CS440/ECE448 Lecture 16: Linear Classifiers

Mark Hasegawa-Johnson, 3/2019
and Julia Hockenmaier 3/2019
Including Slides by
Svetlana Lazebnik, 10/2016

Aliza Aufrichtig @alizauf · Mar 4

Garlic halved horizontally = nature's Voronoi diagram?

en.wikipedia.org/wiki/Voronoi_d...

Classification as a learning problem

- We want assign **one of k class labels** (spam/no spam; hippo/horse/...) to **items** (emails, images, …)
- We assume that we have a **set of labeled examples**: (x_i, y_i)…..(x_N, y_N) (x: item, y: label)
- We use a subset of these labeled examples as **training data (supervised learning)**
- We use a *disjoint* subset of these labeled examples as **test data**
  We evaluate how often we assign the correct label to *unseen* examples.
  We are not allowed to optimize our models on the test data
- We may also use a separate disjoint subset as **development data** to tune our models

Linear Classifiers

- Naïve Bayes/BoW classifiers
- Linear Classifiers in General
- Perceptron
- Differential Perceptron/Neural Net

Naïve Bayes

- Naïve Bayes for text classification: two modeling choices
- Parameter estimation: how do we train our model?

Naïve Bayes for text data

Task: Assign class label c (from a fixed set $C = \{c_1 \ldots c_k\}$ to document d_i

Probabilistic reasoning behind Naïve Bayes:
Assign the most likely class label c to document d_i

$$\mathrm{argmax}_c \, \mathrm{P}(\mathrm{C} = \mathrm{c} \mid \mathrm{D} = \mathrm{d_i}) = \mathrm{argmax}_c P(\, \mathrm{D} = \mathrm{d_i} \mid \mathrm{C} = \mathrm{c}) P(C = c)$$

- c is one of k outcomes of random variable C
  P(C = c) is a categorical distribution over k outcomes.
- But what about P (D = d_i | C=c)?
  How do we model documents as random variables?

Learning $P(C = c)$

- This is the probability that a randomly chosen document from our data has class label c.
- P(C) is a categorical random variable over k outcomes $c_1 \ldots c_k$
- How do we set the parameters of this distribution?
- Given our training data of labeled documents,
  We can simply set $P(C = c_i)$ to the fraction of documents
  that have class label c_i
- This is a **maximum likelihood estimate**:
  Among all categorical distributions over k outcomes,
  this assigns the highest probability (likelihood) to the training data

Documents as random variable

- We assume **a fixed vocabulary** V of M word types: V = {apple, …, zebra}.
- A **document** d_i = "The lazy fox…" is a **sequence of n word tokens**

  $d_i = w_{i1} \ldots w_{iN}$

  The same word type may appear multiple times in d_i.
- Choice 1: We model d_i as a **set of word types**:

  $\forall v_j \in V$: what's the probability that v_j occurs/doesn't occur in d_i?

  We treat $P(v_j)$ as a Bernoulli random variable
- Choice 2: We model d_i as a **sequence of word tokens**:

  $\forall n_{n=1 \ldots N}$: what's the probability that $w_{in} = v_j$ (rather than any other $v_{j'}$)

  We treat $P(w_{in})$ as a categorical random variable (over V)

Modeling documents as sequences of tokens

Given a vocabulary of M word types, we model each document d_i
as a sequence of N categorical variables $w_{i1} \ldots w_{iN}$

What's the probability that the n-th token in d is word type v_m?

Independence assumptions:

We ignore the position of each token

All tokens are conditionally independent given the class label

We just need a single categorical distribution $P(w = v_m \mid C = c)$ per class c:

$$P(\mathrm{D} = w_{i1} \ldots w_{iN} \mid C = c) = \prod_{n=1 \ldots N} \mathrm{P}(\mathrm{w} = \mathrm{v}_m \mid \mathrm{C} = \mathrm{c})$$

How do we estimate the parameters of this distribution?

$\mathrm{P}(\mathrm{w} = \mathrm{v}_m \mid \mathrm{C} = \mathrm{c})$ is the fraction of tokens in documents of class c
that are equal to vm

Modeling documents as sets of word types

Given a vocabulary of M word types, we model each document d_i as a set of M Bernoulli random variables: v_m = true if v_m occurs in d_i

Define an indicator variable $\mathbf{1}_{\text{statement}}$:

$\mathbf{1}_{\text{statement}} = 1$ if the statement is true
$\mathbf{1}_{\text{statement}} = 0$ if the statement is false

$$P(\mathrm{D} = \mathrm{d_i} \mid \mathrm{C} = \mathrm{c}) = P(\mathrm{D} = \{\mathrm{v1}, \neg\mathrm{v2}, \ldots\} \mid \mathrm{C} = \mathrm{c})$$

=

$$\prod_{j=1}^{N} (1_{vj\text{ occurs in }di}\, P(v_j \mid C = c) + 1_{vj\text{ does not occur in }di}\, P(\neg v_j \mid C = c))$$

How do we estimate each of our $P(V_j \mid C = c)$ distributions?

$P(v_j \mid C = c)$ is the fraction of training documents of class c in which v_j occurs.

Naïve Bayes/Bag-of-Words

- Model parameters: feature likelihoods P(word | class) and priors P(class)
  - How do we obtain the values of these parameters?
  - Need *training set* of labeled samples from both classes

$$\text{P(word | class)} = \frac{\text{\# of occurrences of this word in docs from this class}}{\text{total \# of words in docs from this class}}$$

- This is the *maximum likelihood* (ML) estimate, or estimate that maximizes the likelihood of the training data:

$$\prod_{d=1}^{D}\prod_{i=1}^{n_d} P(w_{d,i} \mid class_{d,i})$$

d: index of training document, *i*: index of a word

Indexing in BoW: Types vs. Tokens

- Indexing the training dataset: TOKENS
  - i = document token index, $1 \leq i \leq m$ (there are m document tokens in the training dataset)
  - j = word token index, $1 \leq j \leq n$ (there are n word tokens in each document)
- Indexing the dictionary: TYPES
  - c = class type, $1 \leq c \leq C$ (there are a total of C different class types)
  - w = word type, $1 \leq w \leq V$ (there are a total of V words in the dictionary, i.e., V different word types)

Two Different BoW Algorithms

- **One bit per document**, per word type:
  - F_{iw} = 1 if word “w” occurs anywhere in the i’th document
  - F_{iw} = 0 otherwise
- **One bit per word** token, per word type:
  - F_{jw} = 1 if the j’th word token is “w”
  - F_{jw} = 0 otherwise

Example: “who saw who with who?”

$$F_{i,"who"} = 1$$

$$F_{j,"who"} = \{1,0,1,0,1\}$$

Feature = One Bit Per Document

- Features:
  - F_{iw} = 1 if word "w" occurs anywhere in the i'th document
- Parameters:
  - $\lambda_{cw} \equiv P(F_{iw} = 1 | C = c)$
  - Note this means that $P(F_{iw} = 0 | C = c) = 1 - \lambda_{cw}$
- Parameter Learning:

$$\lambda_{cw} = \frac{(1 + \#\ \text{documents containing}\ w)}{(1 + \#\ \text{documents containing}\ w) + (1 + \#\ \text{documents NOT containing}\ w)}$$

Feature = One Bit Per Word Token

- Features:
  - F_{jw} = 1 if the j'th word token is word "w"
- Parameters:
  - $\lambda_{cw} \equiv P\left(F_{jw} = 1 \middle| C = c\right) = P(W_j = w \,|C = c)$
  - Note this means that $P\left(F_{jw} = 0 \middle| C = c\right) = \sum_{v \neq w} \lambda_{cv}$
- Parameter Learning:

$$\lambda_{cw} = \frac{(1 + \#\text{ tokens of } w \text{ in the training database})}{\sum_{v=1}^{V}(1 + \#\text{ tokens of } v \text{ in the training database})}$$

Feature = One Bit Per Document

Classification:

C* = argmax P(C=c|document)

= argmax P(C=c) P(Document|C=c)

$$= \arg\max_{c} \left(\pi_c \prod_{w:\, f_{cw}=1} \lambda_{cw} \prod_{w:\, f_{cw}=0} (1 - \lambda_{cw}) \right)$$

P(C=c) * prod_{words that occurred}P(word ooccus|C=c) * prod_{didn’t occur} P(didn’t occur|C=c)

Feature = One Bit Per Word Token

Classification:

C* = argmax P(C=c|document)

= argmax P(C=c) P(Document|C=c)

$$= \arg\max_{c} \left(\pi_c \prod_{j=1}^{n} \lambda_{cw_j} \right)$$

P(C=c) prod_{words in the document} P(get that particular word | C=c)

Feature = One Bit Per Document

Classification:

$$C^* = \arg\max_c \left(\pi_c \prod_{w=1}^{V} \left(\frac{\lambda_{cw}}{1-\lambda_{cw}} \right)^{f_{cw}} (1-\lambda_{cw}) \right)$$

$$C^* = \arg\max_c \left(\boxed{\beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw}} \right)$$

In a 2-dimensional feature space (f_{c1}, f_{c2}), this is the equation for a line, with intercept $-\beta_c$, and with slope given by α_{c1}/α_{c2}

$$\alpha_{cw} = \log\left(\frac{\lambda_{cw}}{1-\lambda_{cw}} \right), \qquad \beta_c = \log\left(\pi_c \prod_{w=1}^{V} (1-\lambda_{cw}) \right)$$

Feature = One Bit Per Word Token

Classification:

$$C^* = \arg\max_c \left(\pi_c \prod_{w=1}^{V} {\lambda_{cw}}^{s_w} \right)$$

Where $s_w =$ number of times w occurred in the document!! So…

$$C^* = \arg\max_c \left(\boxed{\beta_c + \sum_{w=1}^{V} \alpha_{cw} s_{cw}} \right)$$

In a 2-dimensional feature space (f_{c1}, f_{c2}), this is the equation for a line, with intercept $-\beta_c$, and with slope given by α_{c1}/α_{c2}

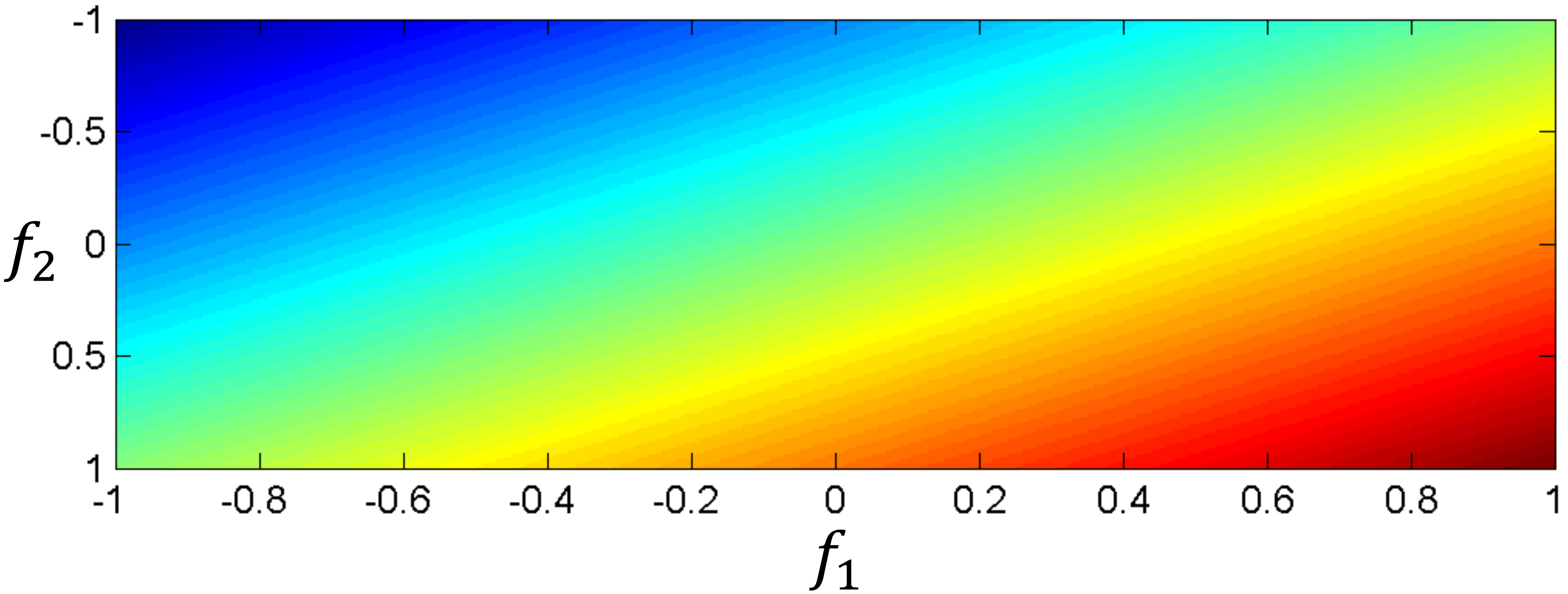
$$\alpha_{cw} = \log \lambda_{cw}, \qquad \beta_c = \log \pi_c$$

Linear Classifiers

- Naïve Bayes/BoW classifiers
- Linear Classifiers in General
- Perceptron
- Differential Perceptron/Neural Net

Linear Classifiers in General

The function $\beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw}$ is an affine function of the features f_{cw}. That means that its contours are all straight lines. Here is an example of such a function, plotted as variations of color in a two-dimensional space f_1 by f_2:



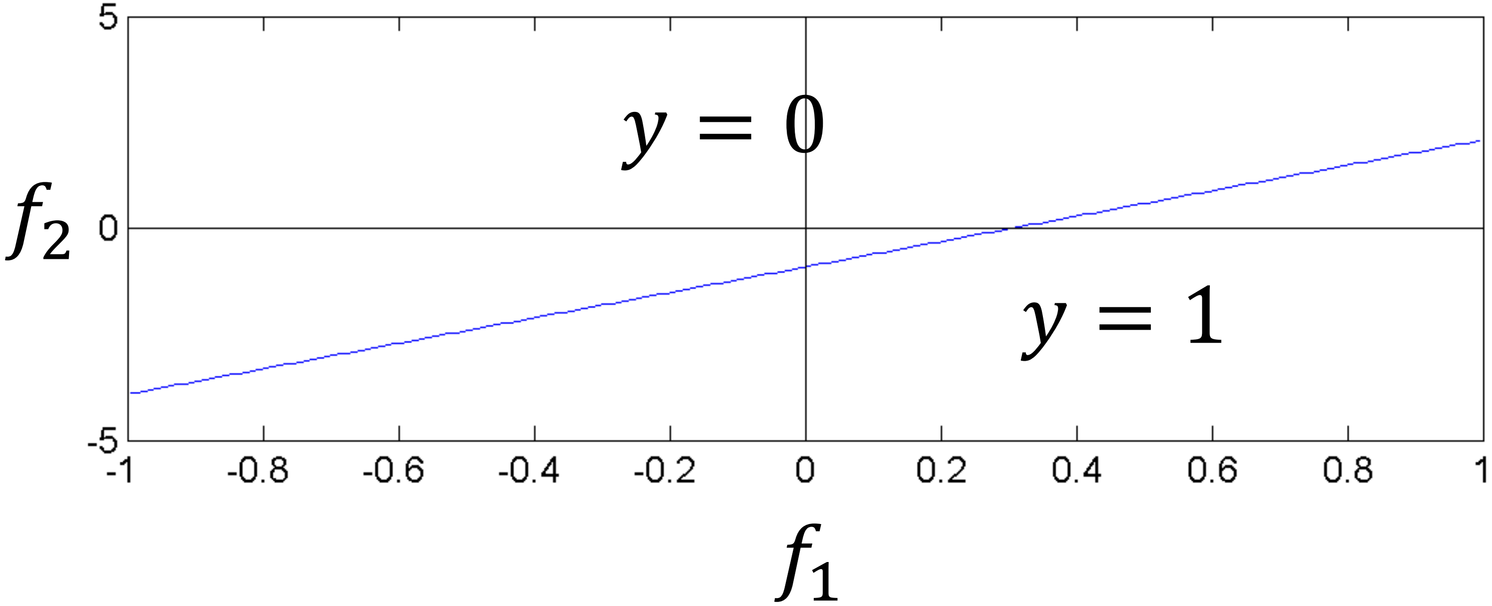
Linear Classifiers in General

Consider the classifier

$$y = 1 \quad \text{if} \quad \beta_c + \textstyle\sum_{w=1}^{V} \alpha_{cw} f_{cw} > 0$$

$$y = 0 \quad \text{if} \quad \beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw} < 0$$

This is called a “linear classifier” because the boundary between the two classes is a line.
Here is an example of such a classifier, with its boundary plotted as a line in the two-dimensional space f_1 by f_2:

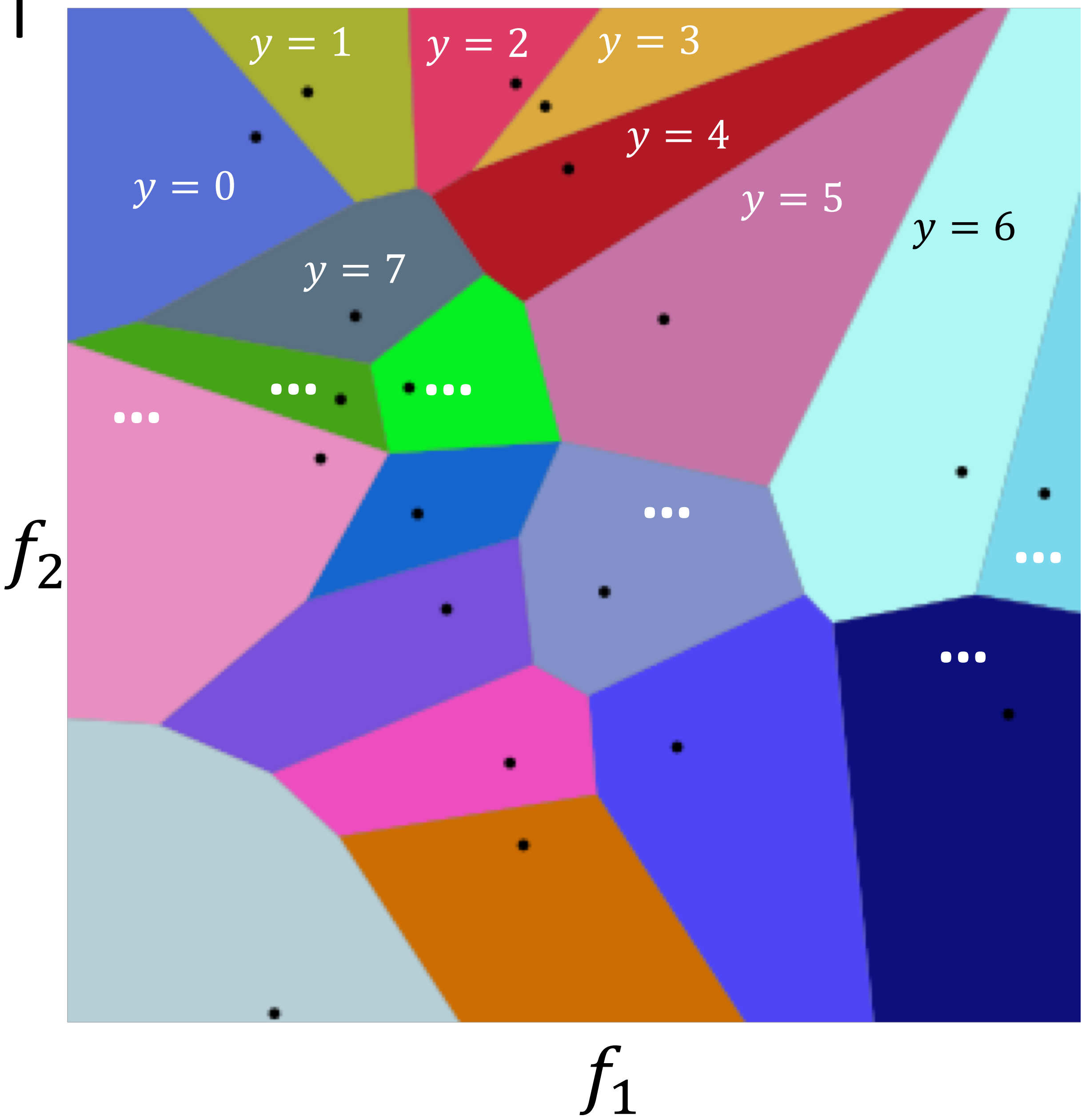


Linear Classifiers in General

Consider the classifier

$$y = \arg\max_c \left(\beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw} \right)$$

- This is called a “multi-class linear classifier.”
- The regions $y = 0, y = 1, y = 2$ etc. are called “Voronoi regions.”
- They are regions with piece-wise linear boundaries. Here is an example from Wikipedia of Voronoi regions plotted in the two-dimensional space f_1 by f_2:



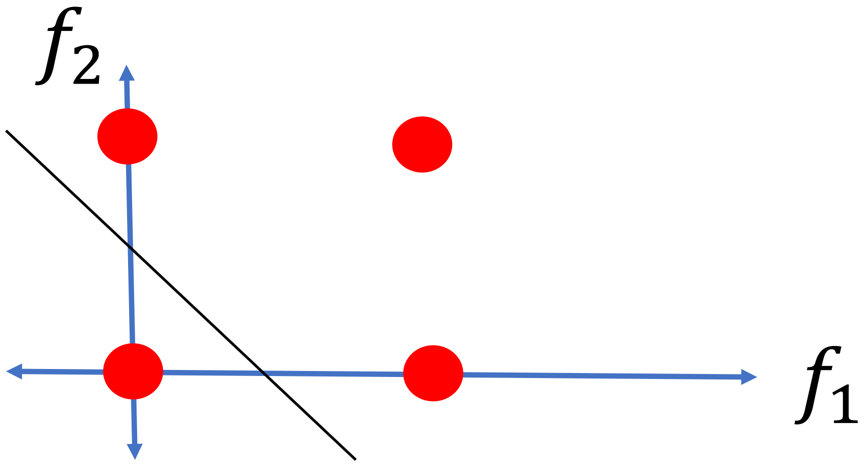
Linear Classifiers in General

When the features are binary ($f_w \in \{0,1\}$), many (but not all!) binary functions can be re-written as linear functions. For example, the function

$$y = (f_1 \vee f_2)$$

can be re-written as

y=1 iff $f_1 + f_2 - 0.5 > 0$

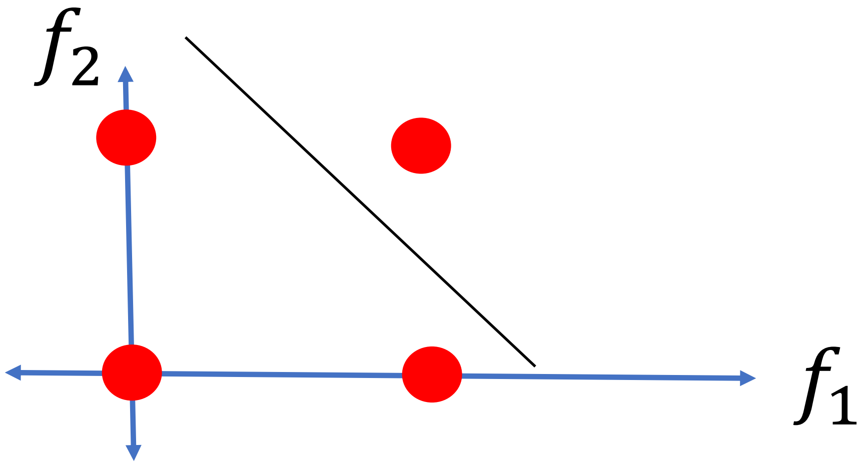


Similarly, the function

$$y = (f_1 \wedge f_2)$$

can be re-written as

y=1 iff $f_1 + f_2 - 1.5 > 0$



Linear Classifiers in General

- Not all logical functions can be written as linear classifiers!
- Minsky and Papert wrote a book called *Perceptrons* in 1969. Although the book said many other things, the only thing most people remembered about the book was that:
- **"A linear classifier cannot learn an XOR function."**
- Because of that statement, most people gave up working on neural networks from about 1969 to about 2006.
- Minsky and Papert also proved that a two-layer neural net can learn an XOR function. But most people didn't notice.

Linear Classifiers

Classification:

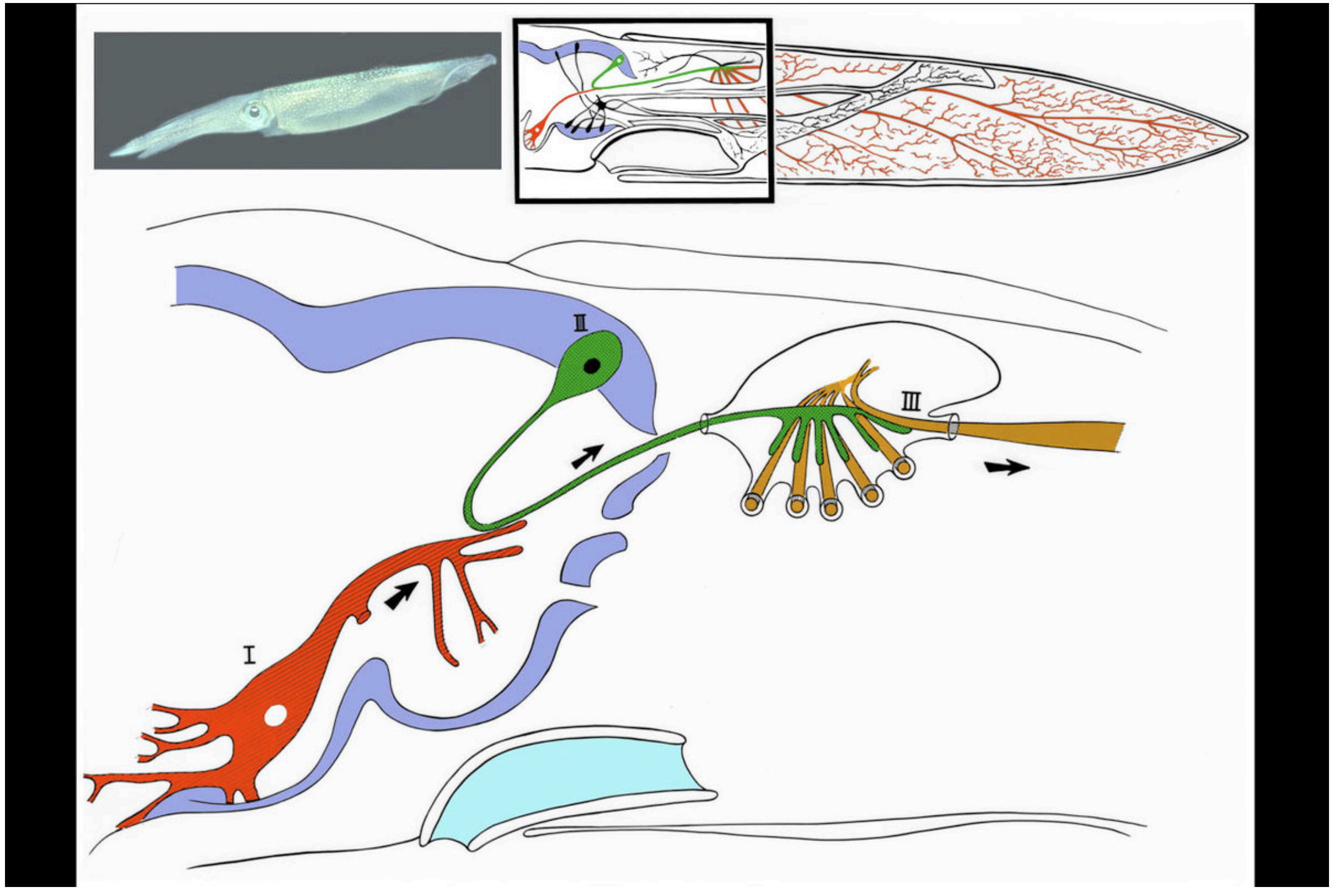
$$y = \arg\max_{c}\left(\beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw}\right)$$

- Where f_{cw} are the features (binary, integer, or real), α_{cw} are the feature weights, and β_c is the offset

Linear Classifiers

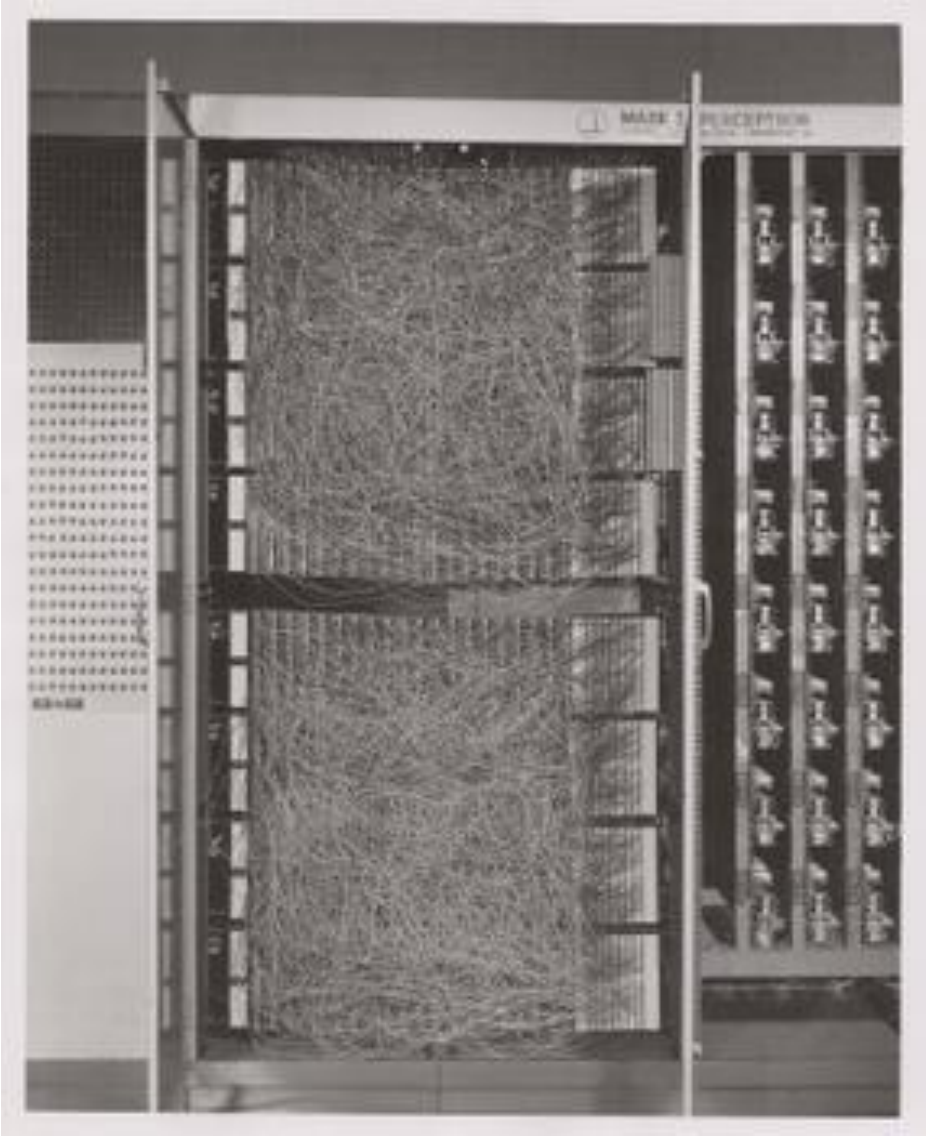
- Naïve Bayes/BoW classifiers
- Linear Classifiers in General
- Perceptron
- Differential Perceptron/Neural Net

The Giant Squid Axon



- 1909: Williams discovers that the giant squid has a giant neuron (axon 1mm thick)
- 1939: Young finds a giant synapse (fig. shown: Llinás, 1999, via Wikipedia). Hodgkin & Huxley put in voltage clamps.
- 1952: Hodgkin & Huxley publish an electrical current model for the generation of binary action potentials from real-valued inputs.

Perceptron

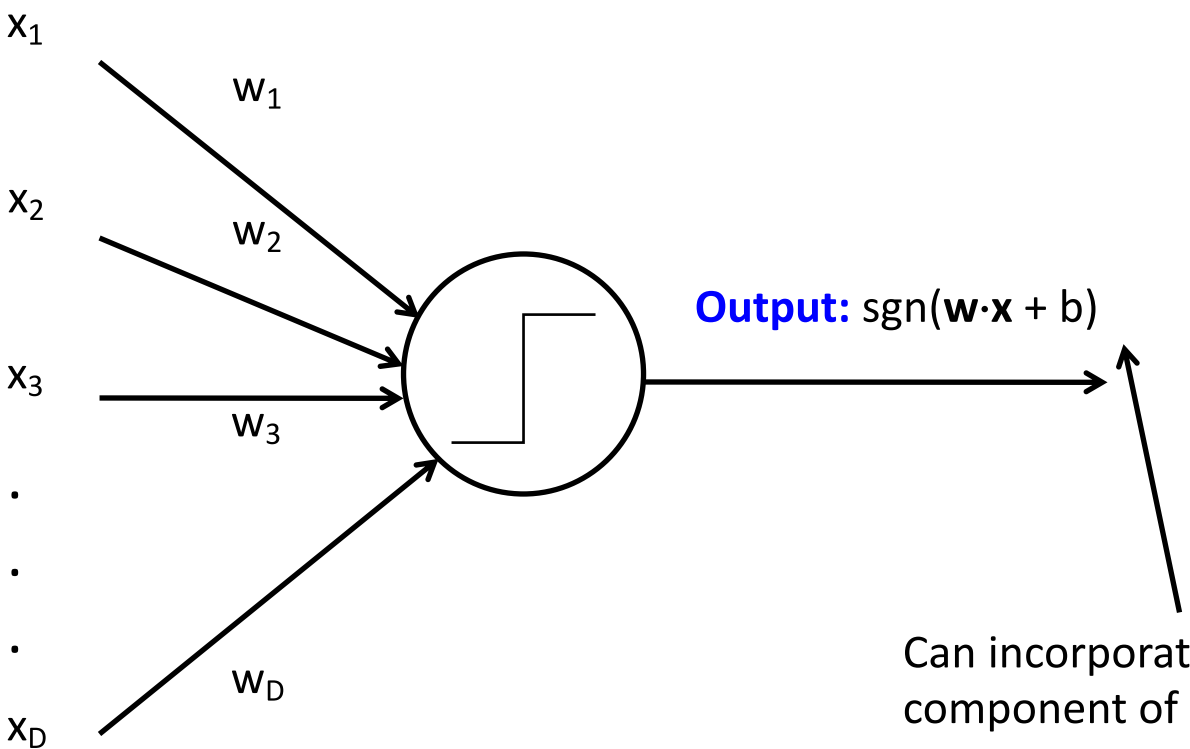


- 1959: Rosenblatt is granted a patent for the “perceptron,” an electrical circuit model of a neuron.

Perceptron

Input

Weights



Output: sgn($\mathbf{w} \cdot \mathbf{x}$ + b)

Can incorporate bias as component of the weight vector by always including a feature with value set to 1

Perceptron model: action potential = signum(affine function of the features)

$$y = \text{sgn}(\alpha_1 f_1 + \alpha_2 f_2 + \ldots + \alpha_V f_V + \beta) = \text{sgn}(\vec{w}^T \vec{f})$$

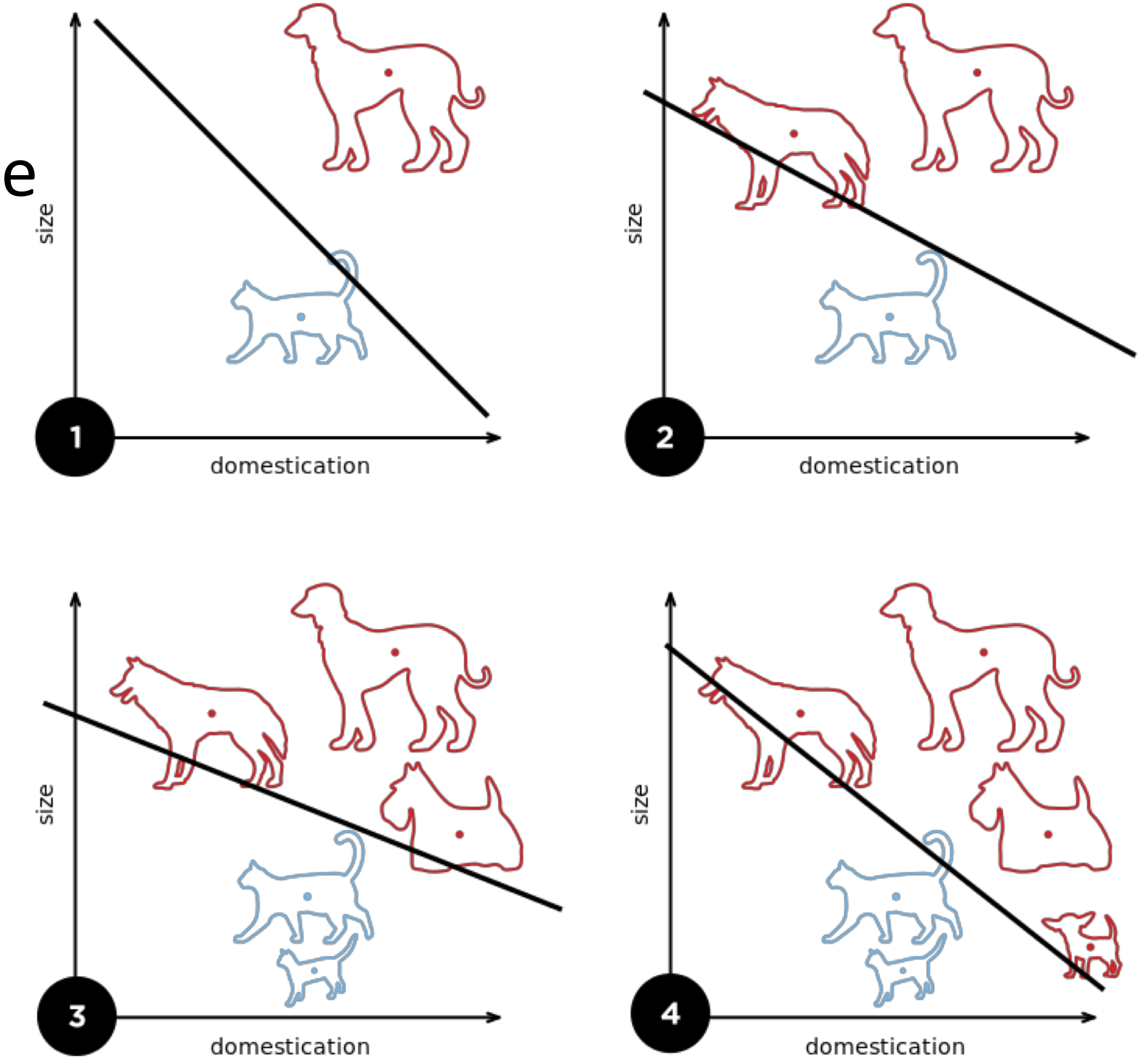
Where $\vec{w} = [\alpha_1, \ldots, \alpha_V, \beta]^T$

and $\vec{f} = [f_1, \ldots, f_V, 1]^T$

Perceptron

Rosenblatt’s big innovation: the perceptron learns from examples.

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training example:
  - If classified correctly, do nothing
  - If classified incorrectly, update weights



Perceptron

For each training instance $\vec{\boldsymbol{f}}$ with label $y \in \{-1,1\}$:

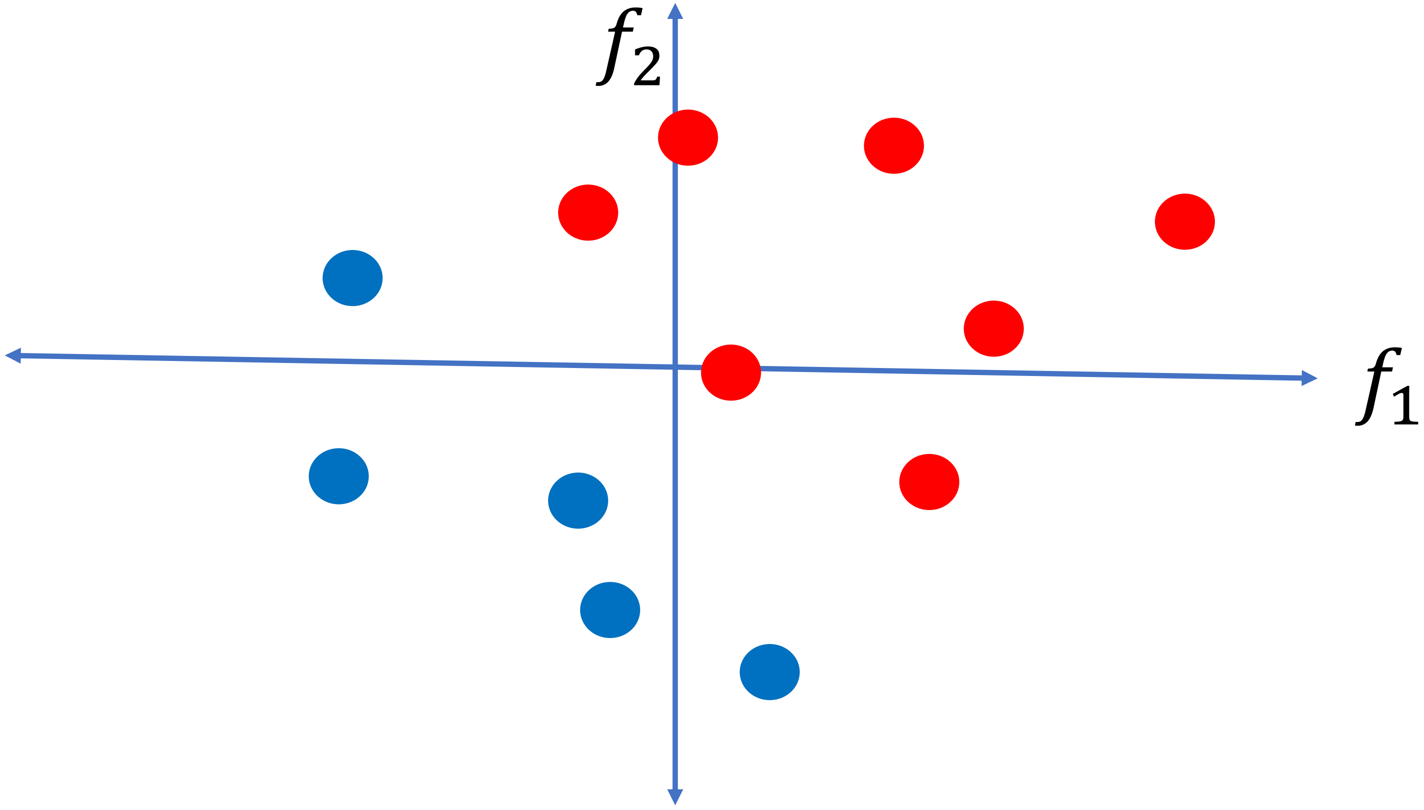
- Classify with current weights: $y' = \operatorname{sgn}(\vec{w}^T \vec{f})$
  - Notice $y' \in \{-1,1\}$ too.
- Update weights:
  - if $y = y'$ then do nothing
  - if $y \neq y'$ then $\vec{w} = \vec{w} + \eta\, y\, \vec{f}$
  - η (eta) is a “learning rate.” More about that later.

Perceptron: Proof of Convergence

- If the data are linearly separable (if there exists a $\vec{w}$ vector such that the true label is given by $y' = \text{sgn}(\vec{w}^T\vec{f})$), then the perceptron algorithm is guarantee to converge, even with a constant learning rate, even η=1.
- In fact, training a perceptron is often the fastest way to find out if the data are linearly separable. If $\vec{w}$ converges, then the data are separable; if $\vec{w}$ diverges toward infinity, then no.
- If the data are not linearly separable, then perceptron converges iff the learning rate decreases, e.g., η=1/n for the n'th training sample.

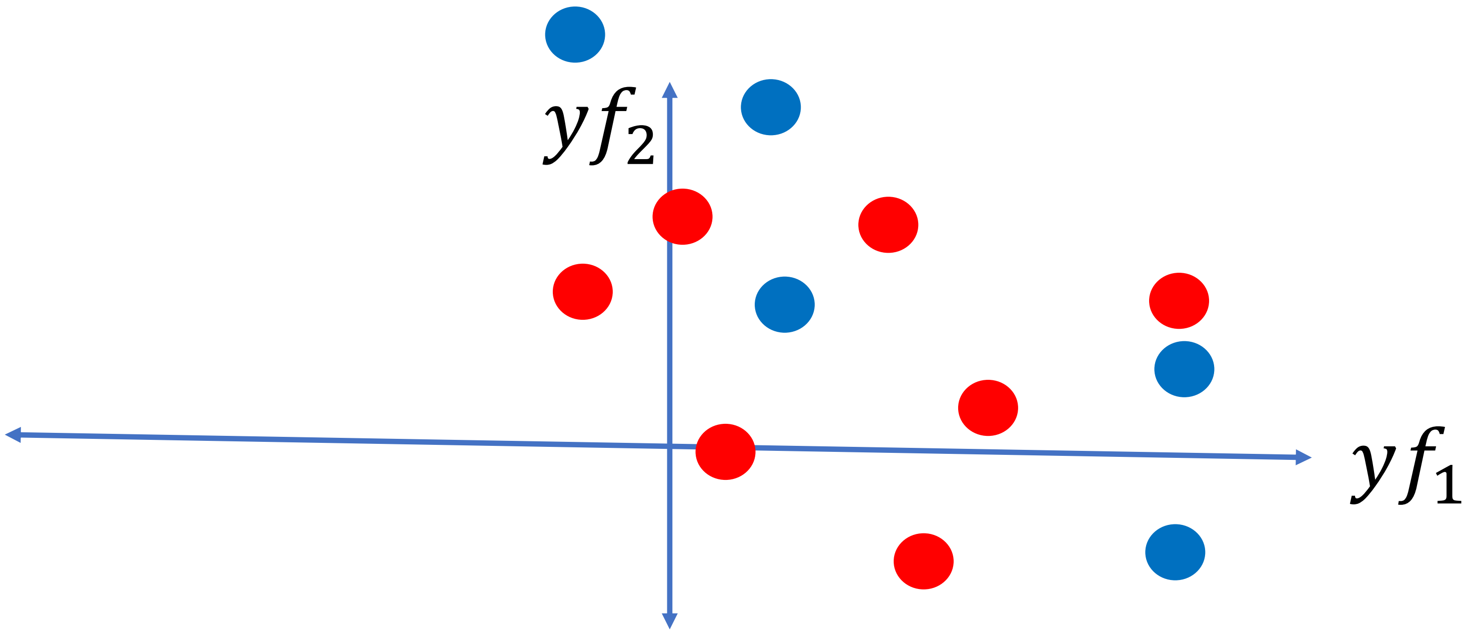
Perceptron: Proof of Convergence

Suppose the data are linearly separable. For example, suppose red dots are the class y=1, and blue dots are the class y=-1:



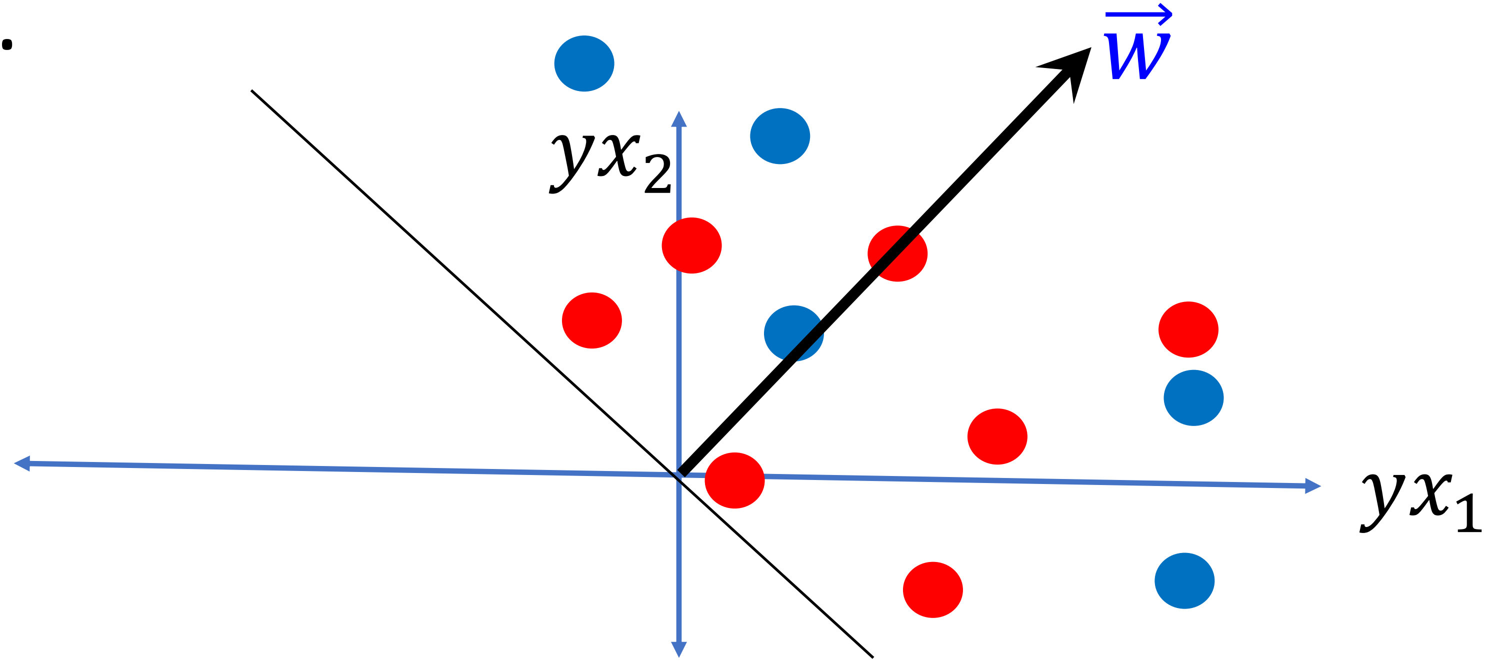
Perceptron: Proof of Convergence

- Instead of plotting $\vec{f}$, plot $y\times\vec{f}$. The red dots are unchanged; the blue dots are multiplied by -1.
- Since the original data were linearly separable, the new data are all in the same half of the feature space.



Perceptron: Proof of Convergence

- Remember the perceptron training rule: if any example is misclassified, then we use it to update $\vec{w} = \vec{w} + y\,\vec{f}$.
- So eventually, $\vec{w}$ becomes just a weighted average of $y\vec{f}$.
- … and the perpendicular line, $\vec{w}^T\vec{f} = 0$, is the classifier boundary.



Perceptron: Proof of Convergence: Conclusion

- If the data are linearly separable, then the perceptron will eventually find the equation for a line that separates them.
- If the data are NOT linearly separable, then perceptron converges iff the learning rate decreases, e.g., $\eta=1/n$ for the n'th training sample. …. In this case, convergence is trivially obvious, because y and $\vec{f}$ are finite, therefore the weight updates $\eta\, y\, \vec{f}$ approach 0 as η approaches 0.

Implementation details

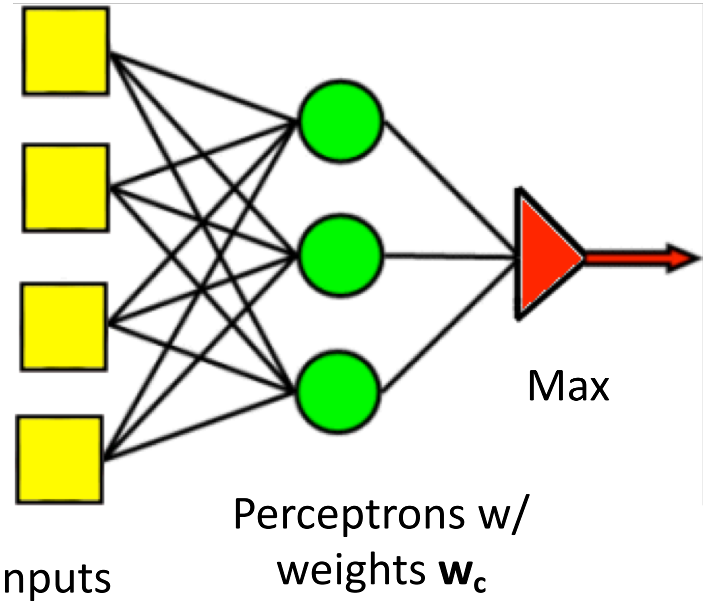
- Bias (add feature dimension with value fixed to 1) vs. no bias
- Initialization of weights: all zeros vs. random
- Learning rate decay function
- Number of epochs (passes through the training data)
- Order of cycling through training examples (random)

Multi-class perceptrons

- *One-vs-others* framework: Need to keep a weight vector $\mathbf{w}_c$ for each class c
- Decision rule: $y = \text{argmax}_c\ \mathbf{w}_c \cdot \mathbf{f}$
- Update rule: suppose example from class c gets misclassified as c’
    - Update for c: $\mathbf{w}_c \leftarrow \mathbf{w}_c + \eta \mathbf{f}$
    - Update for c’: $\mathbf{w}_{c'} \leftarrow \mathbf{w}_{c'} - \eta \mathbf{f}$
    - Update for all classes other than c and c’: no change

Review: Multi-class perceptrons

- *One-vs-others* framework: Need to keep a weight vector $\mathbf{w}_c$ for each class c
- Decision rule: $y = \text{argmax}_c\ \mathbf{w}_c \cdot \mathbf{f}$



Linear Classifiers

- Naïve Bayes/BoW classifiers
- Linear Classifiers in General
- Perceptron
- Differential Perceptron/Neural Net

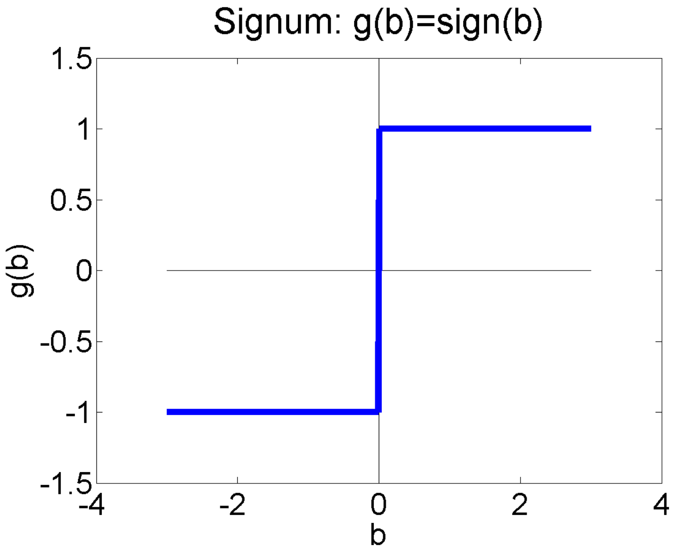
Differentiable Perceptron

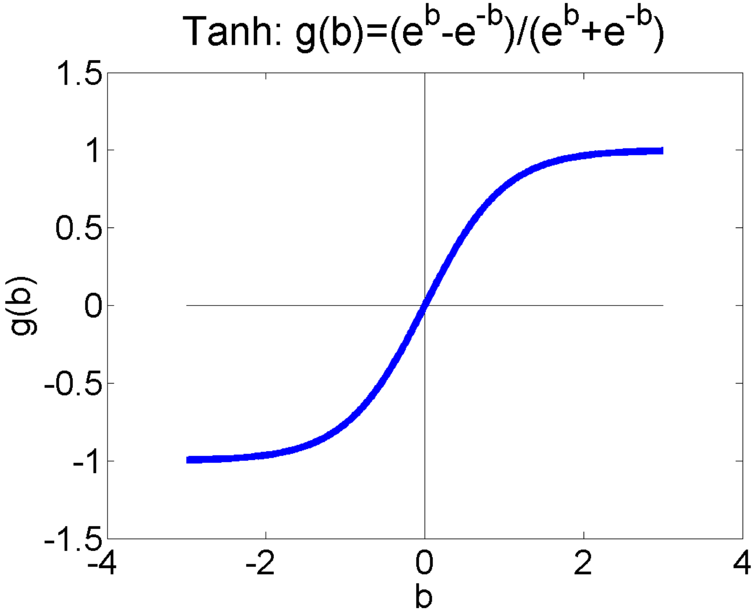
- Also known as a "one-layer feedforward neural network," also known as "logistic regression." Has been re-invented many times by many different people.
- Basic idea: replace the non-differentiable decision function

$$y' = \text{sign}(\vec{w}^T \vec{f})$$

with a differentiable decision function

$$y' = \tanh(\vec{w}^T \vec{f})$$


Signum: g(b)=sign(b)


Tanh: $g(b)=(e^{b}-e^{-b})/(e^{b}+e^{-b})$

Differentiable Perceptron

- Suppose we have n training vectors, $\vec{f}_1$ through $\vec{f}_n$. Each one has an associated label $y_i \in \{-1,1\}$. Then we replace the true loss function,

$$L(y_1, \ldots, y_n, \vec{f}_1, \ldots, \vec{f}_n) = \sum_{i=1}^{n} \left(y_i - \operatorname{sign}(\vec{w}^T \vec{f}_i)\right)^2$$

with a differentiable error

$$L(y_1, \ldots, y_n, \vec{f}_1, \ldots, \vec{f}_n) = \sum_{i=1}^{n} \left(y_i - \tanh(\vec{w}^T \vec{f}_i)\right)^2$$

Why Differentiable?

- Why do we want a differentiable loss function?

$$L(y_1, \ldots, y_n, \vec{f}_1, \ldots, \vec{f}_n) = \sum_{i=1}^{n} \left(y_i - \tanh(\vec{w}^T \vec{f}_i)\right)^2$$

- Answer: because if we want to improve it, we can adjust the weight vector in order to reduce the error:

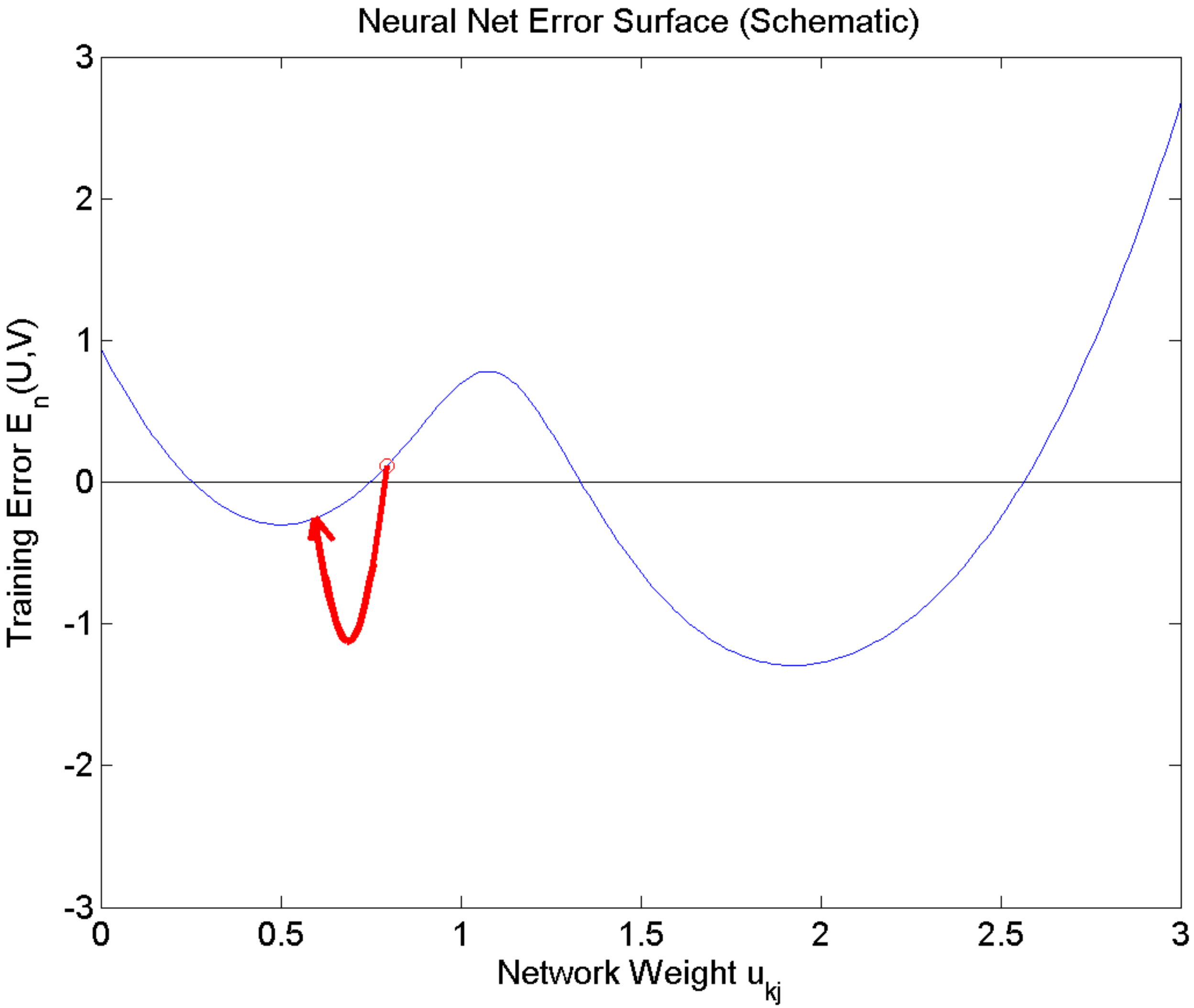
$$\vec{w} = \vec{w} - \eta \nabla_{\vec{w}} L$$

- This is called "gradient descent." We move $\vec{w}$ "downhill," i.e., in the direction that reduces the value of the loss L.

Differential Perceptron

The weights get updated according to

$$\vec{w} = \vec{w} - \eta \nabla_{\vec{w}} L$$

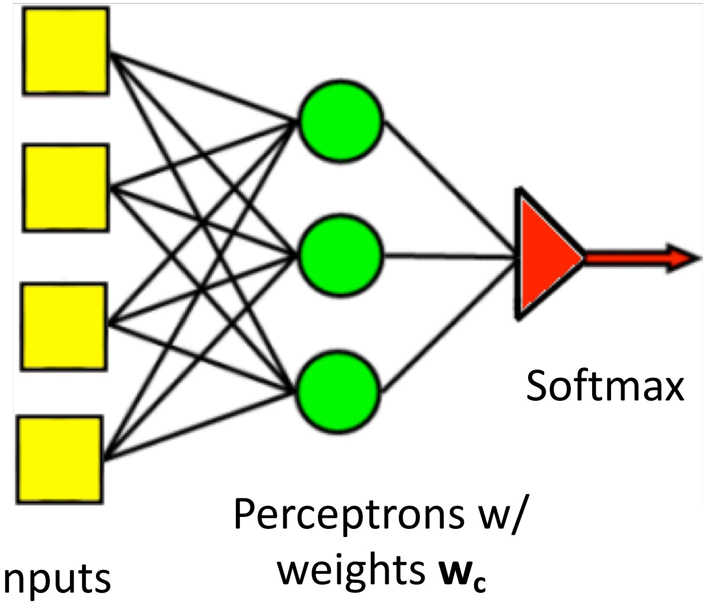


Differentiable Multi-class perceptrons

Same idea works for multi-class perceptrons. We replace the non-differentiable decision rule c = argmax$_c$ **w**$_c$· **f** with the differentiable decision rule c = softmax$_c$ **w**$_c$· **f**, where the softmax function is defined as

Softmax:

$$p(c|\vec{f}) = \frac{e^{\vec{w}_c \cdot \vec{f}}}{\sum_{k=1}^{\#\ classes} e^{\vec{w}_k \cdot \vec{f}}}$$



Differentiable Multi-Class Perceptron

- Then we can define the loss to be:

$$L(y_1, \dots, y_n, \vec{f}_1, \dots, \vec{f}_n) = -\sum_{i=1}^{n} \ln p(c = y_i | \vec{f}_i)$$

- And because the probability term on the inside is differentiable, we can reduce the loss using gradient descent:

$$\vec{w} = \vec{w} - \eta \nabla_{\vec{w}} L$$

Summary

You now know SEVEN!! different types of linear classifiers. These 5 types are things you should completely understand already now:

- One bit per document Naïve Bayes
- One bit per word token Naïve Bayes
- Linear classifier can implement some logical functions, like AND and OR, but not others, like XOR
- Perceptron
- Multi-class Perceptron

These 2 types of linear classifiers have been introduced today, and you should know the general idea, but you don't need to understand the equations yet. We will spend lots more time talking about those equations later in the semester.

- Differentiable Perceptron a.k.a. Logistic Regression
- Differentiable Multi-class perceptron