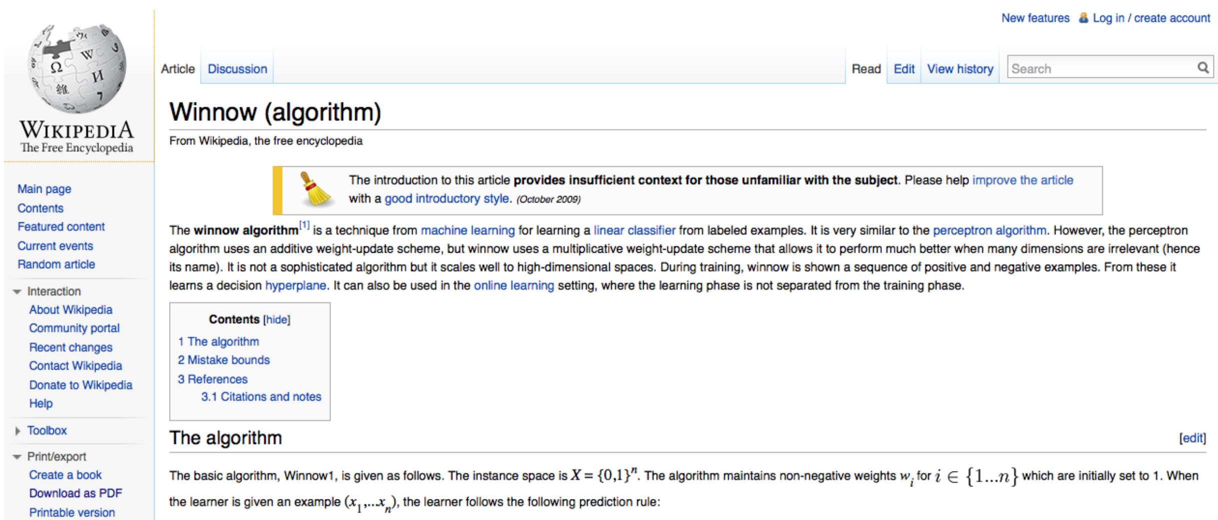
Attribution

- These slides were prepared for the New Jersey Governor’s School course “The Math Behind the Machine” taught in the summer of 2012 by Grant Schoenebeck
- Large parts of these slides were copied or modified from the previous years’ courses given by Troy Lee in 2010.

Learning


New features Log in / create account

Article Discussion Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article

Interaction
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Donate to Wikipedia
Help

Toolbox

Print/export
Create a book
Download as PDF
Printable version

Winnow (algorithm)

From Wikipedia, the free encyclopedia

The introduction to this article **provides insufficient context for those unfamiliar with the subject**. Please help improve the article with a good introductory style. *(October 2009)*

The **winnow algorithm**[1] is a technique from machine learning for learning a linear classifier from labeled examples. It is very similar to the perceptron algorithm. However, the perceptron algorithm uses an additive weight-update scheme, but winnow uses a multiplicative weight-update scheme that allows it to perform much better when many dimensions are irrelevant (hence its name). It is not a sophisticated algorithm but it scales well to high-dimensional spaces. During training, winnow is shown a sequence of positive and negative examples. From these it learns a decision hyperplane. It can also be used in the online learning setting, where the learning phase is not separated from the training phase.

Contents [hide]

The algorithm [edit]

The basic algorithm, Winnow1, is given as follows. The instance space is $X = \{0,1\}^n$. The algorithm maintains non-negative weights w_i for $i \in \{1 \ldots n\}$ which are initially set to 1. When the learner is given an example $(x_1, \ldots x_n)$, the learner follows the following prediction rule:

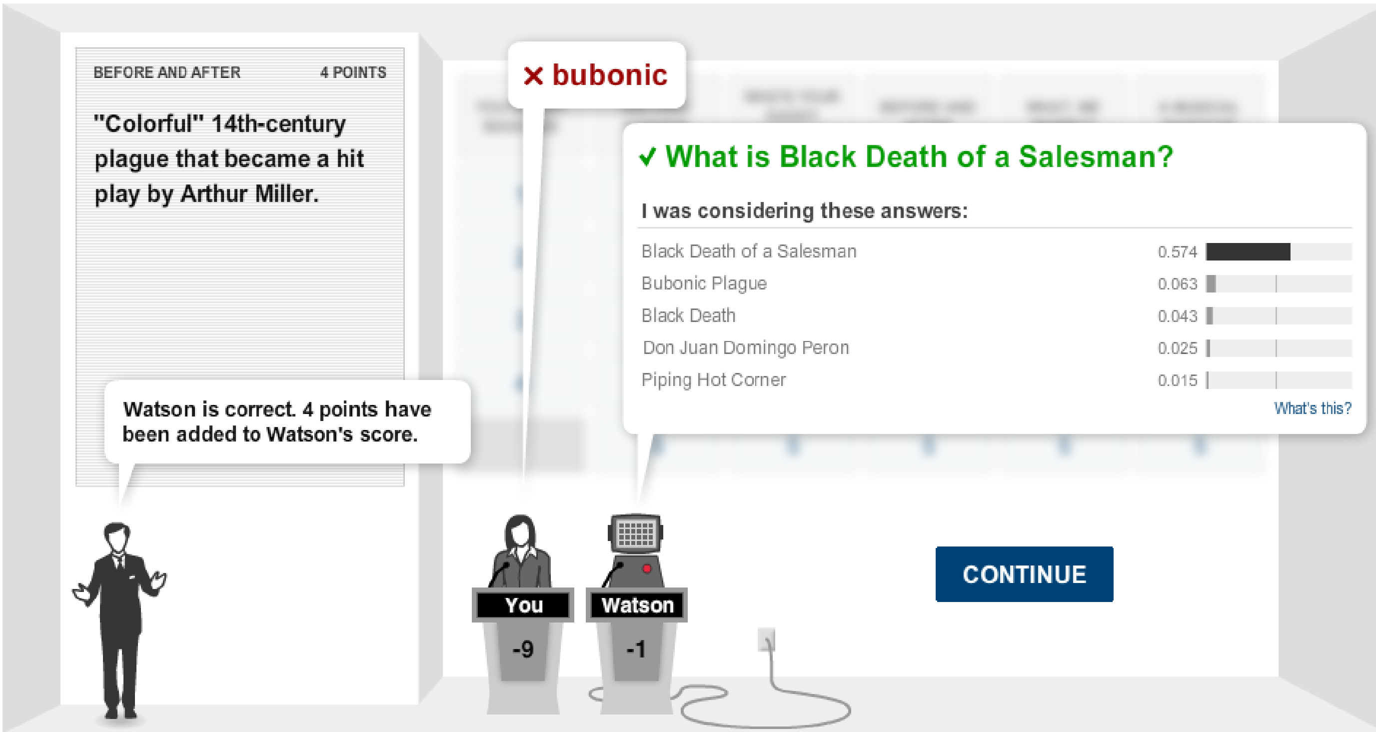
Is our computers learning?

- What does it mean for a computer to “learn”?
  - Prediction with low error
  - Compression
- Cannot “learn” random functions!
  - Unpredictable
  - Uncompressable
- Must assume function “nice” or bias toward certain functions
- Many different scenarios
  - Unsupervised learning
    - Clustering
  - Supervised learning
  - Semi-supervised Learning
  - Learning and Optimizing

Examples of Computational Learning

- Spam
- Deep Blue
- Search Engines
- Datacenter Power conservation
- Image Search
- Face recognition in Facebook

Answering Questions

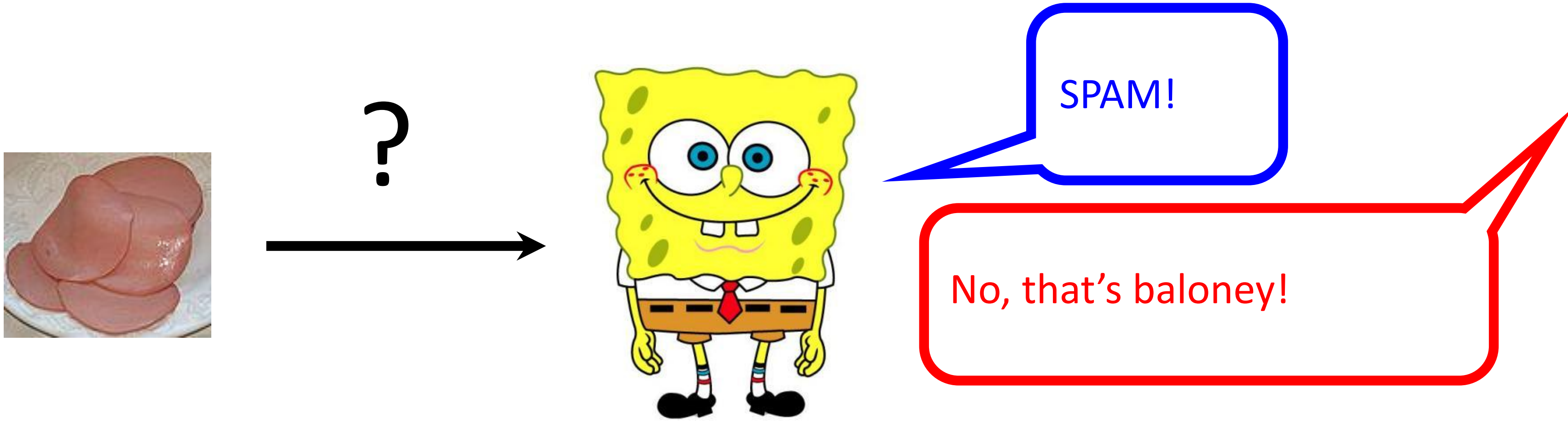


http://www.nytimes.com/2010/06/20/magazine/20Computer-t.html

Prediction

- Predict a user's rating based on previous rankings and rankings of other users.
- Recently, Netflix gave out $1,000,000 in contest to improve their ranking system by 10%.
  - Winning algorithm was a blend of many methods.
  - Surprising relevant information:
    - Ranking a movie just after watching vs. years later...

Online Learning

- Bob is given examples on the fly, and has to classify them.
- 

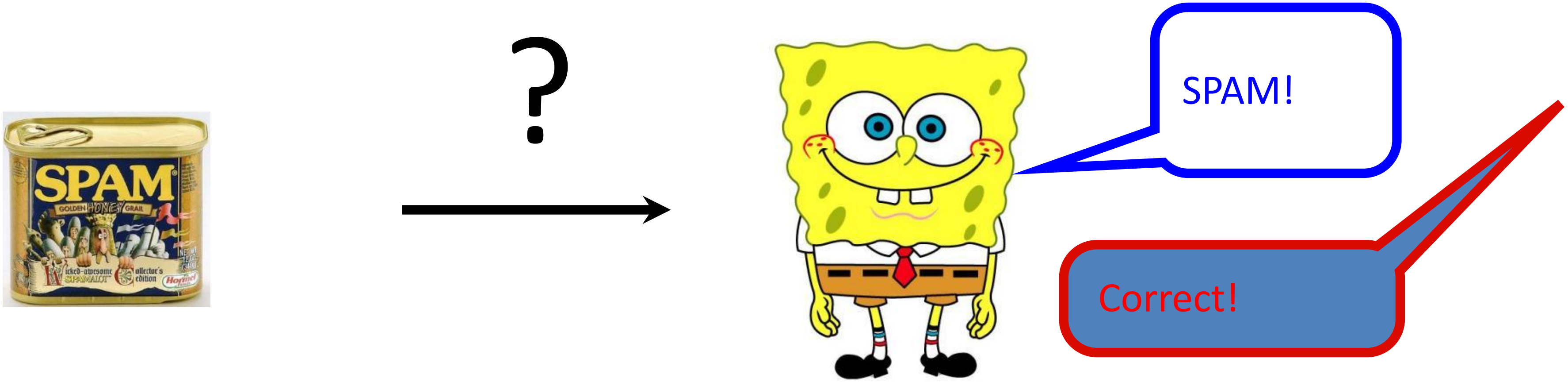
- After each example, Bob is given correct answer.

Online Learning

- Focus on the binary case: spam vs. not spam

SPAM

~~SPAM~~



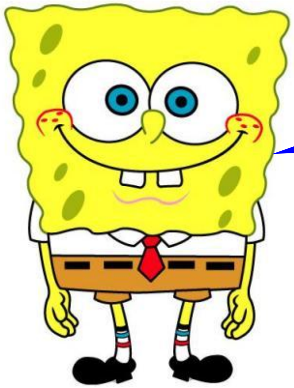
Online Learning

Focus on the binary case:
spam vs. not spam

SPAM

~~SPAM~~

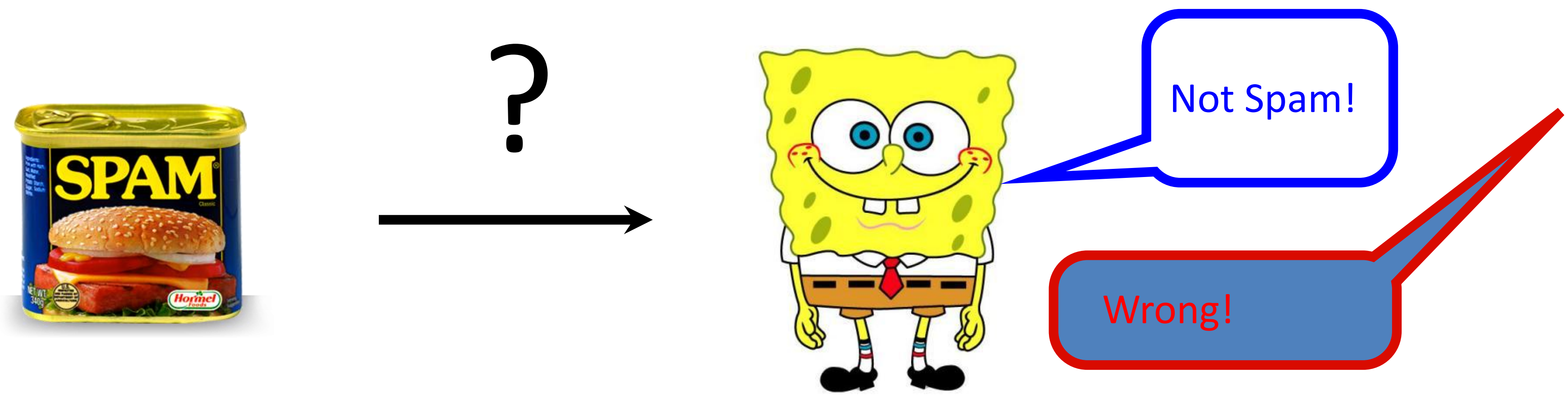
?



Not Spam!

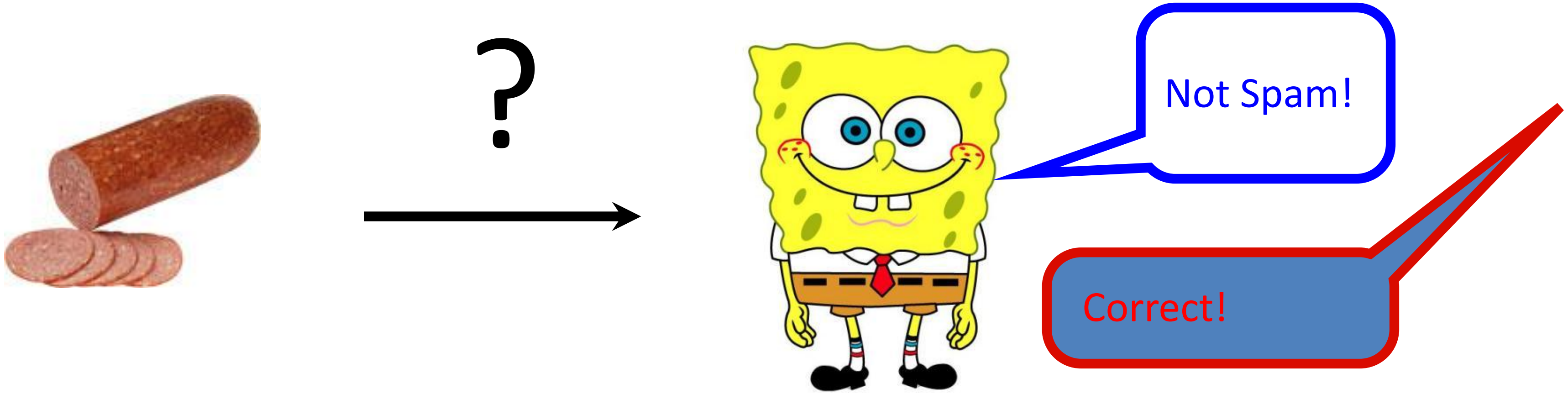
Correct!

Online Learning



Online Learning

Interested in Total Number of Mistakes



More abstractly...

Bob is learning an unknown Boolean function u(x).

– Here, u is a function on lunch meats.

– Usually, have an assumption that u comes from a known family of functions---called a concept class.

→ 0

→ 1

→ 0

Learning Disjunctions

- We will assume the unknown function u comes from a simple family: monotone disjunctions.
  - $u(x) = x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_k}$
  - Monotone means no negations

Disjunctions

x_1 = [Aardvark]

x_2 = [Aarrghh]

$\vdots$

$x_{601,384}$ =[Zyzzyva]

A simple “bag of words” model of spam---

An email is spam if it contains one of a handful of keywords: viagra, lottery, xanax...

SPAM(x)=viagra(x) OR lottery(x) OR ... OR xanax(x)

We have variables to indicate if a word is present in email x.

n=Total # variables, k=number present in disjunction.

Basic Algorithm

Maintain a hypothesis h(x) for what the disjunction is.

Initially, $$h(x) = x_1 \vee x_2 \vee \cdots \vee x_n$$

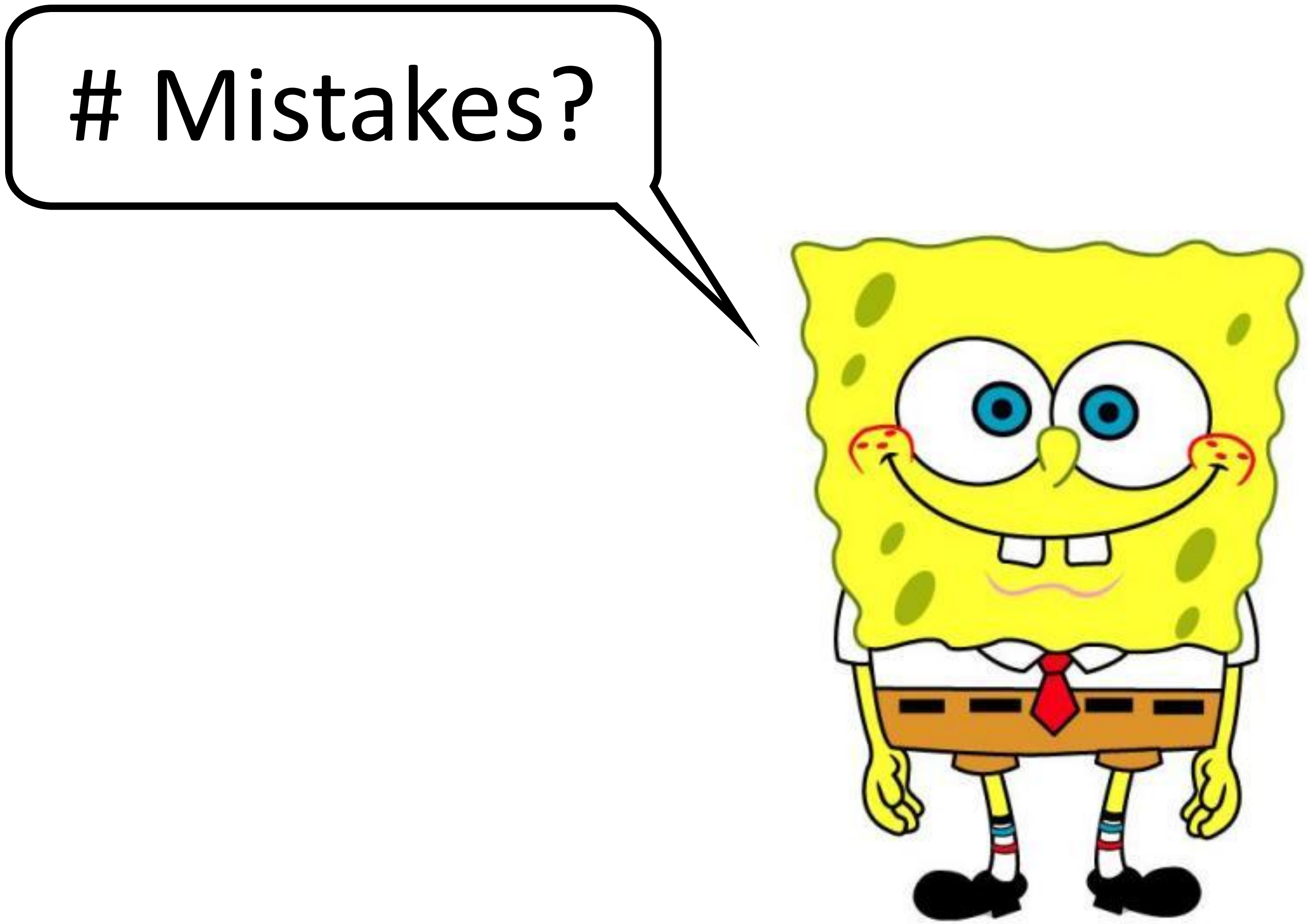
That is, we assume disjunction includes all variables.

This will label everything as spam!

Update Rule

Every time we get an answer wrong, we will modify our hypothesis, as follows:

- On false positive, i.e. SPAM(x)=0 and h(x)=1, remove all variables from h where $x_i = 1$.
- On false negative, i.e. SPAM(x)=1 and h(x)=0, output FAIL.
  - If the function we are learning is actually a monotone disjunction, false negatives never happen.
  - Variables present in hypothesis always a superset of those in SPAM function.


Mistakes?

Number of Mistakes

Claim: This algorithm makes at most n mistakes if unknown function is indeed a disjunction.

Every time we get an answer wrong, we will modify our hypothesis, as follows:

- On false positive, i.e. SPAM(x)=0 and h(x)=1, remove all variables from h where $x_i = 1$.
- On false negative, i.e. SPAM(x)=1 and h(x)=0, output FAIL.
  - If the function we are learning is actually a monotone disjunction, false negatives never happen.
  - Variables present in hypothesis always a superset of those in SPAM function.

- When we make a mistake at least one word is eliminated from hypothesis

Importance of Good Teachers...

- What happens if some example is mislabeled?
  - $u(x)=1$ but Teacher mistakenly says $u(x)=0$.
  - The weight for some variable relevant to u will be set to zero, and can never come back!
- Then we can make an unbounded number of mistakes.

Sparse Disjunctions

- This algorithm has disadvantage that the number of mistakes can depend on total number of variables.
  - In spam example, even though there are only a few keywords, we could make make mistakes on order the number of words in English!
- Ideally, would like an algorithm that depends on the number of relevant variables rather than total number of variables.

Winnow Algorithm

- Winnow- to separate the chaff from the grain
- The winnow algorithm quickly separates the irrelevant variables from the relevant ones.

Winnow Algorithm

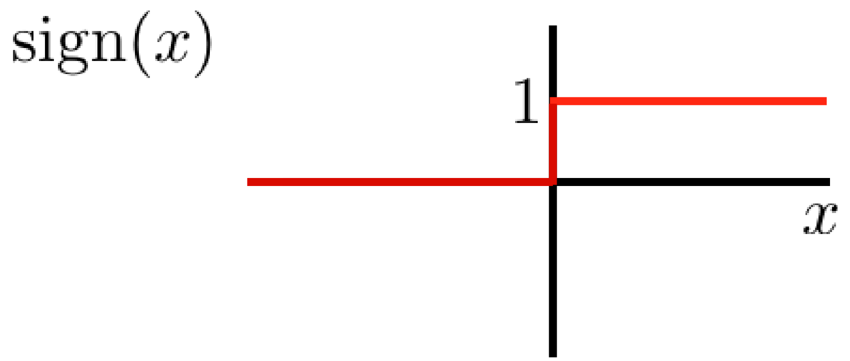
- Say the unknown function only depends on k variables out of a total of n.
  - $u(x) = x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_k}$
  - The winnow algorithm has a mistake bound of about $k \log(n)$
  - Much improved if $k \ll n$

Linear Threshold Function

- In the winnow algorithm, Bob maintains as hypothesis a linear threshold function.

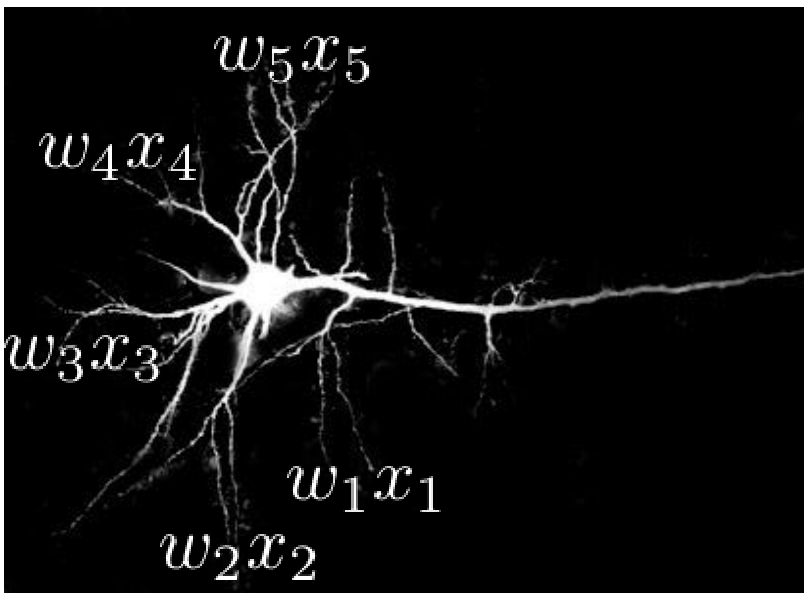
$$h(x) = \mathrm{sign}(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n - \theta).$$

- Weights w_i and threshold θ are real numbers.



Linear Threshold Function

- Linear threshold functions are used as a simple model of a neuron.



$$\sum_i w_i x_i \geq \theta?$$

Linear Threshold Function

$$h(x) = \mathrm{sign}(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n - \theta).$$

What can linear threshold functions do?

$$\mathrm{OR}(x) = \mathrm{sign}(x_1 + x_2 + \cdots + x_n - 1/2).$$

$$\mathrm{AND}(x) = \mathrm{sign}(x_1 + x_2 + \cdots + x_n - (n - 1/2)).$$

$$\mathrm{MAJ}(x) = \mathrm{sign}(x_1 + x_2 + \cdots + x_n - n/2)$$

Linear Threshold Function

Another sanity check: the unknown function

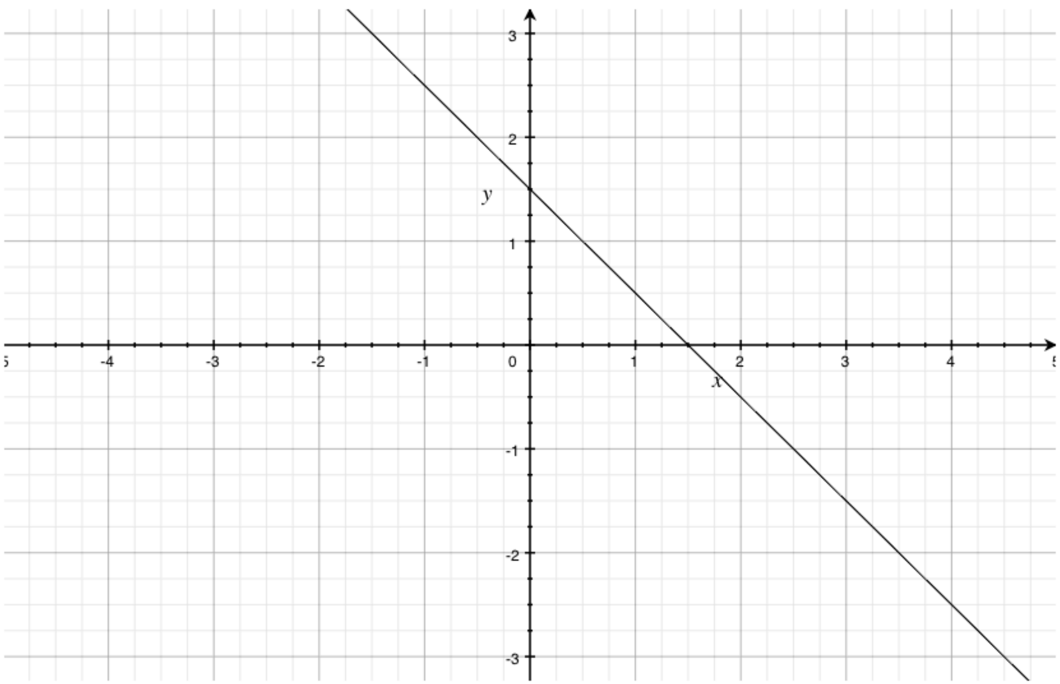
$$u(x) = x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_k}$$

can also be expressed as a LTF.

$$\mathrm{sign}(x_{i_1} + x_{i_2} + \ldots + x_{i_k} - \tfrac{1}{2})$$

Linear Threshold Function

Linear threshold function defines a plane in space, evaluates which side of the plane a point lies on.

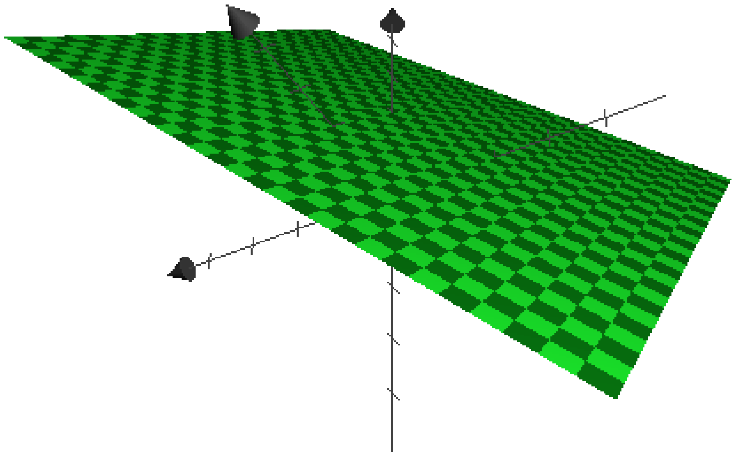


$$x + y = \frac{3}{2}$$

Linear Threshold Function

Linear threshold function defines a plane in space, evaluates which side of the plane a point lies on.

$$-x + y + z = 1$$



Linear Threshold Function

Can every function be expressed as a linear threshold function?

No, for example

$$\mathrm{PARITY}(x) = x_1 + x_2 + \cdots + x_n \mod 2$$

Note that linear threshold functions are monotone in each coordinate.

Linear Threshold Function

- In the winnow algorithm, Bob maintains as hypothesis a linear threshold function.

$$h(x) = \mathrm{sign}(w_1x_1 + w_2x_2 + \cdots + w_nx_n - \theta).$$

- Initially, all weights are set to 1 and θ=n.
- Bob guesses h(x) for the current hypothesis h.
  - If correct: stay the course
  - If wrong: update weights

Recap so far...

We want to learn a function which is a disjunction of k variables out of n possible.

$$u(x) = x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_k}$$

Initially, we take as hypothesis the function

$$\mathrm{sign}(x_1 + x_2 + \ldots + x_n - n)$$

After each mistake we will update the weight of each variable. Threshold will stay the same.

Update Rules

In the winnow algorithm, Bob maintains as hypothesis a linear threshold function.

$$h(x) = \text{sign}(w_1 x_1 + \cdots + w_n x_n - n)$$

Update: On false positive, i.e. u(x)=0 and h(x)=1, Bob sets

$$w_i = 0 \text{ for all } i \text{ where } x_i = 1.$$

On false negatives, i.e. u(x)=1 and h(x)=0, Bob sets

$$w_i \leftarrow 2w_i \text{ for all } i \text{ where } x_i = 1.$$

Reasonable Update?

Update: On false positive, i.e. u(x)=0 and h(x)=1, Bob sets

$$w_i = 0 \text{ for all } i \text{ where } x_i = 1.$$

On false negatives, i.e. u(x)=1 and h(x)=0, Bob sets

$$w_i \leftarrow 2w_i \text{ for all } i \text{ where } x_i = 1.$$

If the unknown disjunction contains x_i , then w_i will never be set to 0.

On false negatives, give more weight to all variables which could be making the disjunction true.

Simple Observations

$$h(x) = \mathrm{sign}(w_1 x_1 + \cdots + w_n x_n - n)$$

Update: On false positive, i.e. u(x)=0 and h(x)=1, Bob sets

$$w_i = 0 \text{ for all } i \text{ where } x_i = 1.$$

On false negatives, i.e. u(x)=1 and h(x)=0, Bob sets

$$w_i \leftarrow 2w_i \text{ for all } i \text{ where } x_i = 1.$$

1) Weights always remain non-negative.

2) No weight will become larger than 2n.

Mistake Bound

Call a weight doubling a promotion, and a weight being set to 0 a demotion.

By the second observation, a weight will only be promoted at most log n times. Otherwise, becomes larger than 2n.

As there are only k relevant variables in the unknown disjunction, total number of promotions is at most k log n.

Mistake Bound

Consider when u(x)=0 and h(x)=1. Then

$$\sum_{i:x_i=1} w_i \geq n.$$

After update, all of these weights will be set to 0.

Thus the sum of all the weights will decrease by at least n after the update.

Mistake Bound

Consider when u(x)=1 and h(x)=0. Then

$$\sum_{i:x_i=1} w_i < n.$$

After update, all of these weights will be doubled.

Thus the sum of all the weights will increase by at most n after the update.

Mistake Bound

- For every demotion:
  - sum of weights decreases by at least n.
- For every promotion:
  - sum of weights increases by at most n.
- Total number of promotions is at most k log n.
- We know that the sum of the weights must remain non-negative.
- Thus number of demotions at most 1+k log n.
- If the unknown function is a k-disjunction, the total number of mistakes of the winnow algorithm is bounded by about 2k log n.
- Still suffers from errors in Teacher ☹

A Softer Version

This algorithm is harsh---once a weight is set
to zero it can never come back.

We can instead consider a softer variation:

Update: On false positive, i.e. u(x)=0 and h(x)=1, Bob sets

$$w_i \leftarrow \frac{w_i}{2} \text{ for all } i \text{ where } x_i = 1$$

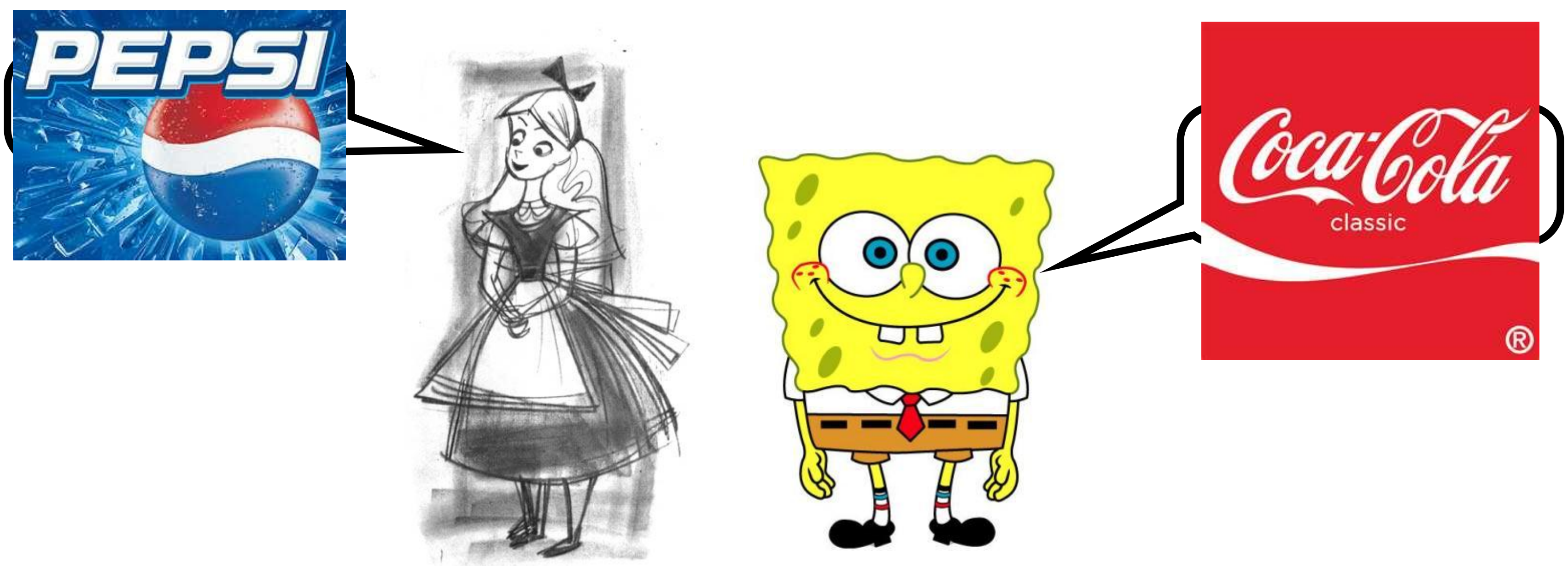
On false negatives, i.e. u(x)=1 and h(x)=0, Bob sets

$$w_i \leftarrow 2w_i \text{ for all } i \text{ where } x_i = 1.$$

Learning from Experts

Will Pepsi outperform Cocacola next month??

Everyone has a prediction!



Learning from Experts

- Whom should we listen to?
  - n experts
  - k is number of mistakes made by best expert
  - t is number of predictions
- Regret = Best Expert's Performance - Our Performance
- Can follow an amazing algorithm!
  - algorithm makes < $k + \sqrt{t \log(n)}$ mistakes
  - This is without knowing the best expert in advance!

Weighted Majority

The algorithm is very similar to Winnow.

Give a weight to each expert. Initially, set all weights equal to 1.

Predict according to the side with more weight.

After each example, update the weights:

If expert i is wrong: $w_i \leftarrow 0.99 w_i$

If expert i is right: $w_i \leftarrow w_i$

Uses of multiplicative Updates

- Weak Learners to Strong Learners
  - Adaboost
- Hard Core Sets
- Approximate solutions to linear programs
- Approximate solutions to semidefinite programs
- Find optimal strategies in zero-sum games