CS446: Policies

- **Cheating**
  No.
  *We take it very seriously.*

- **Homework:**
  - Collaboration is encouraged
  - But, you have to write your own solution/program.
  - *(Please don’t use old solutions)*

- **Late Policy:**
  You have a credit of 4 days (4*24hours); That’s it.

- **Grading:**
  - Possibly separate for grads/undergrads.
  - 5% Class work; 25% - homework; 30%-midterm; 40%-final;
  - Projects: 25% (4 hours)

- **Questions?**
CS446 Team

- **Dan Roth** (3323 Siebel)
  - Tuesday/Thursday, 1:45 PM – 2:30 PM (or: appointment)

- **TAs** (office hours will be outside the offices below)
  - Adam Vollrath: Wed 1:00pm-2:00pm (3333 SC)
  - Daniel Khashabi: Tue 11:15pm-2:15pm (3333 SC)
  - Haoruo Peng: Mon 3:30pm-4:30pm (3333 SC)
  - Himel Dev: Wed 3:30pm-4:30pm (1117 SC)
  - Shyam Upadhyay: Mon 2:30pm-3:30pm (3333 SC)

- **Discussion Sections**: (starting 3rd week)
  - Mondays: 5:00pm-6:00pm 3405-SC  Adam Vollrath [A-C]
  - Tuesday: 6:00pm-7:00pm 3405-SC  Shyam Upadhyay [D-H]
  - Wednesdays: 5:00pm-6:00pm 3405-SC  Himel Dev [I-M]
  - Thursdays: 6:00pm-7:00pm 3405-SC  Haoruo Peng [N-T]
  - Fridays: 3:00pm-4:00pm 3405-SC  Daniel Khashab [U-Z]
Check our class website:

- Schedule, slides, videos, policies

- Sign up, participate in our Piazza forum:
  - Announcements and discussions
  - [https://piazza.com/class#fall2015/cs446](https://piazza.com/class#fall2015/cs446)

- Log on to Compass:
  - Submit assignments, get your grades
  - [https://compass2g.illinois.edu](https://compass2g.illinois.edu)
CS446 on the web

Check our class website:
- Schedule, slides, videos, policies
  - [http://l2r.cs.uiuc.edu/~danr/Teaching/CS446-14/index.html](http://l2r.cs.uiuc.edu/~danr/Teaching/CS446-14/index.html)
- Sign up, participate in our Piazza forum:
  - Announcements and discussions
    - [https://piazza.com/class#fall2014/cs446](https://piazza.com/class#fall2014/cs446)
- Log on to Compass:
  - Submit assignments, get your grades
    - [https://compass2g.illinois.edu](https://compass2g.illinois.edu)

Homework:
No need to submit Hw0;
Later: submit electronically.
What is Learning

- The Badges Game
  - This is an example of the key learning protocol: supervised learning
- First question: Are you sure you got it right?
  - Why?
- Issues:
  - Prediction or Modeling?
  - Representation
  - Problem setting
  - Background Knowledge
  - When did learning take place?
  - Algorithm
Training data

+ Naoki Abe
- Myriam Abramson
+ David W. Aha
+ Kamal M. Ali
- Eric Allender
+ Dana Angluin
- Chidanand Apte
+ Minoru Asada
+ Lars Asker
+ Javed Aslam
+ Jose L. Balcazar
- Cristina Baroglio
+ Peter Bartlett
- Eric Baum
+ Welton Becket
- Shai Ben-David
+ George Berg
+ Neil Berkman
+ Malini Bhandaru
+ Bir Bhanu
+ Reinhard Blasig
- Avrim Blum
- Anselm Blumer
+ Justin Boyan
+ Carla E. Brodley
+ Nader Bshouty
- Wray Buntine
- Andrey Burago
+ Tom Bylander
+ Bill Byrne
- Claire Cardie
+ John Case
+ Jason Catlett
- Philip Chan
- Zhixiang Chen
- Chris Darken
The Badges game

+ Naoki Abe  - Eric Baum

Conference attendees to the 1994 Machine Learning conference were given name badges labeled with + or −.

What function was used to assign these labels?
Raw test data

Gerald F. DeJong
Chris Drummond
Yolanda Gil
Attilio Giordana
Jiarong Hong
J. R. Quinlan

Priscilla Rasmussen
Dan Roth
Yoram Singer
Lyle H. Ungar
Labeled test data

+ Gerald F. DeJong
- Chris Drummond
+ Yolanda Gil
- Attilio Giordana
+ Jiarong Hong
- J. R. Quinlan
- Priscilla Rasmussen
+ Dan Roth
+ Yoram Singer
- Lyle H. Ungar
What is Learning

- The Badges Game
  - This is an example of the key learning protocol: supervised learning

- First question: Are you sure you got it right?
  - Why?

- Issues:
  - Prediction or Modeling?
  - Representation
  - Problem setting
  - Background Knowledge
  - When did learning take place?
  - Algorithm
Supervised Learning

We consider systems that apply a function $f()$ to input items $x$ and return an output $y = f(x)$. 

- An item $x \in X$ drawn from an input space $X$
- An item $y \in Y$ drawn from an output space $Y$
- System $y = f(x)$
In (supervised) machine learning, we deal with systems whose \( f(x) \) is learned from examples.
We typically use machine learning when the function $f(x)$ we want the system to apply is too complex to program by hand.
Supervised learning

Input

\( x \in X \)

An item \( x \) drawn from an instance space \( X \)

Target function

\( y = f(x) \)

Output

An item \( y \) drawn from a label space \( Y \)

Learned Model

\( y = g(x) \)
Supervised learning: Training

Labeled Training Data

\[ \mathcal{D}_{\text{train}} \]

\( (x_1, y_1) \)
\( (x_2, y_2) \)
...
\( (x_N, y_N) \)

Learning Algorithm

Learned model \( g(x) \)

- Give the learner examples in \( \mathcal{D}_{\text{train}} \)
- The learner returns a model \( g(x) \)

Can you suggest other learning protocols?
Supervised learning: Testing

Labeled Test Data

\( D_{test} \)

\( (x'_1, y'_1) \)

\( (x'_2, y'_2) \)

\( \ldots \)

\( (x'_M, y'_M) \)

- Reserve some labeled data for testing
Supervised learning: Testing

Labeled Test Data
\( \mathcal{D}_{\text{test}} \)

Raw Test Data
\( \mathbf{X}_{\text{test}} \)

Test Labels
\( \mathbf{y}_{\text{test}} \)

\((x'_{1}, y'_1)\)
\((x'_{2}, y'_2)\)
...
\((x'_{M}, y'_M)\)

\(x'_{1}\)
\(x'_{2}\)
...
\(x'_{M}\)
Supervised learning: Testing

- Apply the model to the raw test data
- Evaluate by comparing predicted labels against the test labels

Can you use the test data otherwise?

- Raw Test Data $\mathbf{X}^{\text{test}}$
  - $x'_1$
  - $x'_2$
  - ....
  - $x'_M$

- Learned model $g(\mathbf{x})$

- Predicted Labels $g(\mathbf{X}^{\text{test}})$
  - $g(x'_1)$
  - $g(x'_2)$
  - ....
  - $g(x'_M)$

- Test Labels $\mathbf{Y}^{\text{test}}$
  - $y'_1$
  - $y'_2$
  - ....
  - $y'_M$
What is Learning

- The Badges Game
  - This is an example of the key learning protocol: supervised learning

- First question: Are you sure you got it?
  - Why?

- Issues:
  - Prediction or Modeling?
  - Representation
  - Problem setting
  - Background Knowledge
  - When did learning take place?
  - Algorithm
Course Overview

- Introduction: Basic problems and questions
- A detailed example: Linear threshold units
  - Online Learning
- Two Basic Paradigms:
  - PAC (Risk Minimization)
  - Bayesian theory
- Learning Protocols:
  - Supervised; Unsupervised; Semi-supervised
- Algorithms
  - Decision Trees (C4.5)
  - [Rules and ILP (Ripper, Foil)]
  - Linear Threshold Units (Winnow; Perceptron; Boosting; SVMs; Kernels)
  - [Neural Networks (Backpropagation)]
  - Probabilistic Representations (naïve Bayes; Bayesian trees; Densities)
  - Unsupervised /Semi supervised: EM
- Clustering; Dimensionality Reduction
Supervised Learning

Given: Examples \((x, f(x))\) of some unknown function \(f\)

Find: A good approximation of \(f\)

\(x\) provides some representation of the input
- The process of mapping a domain element into a representation is called Feature Extraction. (Hard; ill-understood; important)
- \(x \in \{0,1\}^n\) or \(x \in \mathbb{R}^n\)

The target function (label)
- \(f(x) \in \{-1,+1\}\) Binary Classification
- \(f(x) \in \{1,2,3,..,k-1\}\) Multi-class classification
- \(f(x) \in \mathbb{R}\) Regression
Supervised Learning: Examples

- **Disease diagnosis**
  - \( x \): Properties of patient (symptoms, lab tests)
  - \( f \): Disease (or maybe: recommended therapy)

- **Part-of-Speech tagging**
  - \( x \): An English sentence (e.g., The can will rust)
  - \( f \): The part of speech of a word in the sentence

- **Face recognition**
  - \( x \): Bitmap picture of person’s face
  - \( f \): Name the person (or maybe: a property of)

- **Automatic Steering**
  - \( x \): Bitmap picture of road surface in front of car
  - \( f \): Degrees to turn the steering wheel

Many problems that do not seem like classification problems can be decomposed to classification problems. E.g, *Semantic Role Labeling*
Key Issues in Machine Learning

- **Modeling**
  - How to formulate application problems as machine learning problems? How to represent the data?
  - Learning Protocols (where is the data & labels coming from?)

- **Representation**
  - What functions should we learn (hypothesis spaces)?
  - How to map raw input to an instance space?
  - Any rigorous way to find these? Any general approach?

- **Algorithms**
  - What are good algorithms?
  - How do we define success?
  - Generalization Vs. over fitting
  - The computational problem
Using supervised learning

- What is our instance space?
  - Gloss: What kind of features are we using?

- What is our label space?
  - Gloss: What kind of learning task are we dealing with?

- What is our hypothesis space?
  - Gloss: What kind of model are we learning?

- What learning algorithm do we use?
  - Gloss: How do we learn the model from the labeled data?

- What is our loss function/evaluation metric?
  - Gloss: How do we measure success? What drives learning?
1. The instance space $\mathcal{X}$

**Input**

$x \in \mathcal{X}$

An item $x$ drawn from an instance space $\mathcal{X}$

**Output**

$y \in \mathcal{Y}$

An item $y$ drawn from a label space $\mathcal{Y}$

Learned Model

$y = g(x)$

- Designing an appropriate instance space $\mathcal{X}$ is crucial for how well we can predict $y$. 

INTRODUCTION   CS446 Fall ’15
1. The instance space $\mathcal{X}$

- When we apply machine learning to a task, we first need to define the instance space $\mathcal{X}$.
- Instances $x \in \mathcal{X}$ are defined by features:
  - Boolean features:
    - Does this email contain the word ‘money’?
  - Numerical features:
    - How often does ‘money’ occur in this email? What is the width/height of this bounding box?
What’s $\chi$ for the Badges game?

- Possible features:
  - Gender/age/country of the person?
  - Length of their first or last name?
  - Does the name contain letter ‘x’?
  - How many vowels does their name contain?
  - Is the n-th letter a vowel?
X as a vector space

- X is an N-dimensional vector space (e.g. \( \mathbb{R}^N \))
  - Each dimension = one feature.
- Each x is a feature vector (hence the boldface x).
- Think of \( x = [x_1 \ldots x_N] \) as a point in X:

![Diagram](image)
From feature templates to vectors

- When designing features, we often think in terms of **templates**, not individual features:

- **What is the 2nd letter?**
  - Naoki $\rightarrow [1\ 0\ 0\ 0\ ...]$  
  - Abe $\rightarrow [0\ 1\ 0\ 0\ ...]$  
  - Scrooge $\rightarrow [0\ 0\ 1\ 0\ ...]$  

- **What is the $i$-th letter?**
  - Abe $\rightarrow [1\ 0\ 0\ 0\ 0\ ...\ 0\ 1\ 0\ 0\ 0\ 0\ ...\ 0\ 0\ 0\ 0\ 1\ ...]$  
    - 26*2 positions in each group;  
    - # of groups == upper bounds on length of names
Good features are essential

The choice of features is crucial for how well a task can be learned.
- In many application areas (language, vision, etc.), a lot of work goes into designing suitable features.
- This requires domain expertise.

CS446 can’t teach you what specific features to use for your task.
- But we will touch on some general principles
2. The label space $\mathbf{y}$

- **Input**
  - $x \in \mathbf{X}$
  - An item $x$ drawn from an instance space $\mathbf{X}$

- **Learned Model**
  - $y = g(x)$

- **Output**
  - $y \in \mathbf{Y}$
  - An item $y$ drawn from a label space $\mathbf{Y}$

- The label space $\mathbf{Y}$ determines what kind of supervised learning task we are dealing with
Output labels $y \in Y$ are categorical:

- **Binary classification**: Two possible labels
- **Multiclass classification**: $k$ possible labels

- Output labels $y \in Y$ are **structured** objects (sequences of labels, parse trees, etc.)
- **Structure learning**
Output labels $y \in Y$ are numerical:

- **Regression (linear/polynomial):**
  - Labels are continuous-valued
  - Learn a linear/polynomial function $f(x)$

- **Ranking:**
  - Labels are ordinal
  - Learn an ordering $f(x_1) > f(x_2)$ over input
3. The model $g(x)$

- **Input**
  - $x \in \mathcal{X}$
  - An item $x$ drawn from an instance space $\mathcal{X}$

- **Learned Model**
  - $y = g(x)$

- **Output**
  - $y \in \mathcal{Y}$
  - An item $y$ drawn from a label space $\mathcal{Y}$

- We need to choose what *kind* of model we want to learn.
A Learning Problem

\[ y = f(x_1, x_2, x_3, x_4) \]

<table>
<thead>
<tr>
<th>Example</th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>X4</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Can you learn this function? What is it?
Complete Ignorance:
There are $2^{16} = 65536$ possible functions over four input features.

We can’t figure out which one is correct until we’ve seen every possible input-output pair.

After seven examples we still have $2^9$ possibilities for $f$

Is Learning Possible?

<table>
<thead>
<tr>
<th>Example</th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>X4</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- There are $|Y|^{|X|}$ possible functions $f(x)$ from the instance space $X$ to the label space $Y$.
- Learners typically consider only a subset of the functions from $X$ to $Y$, called the hypothesis space $H$. $H \subseteq |Y|^{|X|}$
**Hypothesis Space (2)**

**Simple Rules:** There are only 16 simple **conjunctive rules** of the form \( y = x_i \land x_j \land x_k \).

<table>
<thead>
<tr>
<th>Rule</th>
<th>Counterexample</th>
<th>Rule</th>
<th>Counterexample</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y = c )</td>
<td></td>
<td>( x_2 \land x_3 )</td>
<td>0011 1</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>1100 0</td>
<td>( x_2 \land x_4 )</td>
<td>0011 1</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>0100 0</td>
<td>( x_3 \land x_4 )</td>
<td>1001 1</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>0110 0</td>
<td>( x_1 \land x_2 \land x_3 )</td>
<td>0011 1</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>0101 1</td>
<td>( x_1 \land x_2 \land x_4 )</td>
<td>0011 1</td>
</tr>
<tr>
<td>( x_1 \land x_2 )</td>
<td>1100 0</td>
<td>( x_1 \land x_3 \land x_4 )</td>
<td>0011 1</td>
</tr>
<tr>
<td>( x_1 \land x_3 )</td>
<td>0011 1</td>
<td>( x_2 \land x_3 \land x_4 )</td>
<td>0011 1</td>
</tr>
<tr>
<td>( x_1 \land x_4 )</td>
<td>0011 1</td>
<td>( x_1 \land x_2 \land x_3 \land x_4 )</td>
<td>0011 1</td>
</tr>
</tbody>
</table>

No simple rule explains the data. The same is true for **simple clauses**.
**Hypothesis Space (3)**

**Notation:** 2 variables from the set on the left. **Value:** Index of the counterexample.

m-of-n rules: There are 32 possible rules of the form “y = 1 if and only if at least m of the following n variables are 1”

<table>
<thead>
<tr>
<th>variables</th>
<th>1-of</th>
<th>2-of</th>
<th>3-of</th>
<th>4-of</th>
</tr>
</thead>
<tbody>
<tr>
<td>{X1}</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{X2}</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{X3}</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{X4}</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{X1, X2}</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{X1, X3}</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{X1, X4}</td>
<td>6</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{X2, X3}</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>variables</th>
<th>1-of</th>
<th>2-of</th>
<th>3-of</th>
<th>4-of</th>
</tr>
</thead>
<tbody>
<tr>
<td>{X2, X4}</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{X3, X4}</td>
<td>4</td>
<td>4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{X1, X2, X3}</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>{X1, X2, X4}</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>{X1, X3, X4}</td>
<td>1</td>
<td>***</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>{X2, X3, X4}</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>{X1, X2, X3, X4}</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Found a consistent hypothesis.
Views of Learning

- Learning is the removal of our remaining uncertainty:
  - Suppose we knew that the unknown function was an m-of-n Boolean function, then we could use the training data to infer which function it is.
- Learning requires guessing a good, small hypothesis class:
  - We can start with a very small class and enlarge it until it contains an hypothesis that fits the data.
- We could be wrong!
  - Our prior knowledge might be wrong:
    - $y = x_4 \land \text{one-of (x}_1, x_3\text{)}$ is also consistent
  - Our guess of the hypothesis space could be wrong
- If this is the unknown function, then we will make errors when we are given new examples, and are asked to predict the value of the function
General strategies for Machine Learning

- Develop flexible hypothesis spaces:
  - Decision trees, neural networks, nested collections.

- Develop representation languages for restricted classes of functions:
  - Serve to limit the expressivity of the target models
  - E.g., Functional representation (n-of-m); Grammars; linear functions; stochastic models;
  - Get flexibility by augmenting the feature space

In either case:

- Develop algorithms for finding a hypothesis in our hypothesis space, that fits the data
- And hope that they will generalize well
Terminology

- **Target function (concept):** The true function \( f : X \rightarrow \{\ldots \text{Labels} \ldots\} \)

- **Concept:** Boolean function. Example for which \( f(x) = 1 \) are positive examples; those for which \( f(x) = 0 \) are negative examples (instances)

- **Hypothesis:** A proposed function \( h \), believed to be similar to \( f \). The output of our learning algorithm.

- **Hypothesis space:** The space of all hypotheses that can, in principle, be the output of the learning algorithm.

- **Classifier:** A discrete valued function produced by the learning algorithm. The possible value of \( f \): \( \{1,2,\ldots K\} \) are the classes or class labels. (In most algorithms the classifier will actually return a real valued function that we’ll have to interpret).

- **Training examples:** A set of examples of the form \( \{(x, f(x))\} \)
Registration

**Hw1** is out
- Please start working on it as soon as possible
- Come to sections with questions

Projects
- Still in the making

On Thursday, we will have two lectures:
- Usual one, 12:30-11:45
- An additional one, 7pm-8:15pm; 1320 DCL
An Example

I don’t know \{whether, weather\} to laugh or cry

How can we make this a learning problem?

- We will look for a function
  \[ F: \text{Sentences} \rightarrow \{\text{whether, weather}\} \]
- We need to define the domain of this function better.

- **An option:** For each word \(w\) in English define a Boolean feature \(x_w\):
  \[ [x_w = 1] \iff w \text{ is in the sentence} \]
  - This maps a sentence to a point in \(\{0,1\}^{50,000}\)
  - In this space: some points are \textbf{whether} points
    some are \textbf{weather} points
Representation Step: What’s Good?

- Learning problem: Find a function that best separates the data
- What function?
- What’s best?
- (How to find it?)

A possibility: Define the learning problem to be: A (linear) function that best separates the data

Linear = linear in the feature space

$$x = \text{data representation}; \; w = \text{the classifier}$$

$$y = \text{sgn} \{w^T x\}$$

- Memorizing vs. Learning
- Accuracy vs. Simplicity
- How well will you do?
- On what?
- Impact on Generalization
Expressivity

\[ f(x) = \text{sgn} \{ x \cdot w - \theta \} = \text{sgn} \{ \sum_{i=1}^{n} w_i x_i - \theta \} \]

- Many functions are Linear
  - Conjunctions:
    - \( y = x_1 \land x_3 \land x_5 \)
    - \( y = \text{sgn}\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 3\}; \quad w = (1, 0, 1, 0, 1) \quad \theta = 3 \)
  - At least \( m \) of \( n \):
    - \( y = \text{at least 2 of } \{x_1, x_3, x_5\} \)
    - \( y = \text{sgn}\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 2\}; \quad w = (1, 0, 1, 0, 1) \quad \theta = 2 \)

- Many functions are not
  - Xor: \( y = x_1 \land x_2 \lor \neg x_1 \land \neg x_2 \)
  - Non trivial DNF: \( y = x_1 \land x_2 \lor x_3 \land x_4 \)

- But can be made linear

Probabilistic Classifiers as well
Exclusive-OR (XOR)

In general: a parity function.

- \( (x_1 \land x_2) \lor (\neg \{x_1\} \land \neg \{x_2\}) \)

- \( x_i \in \{0, 1\} \)

- \( f(x_1, x_2, \ldots, x_n) = 1 \) iff \( \sum x_i \) is even

This function is not linearly separable.
Functions Can be Made Linear

- Data are not linearly separable in one dimension
- Not separable if you insist on using a specific class of functions
Blown Up Feature Space

- Data are separable in $\langle x, x^2 \rangle$ space

- Key issue: Representation:
  - what features to use.
  - Computationally, can be done implicitly (kernels)
  - Not always ideal.
Functions Can be Made Linear

Discrete Case

New discriminator is functionally simpler

\[ y_3 \lor y_4 \lor y_7 \]

A real Weather/Whether example

\[ x_1 \ x_2 \ x_4 \lor x_2 \ x_4 \ x_5 \lor x_1 \ x_3 \ x_7 \]

Space: \( X = x_1, x_2, \ldots, x_n \)

New Space: \( Y = \{y_1, y_2, \ldots\} = \{x_i, x_i \ x_j, x_i \ x_j \ x_j, \ldots\} \)
Third Step: How to Learn?

- A possibility: Local search
  - Start with a linear threshold function.
  - See how well you are doing.
  - Correct
  - Repeat until you converge.

- There are other ways that do not search directly in the hypotheses space
  - Directly compute the hypothesis
A General Framework for Learning

- **Goal**: predict an unobserved output value \( y \in Y \) based on an observed input vector \( x \in X \)

- Estimate a functional relationship \( y \sim f(x) \) from a set \( \{(x,y)_i\}_{i=1,n} \)

- Most relevant - **Classification**: \( y \in \{0,1\} \) (or \( y \in \{1,2,...,k\} \))
  - (But, within the same framework can also talk about **Regression**, \( y \in \mathbb{R} \))

- What do we want \( f(x) \) to satisfy?
  - **We want to minimize the Risk**: \( L(f()) = E_{X,Y}(I[f(x)\neq y]) \)
  - **Where**: \( E_{X,Y} \) denotes the expectation with respect to the true distribution.

**Simple loss function**: # of mistakes

[...] is a indicator function
We want to minimize the Loss: \[ L(f()) = E_{X,Y}( [f(X) \neq Y] ) \]

Where: \( E_{X,Y} \) denotes the expectation with respect to the true distribution.

We cannot minimize this loss.

Instead, we try to minimize the empirical classification error. For a set of training examples \( \{(X_i,Y_i)\}_{i=1,n} \)

Try to minimize: \[ L'(f()) = \frac{1}{n} \sum_i [f(X_i) \neq Y_i] \]

(Issue I: why/when is this good enough? Not now)

This minimization problem is typically NP hard.

To alleviate this computational problem, minimize a new function – a convex upper bound of the classification error function

\[ I(f(x), y) = [f(x) \neq y] = \{1 \text{ when } f(x) \neq y; 0 \text{ otherwise}\} \]

Side note: If the distribution over \( X \times Y \) is known, predict: \( y = \arg \max_y P(y|x) \)

This is the best possible (the optimal Bayes' error).
A Loss Function $L(f(x), y)$ measures the penalty incurred by a classifier $f$ on example $(x, y)$.

There are many different loss functions one could define:

- **Misclassification Error:**
  \[ L(f(x), y) = 0 \text{ if } f(x) = y; \quad 1 \text{ otherwise} \]

- **Squared Loss:**
  \[ L(f(x), y) = (f(x) - y)^2 \]

- **Input dependent loss:**
  \[ L(f(x), y) = 0 \text{ if } f(x) = y; \quad c(x) \text{ otherwise.} \]

A continuous convex loss function allows a simpler optimization algorithm.
Here \( f(x) \) is the prediction \( \in \mathbb{R} \)
\( y \in \{-1, 1\} \) is the correct value

**0-1 Loss**  
\[ L(y, f(x)) = \frac{1}{2} (1 - \text{sgn}(yf(x))) \]

**Log Loss**  
\[ \frac{1}{\ln 2} \log (1 + \exp{-yf(x)}) \]

**Hinge Loss**  
\[ L(y, f(x)) = \max(0, 1 - yf(x)) \]

**Square Loss**  
\[ L(y, f(x)) = (y - f(x))^2 \]

0-1 Loss: x axis = \( yf(x) \)
Log Loss: x axis = \( yf(x) \)
Hinge Loss: x axis = \( yf(x) \)
Square Loss: x axis = \( (y - f(x) - 1) \)
Example

Putting it all together:

A Learning Algorithm
Third Step: How to Learn?

- A possibility: Local search
  - Start with a linear threshold function.
  - See how well you are doing.
  - Correct
  - Repeat until you converge.

- There are other ways that do not search directly in the hypotheses space
  - Directly compute the hypothesis
Learning Linear Separators (LTU)

\[ f(x) = \text{sgn} \{x^T \cdot w - \theta\} = \text{sgn}\{\sum_{i=1}^{n} w_i x_i - \theta\} \]

- \( x^T = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n \)
  - is the feature based encoding of the data point

- \( w^T = (w_1, w_2, \ldots, w_n) \in \mathbb{R}^n \)
  - is the target function.

- \( \theta \) determines the shift with respect to the origin
Canonical Representation

\[ f(x) = \text{sgn} \{ w^T \cdot x - \theta \} = \text{sgn} \{ \sum_{i=1}^{n} w_i x_i - \theta \} \]

- \[ \text{sgn} \{ w^T \cdot x - \theta \} \equiv \text{sgn} \{ (w')^T \cdot x' \} \]
- **Where:**
  - \[ x' = (x, -1) \] and \[ w' = (w, \theta) \]

- Moved from an \( n \) dimensional representation to an \( (n+1) \) dimensional representation, but now can look for hyperplanes that go through the origin.
Our goal is to find a \( w \) that minimizes the expected risk

\[
J(w) = \mathbb{E}_{x,y} Q(x, y, w)
\]

We cannot do it.

Instead, we approximate \( J(w) \) using a finite training set of independent samples \((x_i, y_i)\)

\[
J(w) \approx 1/m \sum_{1}^{m} Q(x_i, y_i, w)
\]

To find the minimum, we use a batch gradient descent algorithm.

That is, we successively compute estimates \( w^t \) of the optimal parameter vector \( w \):

\[
w^{t+1} = w^t - \nabla J(w) = w^t - 1/m \sum_{1}^{m} \nabla Q(x_i, y_i, w)
\]
We use gradient descent to determine the weight vector that minimizes $J(w) = \text{Err}(w)$.

Fixing the set $D$ of examples, $J=\text{Err}$ is a function of $w^j$.

At each step, the weight vector is modified in the direction that produces the steepest descent along the error surface.
Our Hypothesis Space is the collection of **Linear Threshold Units**

- Loss function:
  - Squared loss LMS (Least Mean Square, $L_2$)
  - $Q(x, y, w) = \frac{1}{2} (w^T x - y)^2$
LMS: An Optimization Algorithm

- \( w(j) \) is the current weight vector we have.
- Our prediction on the \( d \)-th example \( x \) is:
  \[
  o_d = \sum_i w_i^j \cdot x_i = \mathbf{w}^{(i)} \cdot \mathbf{x}
  \]
- \( t_d \) is the target value for this example (real value; represents \( u \cdot x \)).
- The error the current hypothesis makes on the data set is:
  \[
  J(w) = \text{Err}(\mathbf{w}^{(i)}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2
  \]
Gradient Descent

To find the best direction in the weight space $\vec{w}$ we compute the gradient of $E$ with respect to each of the components of

$$\nabla E(\vec{w}) \equiv \left[ \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n} \right]$$

This vector specifies the direction that produces the steepest increase in $E$;

We want to modify $\vec{w}$ in the direction of $-\nabla E(\vec{w})$

$$\vec{w} = \vec{w} + \Delta \vec{w}$$

Where (with a fixed step size $R$):

$$\Delta \vec{w} = -R \nabla E(\vec{w})$$
Gradient Descent: LMS

- We have: \( \text{Err}(\vec{w}(j)) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \)

- Therefore:
  \[
  \frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 = \\
  = \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 = \\
  = \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w}_d \cdot \vec{x}_d) = \\
  = \sum_{d \in D} (t_d - o_d)(-x_{id})
  \]
Gradient Descent: LMS

Weight update rule:

$$\Delta w_i = R \sum_{d \in D} (t_d - o_d)x_{id}$$

Gradient descent algorithm for training linear units:

- Start with an initial random weight vector
- For every example $d$ with target value $t_d$ do:
  - Evaluate the linear unit $o_d = \sum_i w_i \cdot x_{id} = \vec{w} \cdot \vec{x}_d$
  - Update $\vec{w}$ by adding $\Delta w_i$ to each component
- Continue until $E$ below some threshold

This algorithm always converges to a local minimum of $J(w)$, for small enough steps. Here (LMS for linear regression), the surface contains only a single global minimum, so the algorithm converges to a weight vector with minimum error, regardless of whether the examples are linearly separable.

The surface may have local minimum if the loss function is different.
Weight update rule:

$$\Delta w_i = R(t_d - o_d)x_{id}$$

Gradient descent algorithm for training linear units:

- Start with an initial random weight vector
- For every example $d$ with target value $t_d$ do:
  - Evaluate the linear unit $o_d = \sum_i w_i \cdot x_{id} = \vec{w} \cdot \vec{x}_d$
  - update $\vec{w}$ by incrementally by adding $\Delta w_i$ to each component (update without summing over all data)
- Continue until $E$ below some threshold
Incremental (Stochastic) Gradient Descent: LMS

- **Weight update rule:**
  \[ \Delta w_i = R(t_d - o_d)x_{id} \]

- **Gradient descent algorithm for training linear units:**
  - Start with an initial random weight vector
  - For every example d with target value: \( t_d \)
    - Evaluate the linear unit: \( o_d = \sum_i w_i \cdot x_{id} = \vec{w} \cdot \vec{x}_d \)
    - update \( \vec{w} \) by incrementally adding \( \Delta w_i \) to each component (update without summing over all data)
  - Continue until E below some threshold

- In general - does not converge to global minimum
- Decreasing R with time guarantees convergence
- But, on-line algorithms are sometimes advantageous...

Dropped the averaging operation. Sometimes called “on-line” since we don’t need a reference to a training set: observe – predict – get feedback.
In the general (non-separable) case the learning rate $R$ must decrease to zero to guarantee convergence.

The learning rate is called the *step size*. There are more sophisticated algorithms that choose the step size automatically and converge faster.

Choosing a better starting point also has impact.

The gradient descent and its stochastic version are very simple algorithms, but almost all the algorithms we will learn in the class can be traced back to gradient decent algorithms for different loss functions and different hypotheses spaces.
Computational Issues

- Assume the data is linearly separable.
- Sample complexity:
  - Suppose we want to ensure that our LTU has an error rate (on new examples) of less than $\varepsilon$ with high probability (at least $(1-\delta)$)
  - How large does $m$ (the number of examples) must be in order to achieve this? It can be shown that for $n$ dimensional problems
    
    $$m = O\left(\frac{1}{\varepsilon} \left[ \ln\left(\frac{1}{\delta}\right) + (n+1) \ln\left(\frac{1}{\varepsilon}\right) \right] \right).$$

- Computational complexity: What can be said?
  - It can be shown that there exists a polynomial time algorithm for finding consistent LTU (by reduction from linear programming).
  - [Contrast with the NP hardness for 0-1 loss optimization]
  - (On-line algorithms have inverse quadratic dependence on the margin)
Other Methods for LTUs

- **Fisher Linear Discriminant:**
  - A direct computation method

- **Probabilistic methods (naïve Bayes):**
  - Produces a stochastic classifier that can be viewed as a linear threshold unit.

- **Winnow/Perceptron**
  - A multiplicative/additive update algorithm with some sparsity properties in the function space (a large number of irrelevant attributes) or features space (sparse examples)

- **Logistic Regression, SVM...many other algorithms**