{M} \sum_{k=1}^M L(\mathbf{w} \cdot \mathbf{x}^k, t^k)$$

where M is number of training examples.

One part of the “art” of ML is to define a good loss function.

- Here, define the loss function as follows:

$$L(\mathbf{w} \cdot \mathbf{x}^k, t^k) = \max\left(0, -(\mathbf{w} \cdot \mathbf{x}^k) t^k\right)$$

That is, if the prediction is correct, $L = 0$. Otherwise, $L =$ “confidence” of incorrect prediction.

How to solve this minimization problem? **Gradient descent.**

Gradient descent

- To find direction of steepest descent, take the derivative of $J(\mathbf{w})$ with respect to \mathbf{w} .
- A vector derivative is called a “gradient”: $\nabla J(\mathbf{w})$

$$\nabla J(\mathbf{w}) = \left[\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \dots, \frac{\partial J}{\partial w_n} \right]$$

- Here is how we change each weight:

For $i = 0$ to n :

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = -\eta \frac{\partial J}{\partial w_i}$$

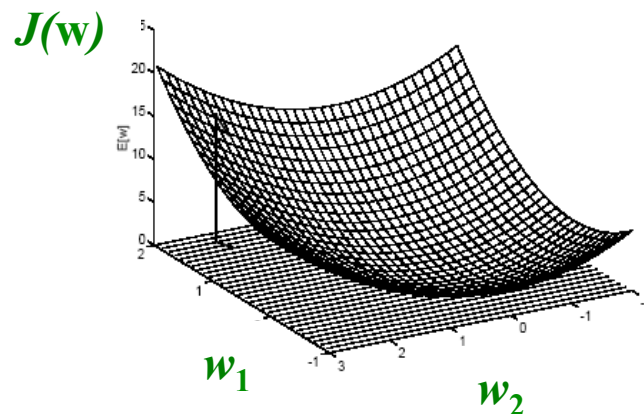
and η is the *learning rate* (i.e., size of step to take downhill).

“True” (or “batch”) gradient descent

- One *epoch* = one iteration through the training data.
- After each epoch, compute average loss over the training set:

$$J(\mathbf{w}) = \frac{1}{M} \sum_{k=1}^M L(\mathbf{w} \cdot \mathbf{x}^k, t^k) = \frac{1}{M} \sum_{k=1}^M \max(0, -(\mathbf{w} \cdot \mathbf{x}^k)) t^k$$

- Change the weights to move in direction of steepest descent in the average “loss” surface:



From T. M. Mitchell, *Machine Learning*

- Problem with *true gradient descent*:

Training process is slow.

Training process can land in local optimum.

- Common approach to this: use *stochastic gradient descent*:
 - Instead of doing weight update after all training examples have been processed, do weight update after each training example has been processed (i.e., perceptron output has been calculated).
 - Stochastic gradient descent approximates true gradient descent increasingly well as $\eta \rightarrow 1/\infty$.

Derivation of perceptron learning rule (stochastic gradient descent)

$$J(\mathbf{w}) = \frac{1}{M} \sum_{k=1}^M \max \left(0, -(\mathbf{w} \cdot \mathbf{x}^k) t^k \right) \quad \text{where } k \text{ indexes training examples}$$

$$J_k(\mathbf{w}) = \max \left(0, -(\mathbf{w} \cdot \mathbf{x}^k) t^k \right)$$

$$\frac{\partial J_k}{\partial w_i} = \begin{cases} 0 & \text{if } (\mathbf{w} \cdot \mathbf{x}^k) t^k > 0, \text{ that is, } \mathbf{x}^k \text{ was classified correctly} \\ -x_i^k t^k & \text{otherwise} \end{cases}$$

So, after processing training example k , if perceptron misclassified the example:

$$\Delta w_i = -\eta \frac{\partial J}{\partial w_i} = \eta x_i^k t^k$$

Perceptron Learning Algorithm

Start with small random weights, $\mathbf{w} = (w_0, w_1, w_2, \dots, w_n)$, where $w_i \in [-1, 1]$.

Repeat until accuracy on the training data stops increasing or for a maximum number of *epochs* (iterations through training set):

For $k = 1$ to M (total number in training set):

1. Select next training example (\mathbf{x}^k, t^k) .
2. Run the perceptron with input \mathbf{x}^k and weights \mathbf{w} to obtain y .
3. If $y \neq t^k$, update weights:
for $i = 0, \dots, n$: ; Note that bias weight w_0 is changed just like all other weights!

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta x_i^k t^k$$

4. Go to 1.

A Letter Recognition Task

(Frey & Slate, 1991)

Instances:

- 20,000 unique letter images were generated by randomly distorting pixel images of the 26 uppercase letters from 20 different commercial fonts

A A A A A A A A A A
 B B B B B B B B B B
 C C C C C C C C C C
 F F F F F F F F F F
 K K K K K K K K K K
 S S S S S S S S S S
 X x X x X x X x X x

Figure 1. Examples of the character images generated by "warping" parameters.

Instances:

- 20,000 unique letter images were generated by randomly distorting pixel images of the 26 uppercase letters from 20 different commercial fonts
- Each image is $\sim 45 \times 45$ pixels. Each pixel is either black (“on”) or white (“off”).

Features: 16 attributes of a pixel-array image

1. The horizontal position, counting pixels from the left edge of the image, of the center of the smallest rectangular box that can be drawn with all "on" pixels inside the box.
2. The vertical position, counting pixels from the bottom, of the above box.
3. The width, in pixels, of the box.
4. The height, in pixels, of the box.
5. The total number of "on" pixels in the character image.
6. The mean horizontal position of all "on" pixels relative to the center of the box and divided by the width of the box. This feature has a negative value if the image is "left-heavy" as would be the case for the letter L.

Etc. See their paper (linked from the class website)

- **Feature scaling:** Each feature is scaled to be in the range 0 – 15
- **Data format:** Each row of letter-recognition.data corresponds to a single instance (i.e., letter image). The first “value” on the row is the letter category. The next 16 values are the feature values.

T,2,8,3,5,1,8,13,0,6,6,10,8,0,8,0,8
 I,5,12,3,7,2,10,5,5,4,13,3,9,2,8,4,10
 D,4,11,6,8,6,10,6,2,6,10,3,7,3,7,3,9
 N,7,11,6,6,3,5,9,4,6,4,4,10,6,10,2,8
 G,2,1,3,1,1,8,6,6,6,6,5,9,1,7,5,10
 S,4,11,5,8,3,8,8,6,9,5,6,6,0,8,9,7
 B,4,2,5,4,4,8,7,6,6,7,6,6,2,8,7,10
 A,1,1,3,2,1,8,2,2,2,8,2,8,1,6,2,7
 J,2,2,4,4,2,10,6,2,6,12,4,8,1,6,1,7
 M,11,15,13,9,7,13,2,6,2,12,1,9,8,1,1,8

Class Distribution:

789 A 766 B 736 C 805 D 768 E 775 F
 773 G 734 H 755 I 747 J 739 K 761 L
 792 M 783 N 753 O 803 P 783 Q 758 R
 748 S 796 T 813 U 764 V 752 W 787 X
 786 Y 734 Z

Dataset in UCI ML Repository

<http://archive.ics.uci.edu/ml/datasets/Letter+Recognition>

State of the art accuracy on this set is about 95%.

Multiclass classification with perceptrons

Two main methods:

- **One vs. All** (also called “One vs. Rest”)
- **All-pairs** (also called “One vs. One”)

One vs. All method

Training:

Train 26 perceptrons:

- “A” vs. “not A”
- “B” vs. “not B”
- etc.

Testing:

Given instance \mathbf{x} :

- Run each perceptron on \mathbf{x} .
- If class is positive, record the *decision value*: $d = \mathbf{w} \cdot \mathbf{x}$
- Return the class with highest decision value (breaking ties at random)

Problems: (1) scale of decision values might be different for different binary classifiers; (2) unbalanced training data

All-Pairs method

Training:

Train separate perceptron for each pair of classes. For k classes, $(k(k-1)/2)$ perceptrons.

“A” vs. “B”	“B” vs. “C”	
“A” vs. “C”	“B” vs. “D”	...
“A” vs. “D”	“B” vs. “E”	

etc.

Testing:

Given instance \mathbf{x} :

- Run each perceptron on \mathbf{x} . Collect “votes” for each class.
- Return class that received most votes (breaking ties at random)

Example of All-Pairs method

Suppose you are classifying “A” vs. “B” vs. “C” vs. “D”

A-B classifier says “A”

What is the class?

A-C classifier says “C”

A-D classifier says “A”

B-C classifier says “B”


B-D classifier says “D”


C-D classifier says “D”

Homework 1

1. Split data for each letter into approx. 50% training, 50% testing.
2. Implement *all-pairs* method with perceptrons for the letter-recognition task.
3. See HW writeup for description of lab report and what to turn in.

Confusion Matrix

Predicted class 

Actual class 

	A	B	C	D	E	F	G	H	I
A									
B									
C									
D									
E									
F									
G									
H									
I									

Confusion Matrix

Actual class ↓

Predicted class →

	A	B	C	D	E	F	G	H	I
A									
B									
C									
D									
E									
F									
G									
H									
I									

Results

True	Predicted
A	A
A	B
A	B
A	A
B	C
B	A
B	A
B	C
C	C
C	C
C	C
C	C

Quiz on Tuesday

Time allotted: 30 minutes.

Format: You are allowed to bring in one (double-sided) page of notes to use during the quiz. You may bring/use a calculator, but you don't really need one.

What you need to know:

- Perceptrons: input, output, weights, bias, threshold
- Given an input vector \mathbf{x} and perceptron weights, how to calculate output
- Given perceptron weights, how to sketch separation line (in two dimensions)
- Perceptron learning algorithm
- Difference between “true” and “stochastic” gradient descent
- How to interpret / fill in a confusion matrix
- How “all-pairs” classification method works

Questions will be similar to examples and exercises we've done in class.