# Machine Learning

## Model and Learning Algorithm Evaluation

Dan Goldwasser

dgoldwas@purdue.edu

## Goal for Today's class

- Model Evaluation and Online learning intro

- Evaluation Metrics
- Hypothesis tests intro
- Online learning and mistake bounds

## model evaluation

## Quick Review: Inductive Bias

- Unbiased learning is impossible!
- Inductive bias: a set of assumptions guiding learning **beyond the data**
- ***Preference for simpler functions!***
- We have seen one type of bias: **Language bias**
  - **Pick the right hypothesis space**

- Today, as part of our discussion on Decision Trees we will introduce a second type – **search bias.**
  - **Similar objective**: Encode assumptions about learning, restrict the complexity of the resulting hypothesis

## Quick Review: Overfitting

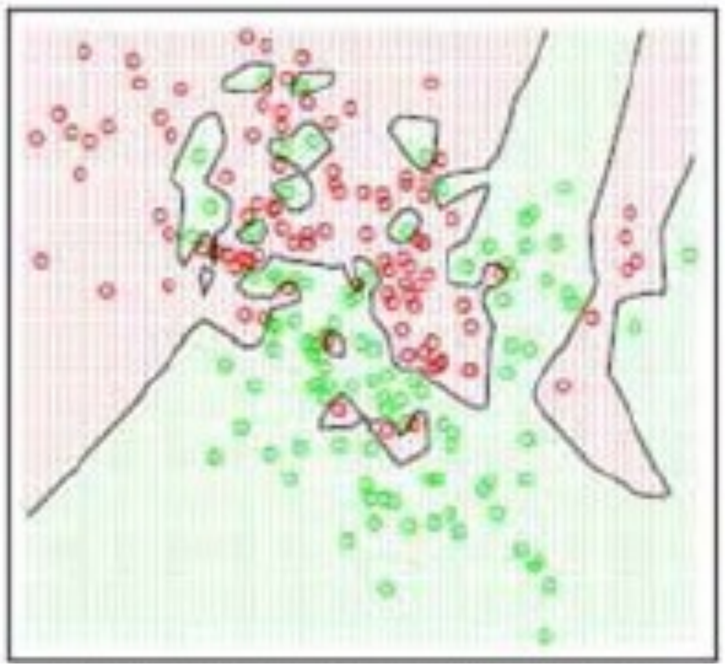
- Learning a tree classifying the training data perfectly may not have the best generalization
  - Algorithm fits tree to noise in training data
  - Sparse data set

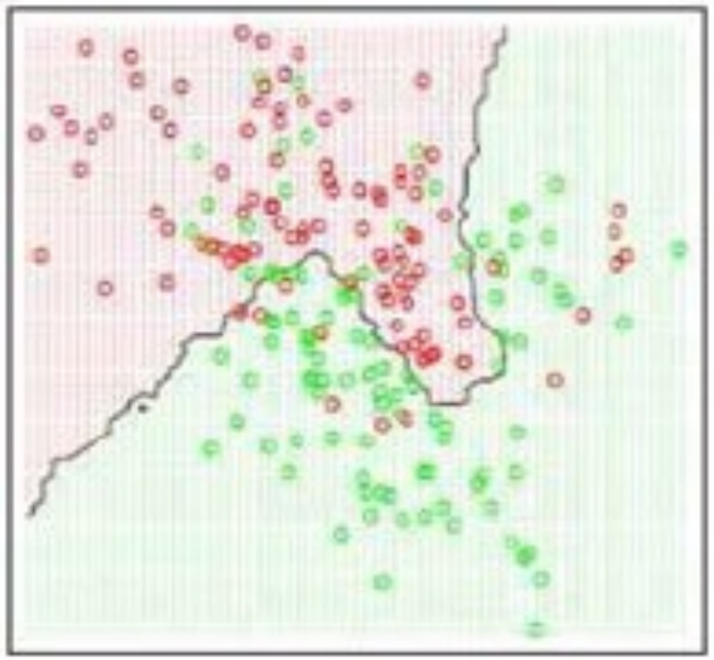
**A hypothesis h is said to overfit the training data** if there is another hypothesis h', such that h has a smaller error than h' on the training data but h has larger error on the test data.

# Quick Review: Expressivity

Let's take a closer look at the learned function

→ ***High sensitivity to noise!***

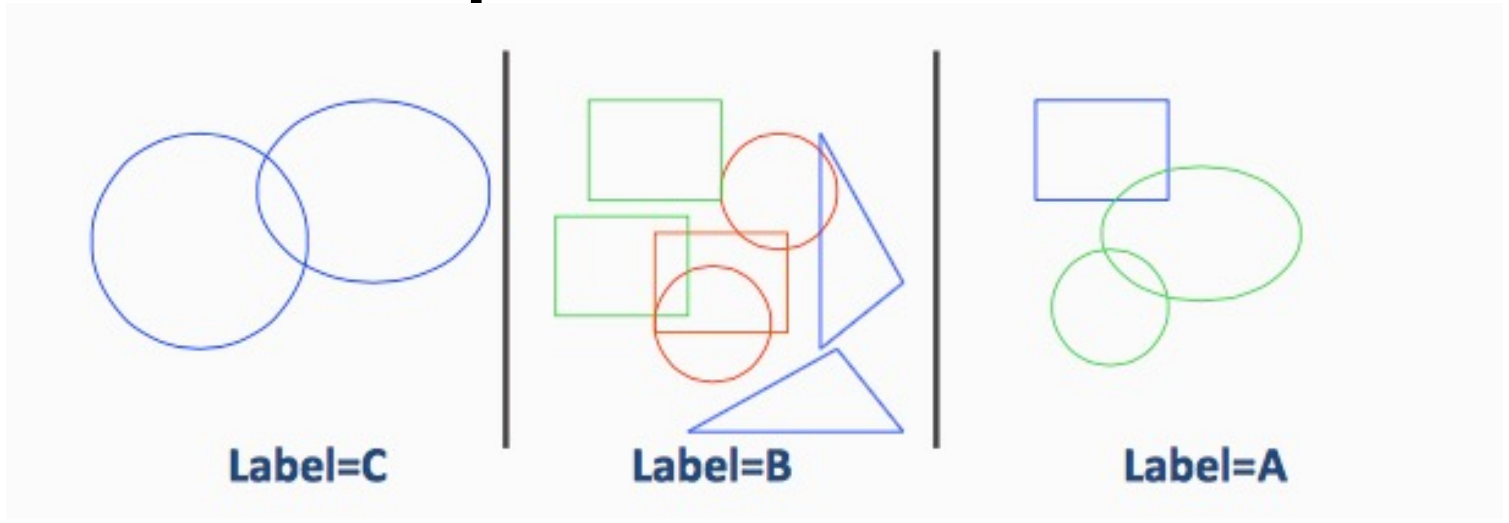


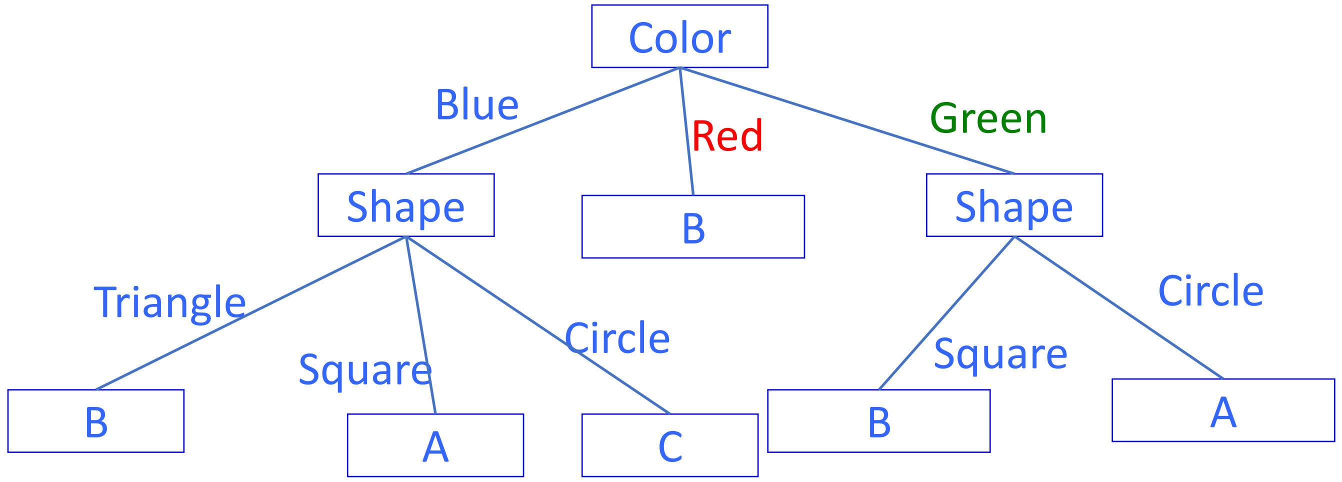


Higher k values results in smoother decision boundaries

Figures from Hastie, Tibshirani and Friedman (Elements of Statistical Learning)

## Decision Tree Representation



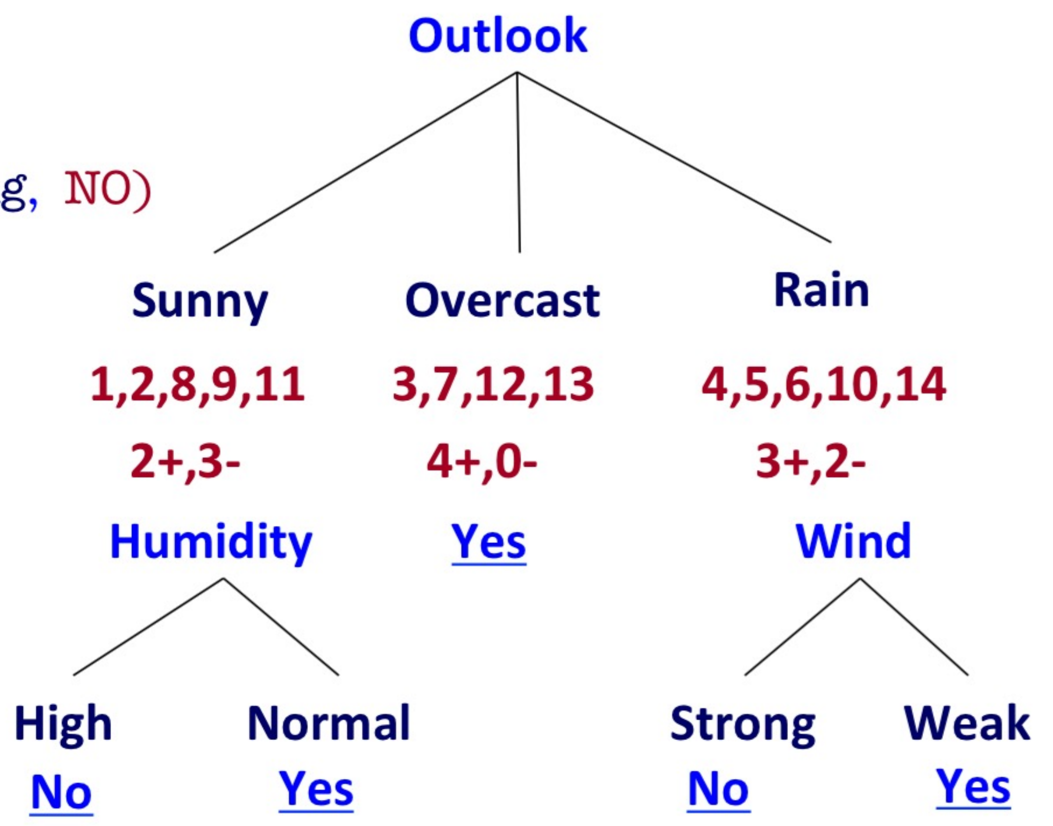


## Bias in decision trees

- Conduct a search of the space of decision trees
  - Can represent all possible functions
- **We prefer short trees!**
  - In DT learning this is implemented as a search bias
  - We bias the search to prefer shorter trees
- Other alternatives?
  - How would you implement language bias on DT?
- Search bias is implemented using greedy heuristics
  - Hill-climbing without backtracking
  - **Overfitting can still be an issue..**

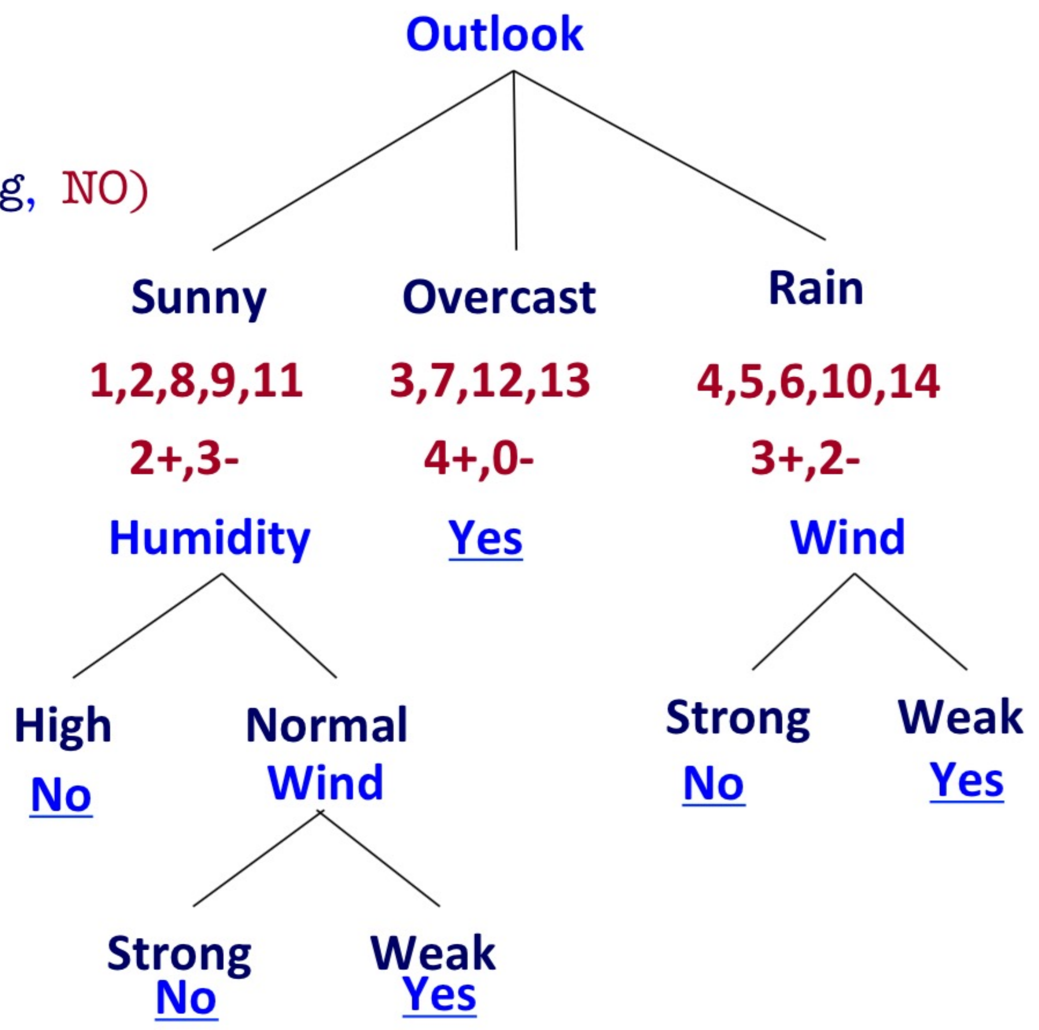
- Noisy example:

(Outlook = Sunny, Temp = Hot,
Humidity = Normal, Wind = Strong, NO)



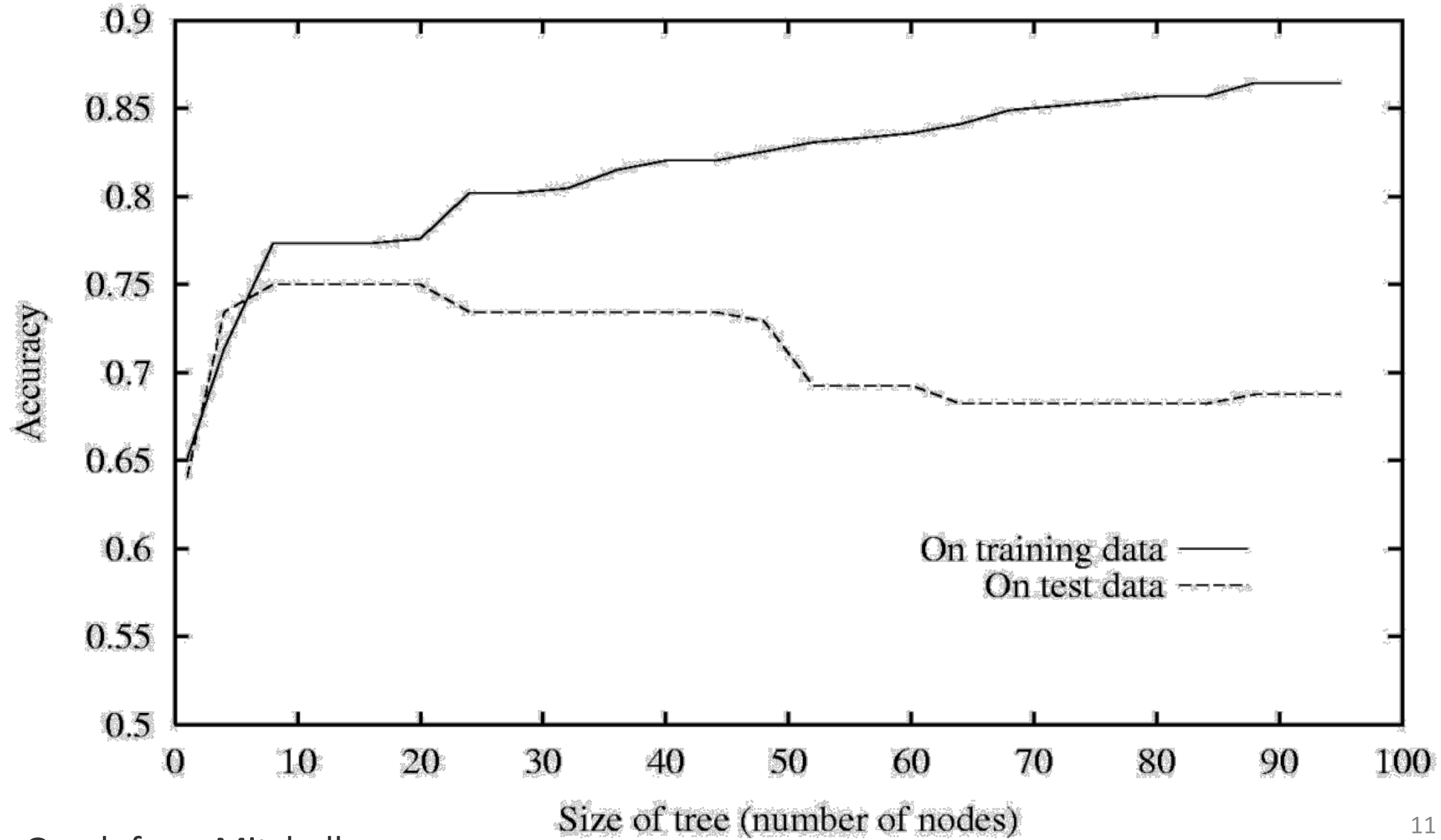
- Noisy example:

(Outlook = Sunny, Temp = Hot, Humidity = Normal, Wind = Strong, NO)



*may fit noise or other coincidental regularities*

## Decision trees *will* overfit



Graph from Mitchell

## Avoiding overfitting in decision trees

- **Occam's Razor**
  - Favor simpler (in this case, shorter) hypotheses
  - Fewer shorter trees, less likely to fit better by coincidence
- **Static**: Fix the depth of the tree
  - Only allow trees of size K
    - Tune K using held-out validation set
    - ***Decision stump*** = a decision tree with only one level
- **Dynamic**: optimize while growing the tree
  - Grow tree on training data
  - Check performance on held-out data after adding a new node

## Avoiding overfitting in decision trees

- **Occam's Razor**
    - Favor simpler (in this case, shorter) hypotheses
    - Fewer shorter trees, less likely to fit better by coincidence

- **Post Pruning:**
- While accuracy on validation set decreases. Bottom up:
    - For each non leaf node:
        - Replace sub-tree under node by a majority vote
        - Test accuracy on validation set

## Decision Trees as Features

- When learning over a large number of features, learning decision trees is difficult and the resulting tree may be very large
- **Instead of pruning you can try:**
  - learn small decision trees, with limited depth.
  - Then, learn another function over these trees

- For example, Linear combination of decision stumps

## Model Selection

- All the algorithms that we saw (and that we will see) can be parameterized, to help control their behavior
  - *I.e., control the properties of the type of models they will produce.*
- These can capture preferences that we have about the classifiers
  - *Smaller trees, preference towards one type of error, etc.*
- In some cases, we just want the settings that get us the **best** classifier
  - *But, well.. what is **best**? and how do we know that we found it? (what can we trust?)*

## Model Selection

- *We can think about selecting the best model as a* <u>secondary learning problem</u>
  - Split the data into: (1) **train** set (2) **test** set
  - Split the **train** data into: (1) **train** set (2) **validation** set
  - **<u>Training</u>**: train m models, with different parameters
    - E.g., Different ways to control the size of the tree
  - **<u>Validation</u>:** estimate the prediction error for each model
  - **<u>Testing</u>**: use the model with the least validation error
- ***The secondary learning problem***:
  - New hypothesis space: m different hypothesis to chose from
  - Pick the one that minimizes validation error

# Model Selection

- ***What are the algorithm hyper-parameters?***
  - ***Decision trees:***
    - Depth of the tree
    - Pruning strategy
    - Pruning decision
    - Attribute selection heuristic
    - *Other choices?*
  - ***KNN***
    - Value of K
    - Similarity metric
- ***Every learning algorithm we will cover has a set of hyper-parameters***

## K-Fold Cross validation

- *This approach is "risky"* ***(why?)***
    - Random selection of training examples for train/test/validation
        - *You could be very unlucky*
    - *Your validation data may not reflect the same distribution as your test data*
    - Optimizing on the validation data will lead to worse performance

# K-Fold Cross validation

- You could get really unlucky..
  - *..but that's not likely to happen too frequently!*
- K-Fold cross validation: repeat the process K times, and average the results.
- Randomly partition the data into K equal-size subsets $S_1..S_k$
  - For i=1… K
    - Train a hypothesis on $S_1..S_{i-1}\ S_{i+1}..S_k$
    - Evaluate on $S_i$ ($Err(S_i)$)
  - Return ($\Sigma_i\ Err(S_i)/K$)

## K-fold cross validation

- Looks like a good way to estimate the true error
  - Run K independent experiments, each sampling a different dataset from P(X,Y)
  - Compute the average error of a learning algorithm over the K datasets
  - For large K, converge to the true error

- **What can go wrong?**

## Evaluating the Learned Hypothesis

- What is the error of h?
  - *How do I know my classifier is good enough?*
    - For example, $Err_p(h) < 0.1$
    - Is the ***training error*** a good estimate of the error?
    - ***Testing error***?
- *What is the error we are "really" after?*

## Learning: *a few formal definitions*

**Intuitions:**

- Learning algorithm gets a training set (*batch* mode)
- Performance should be measured on unseen test data
- Learning works if there is a strong relationship between the *training* and *test* data
- We can trust our evaluation if there is a strong relationship between the *test* data and the "real world"

***Let's formalize these intuitions!***

## Loss functions

- To formalize performance let's define a loss function:

$$loss(y, \hat{y})$$

  - Where $\hat{y}$ is the gold label
- The loss function measures the error on a single instance
  - Specific definition depends on the learning task

***Regression***

$$loss(y, \hat{y}) = (y - \hat{y})^2$$

***Binary classification***

$$loss(y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ 1 & otherwise \end{cases}$$

## Formalizing the learning process

- We assume the data is sampled from an *unknown* distribution D = P(x,y)
  - D assigns high probability to "reasonable" (x,y) pairs
  - Unreasonable (x,y) pairs:
    - *x is an unusual input (*Purdue + Hot days + January*)*
    - *Y an unlikely label for x (*Purdue + January ➔ Swimming*)*

- Performance is defined with respect to:
  - **Loss function**: What "*matters"* in the learning task
  - **Data generating distribution** (D)

# Learning Definitions

**Learning** : representing $P(x, y) = P(y|x)P(x)$

- *P(x) is the statistical model of the "world"*
  - Producing examples according to a distribution (unknown)

- *P(y|x) statistical model of the "teacher"*
  - Generalizing the concept of the teacher
    - Target function– very peaked distribution over y for a given x

$$P(sick|Temprature > 95) = 1$$

    - Now, think about a probabilistic process labeling the data

$$P(sick|Temprature > 95) = 0.8$$

## Learning Definitions

$$P(S) = P((x_1, y_1), \ldots, (x_n, y_n))$$

- Given S, *a dataset*, examples in S are assumed to be *Independent and identically distributed* (iid):

$$P(S) = \prod_i P(x_i, y_i)$$

  - *Independently drawn from the same distribution*
    - When are examples not independent?
    - When are examples not identically distributed?
  - The key assumption behind machine learning algorithms and their theoretic analysis.

## Formalizing the learning process

- Learning goal: Minimize expected loss over D w.r.t *l*

$$\epsilon \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(y,f(x))] = \sum_{(x,y)} \mathcal{D}(x,y)\ell(y,f(x)) \quad )$$

*We do not know D in advance!*

- *But, we have access to training data sampled from D*
- *Instead, compute Empirical loss*
  - ***What is the difference?***

$$\hat{\epsilon} \triangleq \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(x_n))$$

- **Learning**: *find a function with low* ***expected*** *loss over D w.r.t l.*
- ***This is where inductive bias comes in!***

# Evaluating the Learned Hypothesis

- What is a "good" error for a learned hypothesis?
  - ***How do I know my classifier is good enough?***
    - For example, $Err_p(h) < 0.1$
    - Is the training error a good estimate of the error?
    - Testing error? *Is it a good approximation for the* ***true error****?*
      - On average, that's the true performance.
      - We have to account to variability!
      - i.e., different choices of testing sets could lead to different results!

  - **How can we account for the test error variability ?**
    - Run multiple times, and average results
    - Closed form solution

# Evaluating the Learned Hypothesis

- **How are errors generated?**
  - *S (test set) randomly draws an example, and get a label from the classifier*
  - *P(first example is wrong)?*
    - ***True error***
  - *P(second example is wrong)?*
    - ***True error***
  - Draws are independent

  ***This should sound really familiar!***
- N “coin flips” (testing examples)
- What is the probability of K errors?

  ➔ ***Number of errors is binomially distributed!***

## Quick Detour: *Binomial Distribution*

$$P(X = x|p, n) = \frac{n!}{x!(n-x)!}\, p^x (1-p)^{n-x}$$

***x*** number of errors we observe
***p*** is the probability of errors
***n*** is the number of test examples

## Evaluating the Learned Hypothesis

- Our algorithm produced a model (h)
  - 10 errors out of 500 test examples
  - Is that significant evidence for the $\mathrm{Err}_P(h)<$**0.1?**
- Significance test:
  - **Null hypothesis**: p ≥ **0.1**
  - *what is the probability that we see at most 10 errors out of 500?*

$$P(x \leq 10|p = 0.1, n = 500) \approx 10^{-12} \leq 0.05$$

  - *If the null hypothesis was true, the observed performance is unlikely*
  - ***We can reject the null hypothesis!***

## Comparing learning algorithms

- Given two modeling/algorithmic choices that would result in different models. **Can we say if algorithm 1 is better than algorithm 2?**
  - KNN vs. DT
  - Hyperparameter choices: K=5 vs. K=10, tree depth, etc.
- Say, error (algorithm 1) = error(algorithm 2) - 0.1
- **Can we determine if the performance difference happened by change (i.e., specific data sample) or not (i.e., likely for any data sample)?**

# Comparing learning algorithms

- Null hypothesis: Alg 1 is no better than Alg 2.
- We would like to find the probability that the null hypothesis is false.
    - Common values: 90%, 95%, 99.5% (depending on community)
- T-test: assume that difference between the two error distributions has mean zero, and that the data is normally distributed
    - Binomial distribution can be approximated using a Gaussian for large N
- Assume N examples, `a1,…,an` and `b1,…,bn` error on the examples by algorithm 1/2 respectively

## Comparing learning algorithms

- The mean of the error: μ1 (alg 1)  μ2 (alg 2)
- Center the data: $\hat{a} = \boldsymbol{a} - \mu_a$ and $\hat{b} = \boldsymbol{b} - \mu_b$.
- Compute the t-statistic $t = (\mu_a - \mu_b)\sqrt{\frac{N(N-1)}{\sum_n (\hat{a}_n - \hat{b}_n)^2}}$

| $t$ | significance |
|---|---|
| $\geq 1.28$ | 90.0% |
| $\geq 1.64$ | 95.0% |
| $\geq 1.96$ | 97.5% |
| $\geq 2.58$ | 99.5% |

## Precision and Recall

- Given a dataset, we train a classifier that gets 99% accuracy
- **Did we do a good job?**
- Build a classifier for brain tumor:
  - 99.9% of brain scans do not show signs of tumor
  - ***Did we do a good job?***
- By simply saying "NO" to all examples we reduce the error by a factor of 10!
  - ***Clearly Accuracy is not the best way to evaluate the learning system when the data is heavily skewed!***
- **Intuition**: we need a measure that captures the class we care about! (rare)

# Precision and Recall

- The learner can make two kinds of mistakes:
  - False Positive
  - False Negative

| | True Label: 1 | True Label: 0 |
|---|---|---|
| **Predicted: 1** | True Positive | False Positive |
| **Predicted: 0** | False Negative | True Negative |

- **Precision:**
- *"when we predicted the rare class, how often are we right?"*

$$\frac{\text{True Pos}}{\text{Predicted Pos}} = \frac{\text{True Pos}}{\text{True Pos} + \text{False Pos}}$$

- **Recall**
- "Out of all the instances of the rare class, how many did we catch?"

$$\frac{\text{True Pos}}{\text{Actual Pos}} = \frac{\text{True Pos}}{\text{True Pos} + \text{False Neg}}$$

# F-Score

- **Precision and Recall give us two reference points to compare learning performance**

| | Precision | Recall |
|---|---|---|
| **Algorithm 1** | 0.5 | 0.4 |
| **Algorithm 2** | 0.7 | 0.1 |
| **Algorithm 3** | 0.02 | 1 |

- *Which algorithm is better?*

***We need a single score***

- Option 1: Average $\frac{P+R}{2}$
- Option 2: F-Score $2\frac{PR}{P+R}$

**Properties of f-score:**

- *Ranges between 0-1*
- *Prefers precision and recall with similar values*

## Ablation Study

- Making predictions often relies on many different attributes of the input object
- Let's consider email phishing detection:
  - Baseline system: lexical features
    - *"The excellent prince of Mars wants to give you a g1ft"*
    - *Accuracy : 70%*
  - *Complex system:*
    - *Lexical features, sender, email headers, servers, images, dictionaries of suspicious terms, spelling mistakes*
    - *Accuracy: 85%*
- *What aspects are responsible for the improvement?*
  - ***Run an ablation study***

## Ablation Study

- Remove one feature and train+test the model
- **Other things to check:**
  - *What is the influence of features choices on **precision/recall**?*
  - Similar mistakes or different mistakes?

| **Features** | **ACC** |
|---|---|
| Lexical features | 66 |
| Sender | 79 |
| Email headers | 83 |
| Servers | 80 |
| Images | 85 |
| Suspicious terms | 82 |
| Baseline (lexical features) | 70 |

**Error Analysis:**

You can also look at the **type** of mistakes your model is making:

*Some Phishing scams could be easier to detect than others.*

- *Check the influence of different features by mistake types*

## Error Analysis

- Identify the root cause of the mistakes your algorithm makes
  - **Aspects not captured by your features**

***"We wish you a happy new year"***

  - Noisy feature extraction
    - E.g., "Suspicious terms" detector is not comprehensive enough
  - Many other reasons: noisy labels,..

# Online Learning and Linear models

We will introduce a new to quantify performance,
by measuring the number of mistake an algorithm makes.

We'll show that depending on the hypothesis class we choose,
we can use algorithms that have better mistake bounds

Specifically, we'll compare learning disjunctions using Boolean functions and linear functions, and analyze their behavior.

## Quantifying Performance

- We want to be able to say something rigorous about the performance of our **learning algorithm**.
  - *Several ways of doing it*

- We will concentrate on discussing the number of examples one needs to **see** before we can say that our learned hypothesis is good.

## Learning Conjunctions

- There is a hidden (monotone) conjunction the learner (you) is to learn

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- High dimensional space, but concept only uses a few
- *How many examples are needed to learn it ?*
  - **Protocol I**: The learner proposes instances as queries to the teacher ("active learning")
  - **Protocol II**: The teacher (*who knows f*) provides training examples ("hands-on teacher")
  - **Protocol III:** *Some random source (e.g., Nature) provides training examples; the Teacher (Nature) provides the labels (f(x))*
    - *Recall the definition of a Data generating distribution*

## Learning Conjunctions

- **Protocol I: *Learner proposes instances as queries to the teacher***
    - **Queries**: Students picks an instance, teacher labels it (<(1,0,1),1>)
    - *What are the question we ask?*

- Since we know we are after a **monotone** conjunction:
    - Is $x_{100}$ in? <(1,1,1…,1,0), ?> f(x)=0 (conclusion: Yes)
    - Is $x_{99}$ in? <(1,1,…1,0,1), ?> f(x)=1 (conclusion: No)
    - Is $x_1$ in ? <(0,1,…1,1,1), ?> f(x)=1 (conclusion: No)

- *How many queries are needed for learning perfectly?*
    - A straight forward algorithm requires n=100 queries
    - It will produce the hidden conjunction (exactly).

$$h = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

# Learning Conjunctions

- **Protocol II**: *The teacher (who knows f) provides training examples*
  - *Tell student it's a monotone conjunction*
  - ***How do you start?***
    - <(0,1,1,1,1,0,…,0,1), 1> *(We learned a superset of the good variables)*
  - To show you that all these variables are required
    - <(0,0,1,1,1,0,…,0,1), 0> need $x_2$
    - <(0,1,0,1,1,0,…,0,1), 0> need $x_3$
    - …..
    - <(0,1,1,1,1,0,…,0,0), 0> need $x_{100}$

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- A straight forward algorithm requires k = 6 examples to produce the hidden conjunction (exactly).

# Learning Conjunctions

- **Protocol III**: Some random source (e.g., Nature) provides training examples
- Teacher (Nature) provides the labels (f(x))
    - <(1,1,1,1,1,1,…,1,1), 1>
    - <(1,1,1,0,0,0,…,0,0), 0>
    - <(1,1,1,1,1,0,...0,1,1), 1>
    - <(1,0,1,1,1,0,...0,1,1), 0>
    - <(1,1,1,1,1,0,...0,0,1), 1>
    - <(1,0,1,0,0,0,...0,1,1), 0>
    - <(1,1,1,1,1,1,…,0,1), 1>
    - <(0,1,0,1,0,0,...0,1,1), 0>

## Learning Conjunctions

- Protocol III: Some random source (e.g., Nature) provides training examples
  - **Algorithm: Elimination**
  - Start with the set of all literals as candidates
  - ***Eliminate a literal that is not active (0) in a positive example***

$$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \ldots \wedge x_{100}$$

# Learning Conjunctions

- **Protocol III**: Some random source (e.g., Nature) provides training examples
  - **Algorithm: Elimination**
  - Start with the set of all literals as candidates
  - ***Eliminate a literal that is not active (0) in a positive example***

$$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \ldots \wedge x_{100}$$

<(1,1,1,1,1,1,…,1,1), 1>
<(1,1,1,0,0,0,…,0,0), 0> ← learned nothing
<(1,1,1,1,1,0,...0,1,1), 1> ← **Eliminate literals**
<(1,0,1,1,1,0,...0,1,1), 0> ← learned nothing
<(1,1,1,1,1,0,...0,0,1), 1> ← **Eliminate literals**
<(1,0,1,0,0,0,...0,1,1), 0>
<(1,1,1,1,1,1,…,0,1), 1>
<(0,1,0,1,0,0,...0,1,1), 0>

**<u>Is that good ?</u>**
*We only learned an approximation to the true concept*!

**Final Hypothesis:**
(*not the target!*)

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

## Is it GOOD? *Two Directions*

- Follow a probabilistic intuition (PAC)
  - Considers the *Data generating distribution*
  - Never saw $x_1$=0 in positive examples, maybe we never will?
  - **if** we will, *it will be with small probability*, so the concepts we learn may be pretty good
  - Good: in terms of performance on **future** data
- Mistake Driven Learning Algorithms
  - Learn from a stream of examples, update only when you make a mistake
  - Good: in terms of **how many mistakes** you make before you stop, happy with your hypothesis**.**

## Online Learning

- **Online learning**
  - Learn from one example at a time (unlike batch)
  - Update current hypothesis based on that example
- **Mistake (*error*) driven learning**
  - **Update only on mistakes**
  - *Not all online learning algorithms are mistake driven*
- **Discuss two learning algorithms for linear functions**
  - Perceptron and Winnow

## Online Learning: *Motivation*

- Consider a learning problem in a very high dimensional space

$$\{x_1, x_2, x_3, \ldots\ldots, x_{1000000}\}$$

- Assume the function space is very *sparse*
  - every function of interest depends on a small number of attributes.)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge .x_{100}$$

*"I don't know {**whether**, weather} to laugh or cry"*

- Can we develop an algorithm that:
  - Depends only **weakly on the space dimensionality**
  - Mostly on the **number of relevant attributes** ?
- How should we represent the hypothesis?
  - Target function is a conjunction, but we can learn ***others***

## Online Learning: *Motivation*

- **Simple + Intuitive + Broadly applicable model**
  - Robot in an assembly line, language learning,…
- **Realistic settings**: very large data sets, when the data cannot fit memory – Streaming data
- **Evaluation**: We will try to make the smallest number of mistakes in the long run (**mistake bounds**)

# Online Learning: *Evaluation*

- Model:
  - Instance space: X (*dimensionality – n*)
  - Target: f: X →{0,1}, f ∊ C, concept class (***parameterized by n***)

- Protocol:
  - learner is given x ∊ X
  - learner predicts h(x), and is then given f(x) (*feedback*)

- Performance*: learner makes a mistake when h(x) ≠ f(x)*
  - # mistakes algorithm A makes on sequence S of examples, for the target function f

$$M_A(C) = \max_{f \in C, S} M_A(f, S)$$

- A is a mistake bound algorithm for the concept class C, **if** $M_A$(c) is polynomial in n, the complexity parameter of the target concept.
  - Worse case model – No notion of distribution

## **Question:**

*Is it a realistic analysis?*

*Can we bound the number of mistakes?*

## Generic Mistake Bound Algorithms

- Let C be a concept class. Learn f ∈ C
- **CON**:
  - In the i-th stage of the algorithm:
    - $C_i$ all concepts in C consistent with all (*i-1*) previously seen examples
    - Choose randomly f∈ $C_i$ and use to predict the next example
- Clearly, $C_{i+1} \subseteq C_i$ and, if a mistake is made on the ith example, then $|C_{i+1}| < |C_i|$ so progress is made.
- *The CON algorithm makes at most |C|-1 mistakes*
- **Can we do better ?**

## The **Halving** Algorithm

- Let C be a concept class. Learn f ∈ C
- **Halving**:
- In the i-th stage of the algorithm:
    - $C_i$ all concepts in C consistent with all (i-1) previously seen examples
- Given an example $e_i$ consider the value $f_j(e_i)$ for all $f_j \in C_i$ and **predict by majority**.
- Predict 1 if $|\{f_j \in C_i; f_j(e_i) = 0\}| < |\{f_j \in C_i; f_j(e_i) = 1\}|$
- Clearly $C_{i+1} \subseteq C_i$ and if a mistake is made in the i-th example,

then $|C_{i+1}| < \frac{1}{2}|C_i|$

- The Halving algorithm makes at most log(|C|) mistakes

## The Halving Algorithm

- **Hard to compute** (why?)
- In some cases Halving is optimal (C - class of all Boolean functions)

- We discuss these algorithms since they give us an idea of the ***theoretical bound for mistake driven learning***
  - *Can we find efficient algorithms that are close to the bounds?*

# Learning Disjunctions

- There is a hidden disjunction the learner is to learn

$$f = x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_{100}$$

- The number of disjunctions: $3^n$
  - log(|C|) = n
- *Can you find a mistake bound algorithm for disjunctions?*
  - The elimination algorithm makes n mistakes
    - *How would you adapt it to the disjunctive case?*
  - The Halving Algorithm makes n mistakes as well

- Great news!
  - We have a mistake bound algorithm for disjunctions!

# Learning K-Disjunctions

- **k-disjunctions:**
  - Assume that only k<<n attributes occur in the disjunction
- A reasonable scenario:
  - A spam email depends on a subset of words:

    "drugs" OR "credit" OR "pill"
- ***What is n and what is k in this example?***
- The number of k-disjunctions:
  - How many mistakes: (1) elimination (2) Halving
  - Can we learn **efficiently** with this number of mistakes ?

## The Importance of Representation

- **Assume that you want to learn disjunctions. Should your hypothesis space be the class of disjunctions?**

***Theorem [Haussler 1988]:*** *Given a sample on n attributes consistent with a disjunctive concept, it is NP-hard to find a pure disjunctive hypothesis that is both consistent with the sample and has the minimum number of attributes*

- Intuition: Reduction to minimum set cover problem.
- ➔ Cannot learn the concept efficiently **as a disjunction.**
- But, we will see that we can do that, if we are willing to learn the concept as a **Linear Threshold function**.
  - In a more expressive class, the search for a good hypothesis sometimes becomes combinatorially easier.

## Linear Models

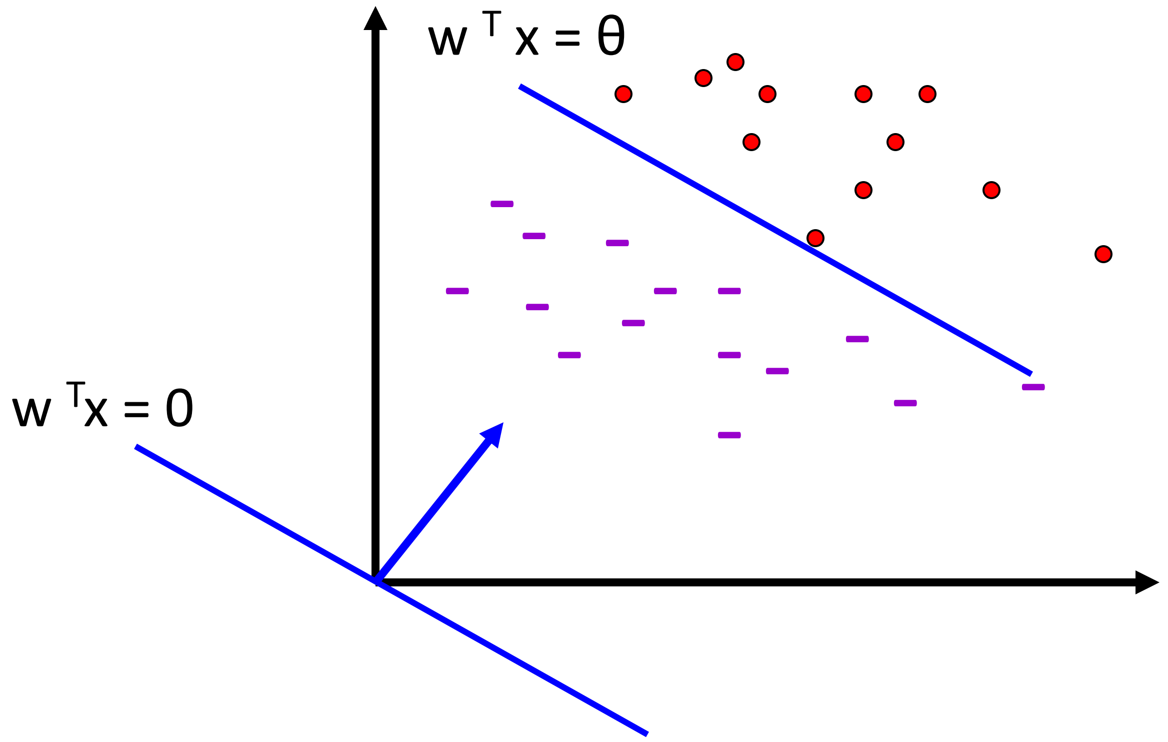
- *Input is a n-dimensional vector* $(\mathbf{x})$
- *Output label* $y \in \{-1,1\}$

- *Linear threshold functions classify examples* $(\mathbf{x})$ *using a parameter vector* $(\mathbf{w})$ *and a real number* $(b)$

- $y = \mathrm{sign}(\mathbf{w}^T\mathbf{x}+b) = \mathrm{sign}(b + \sum_i w_i x_i)$
  - $\mathbf{w}^T\mathbf{x}+b \geq 0 \rightarrow y=1$
  - $\mathbf{w}^T\mathbf{x}+b < 0 \rightarrow y=-1$

## Linear Model

A linear classifier represents a hyperplane (line in 2D), that separates the space into two half spaces.



# Expressivity of Linear Functions

- **What can Linear Functions Represent?**
  - *Linear classifiers are an expressive hypothesis class*
- Many Boolean functions can be compactly represented using linear functions
  - We refer to it as *linear separablity*
- Some Boolean functions are not linearly separable

## Expressivity of Linear Functions

- **Conjunctions** and **Disjunctions**
  - $y=x_1 \land x_2 \land x_3$ is equivalent to $y=1$ *if* $x_1 + x_2+x_3 \geq 3$

- Question:
  - How about disjunctions? $y=x_1 \lor x_2 \lor x_3$
  - $y=x_1 \lor x_2 \lor x_3$ is equivalent to $y=1$ if $x_1 + x_2+x_3 \geq 1$
  - How can you represent **negations**? $y=x_1 \land x_2 \land \neg x_3$
  - $y=x_1 \land x_2 \land \neg x_3$ is equivalent to $y= x_1 + x_2-x_3 \geq 2$

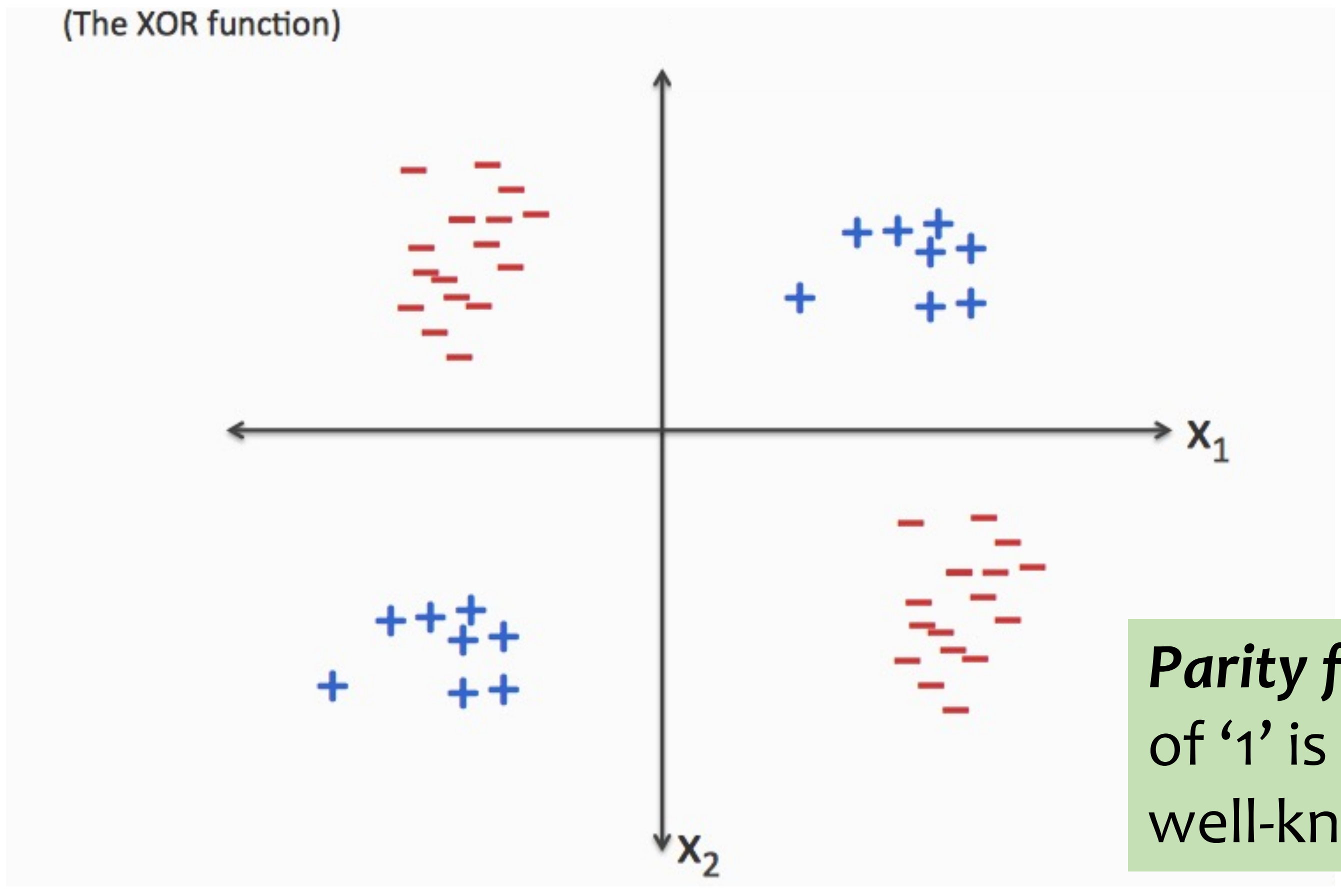
# Expressivity of Linear Functions

- M-of-N rule
  - Pick N variables (N can be all of the variables or a subset)
  - Y=1 if at least M are active

- How can you represent 3 of $\{x_1,x_2,x_3,x_4,x_5\}$
  - As a Boolean function?
  - As a Linear function?
    - $x1+x_2+x_3+x_4+x_5 \geq 3$

## Expressivity of Linear Functions

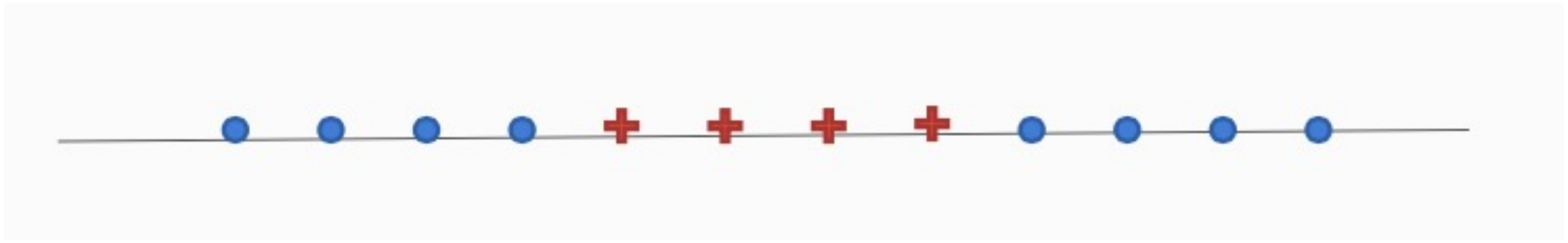
- ***Not***



***Parity function*** (number of ‘1’ is even) is another well-known example

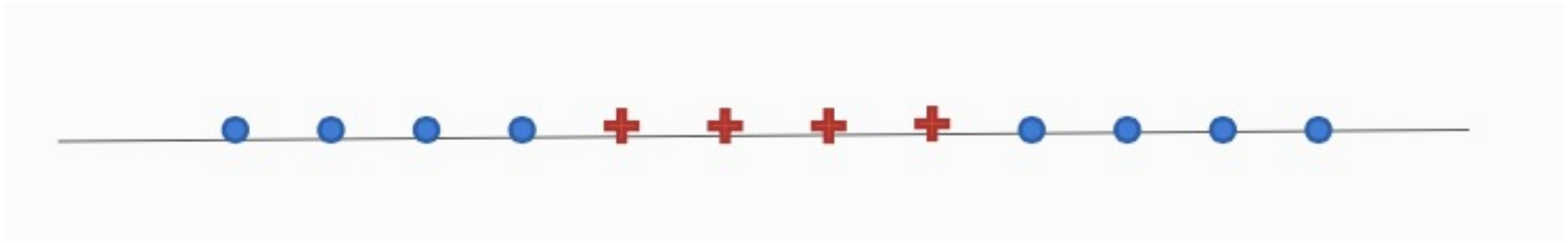
## Expressivity of Linear Functions

- Non-Linear functions can be made linear
  - *We can change the representation of the input instances to represent functions that are not linear in the original space.*



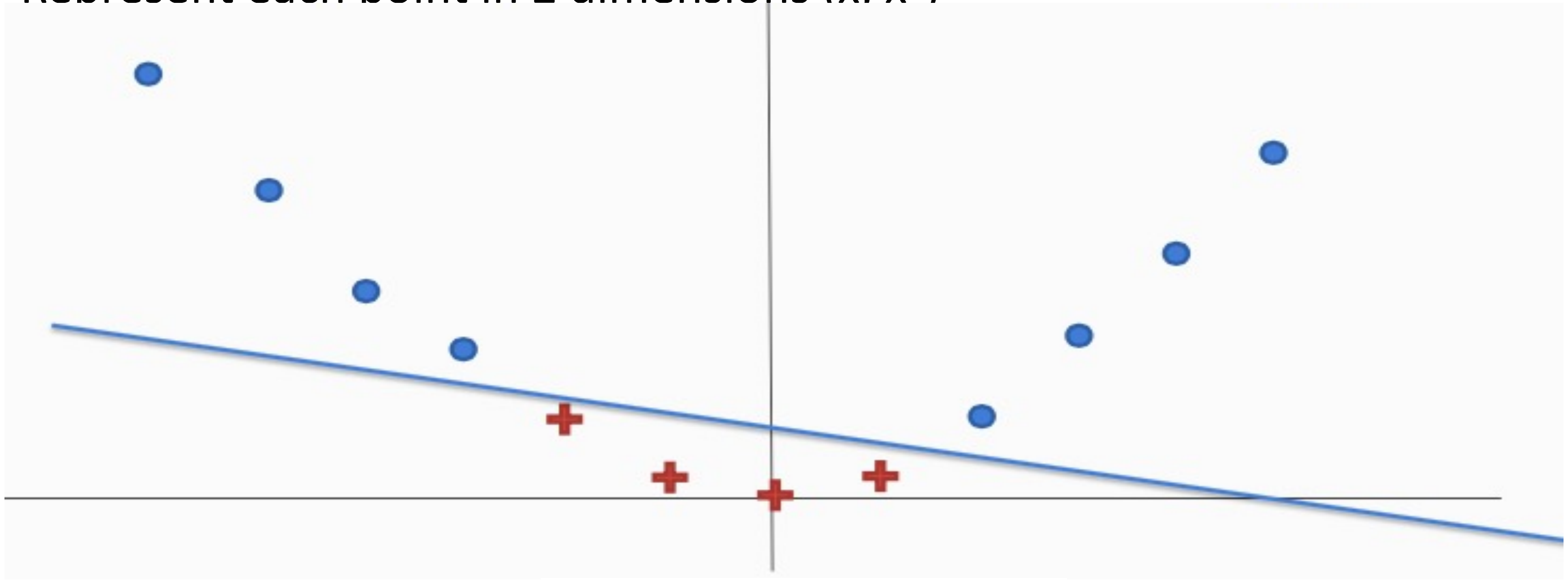
# Expressivity of Linear Functions

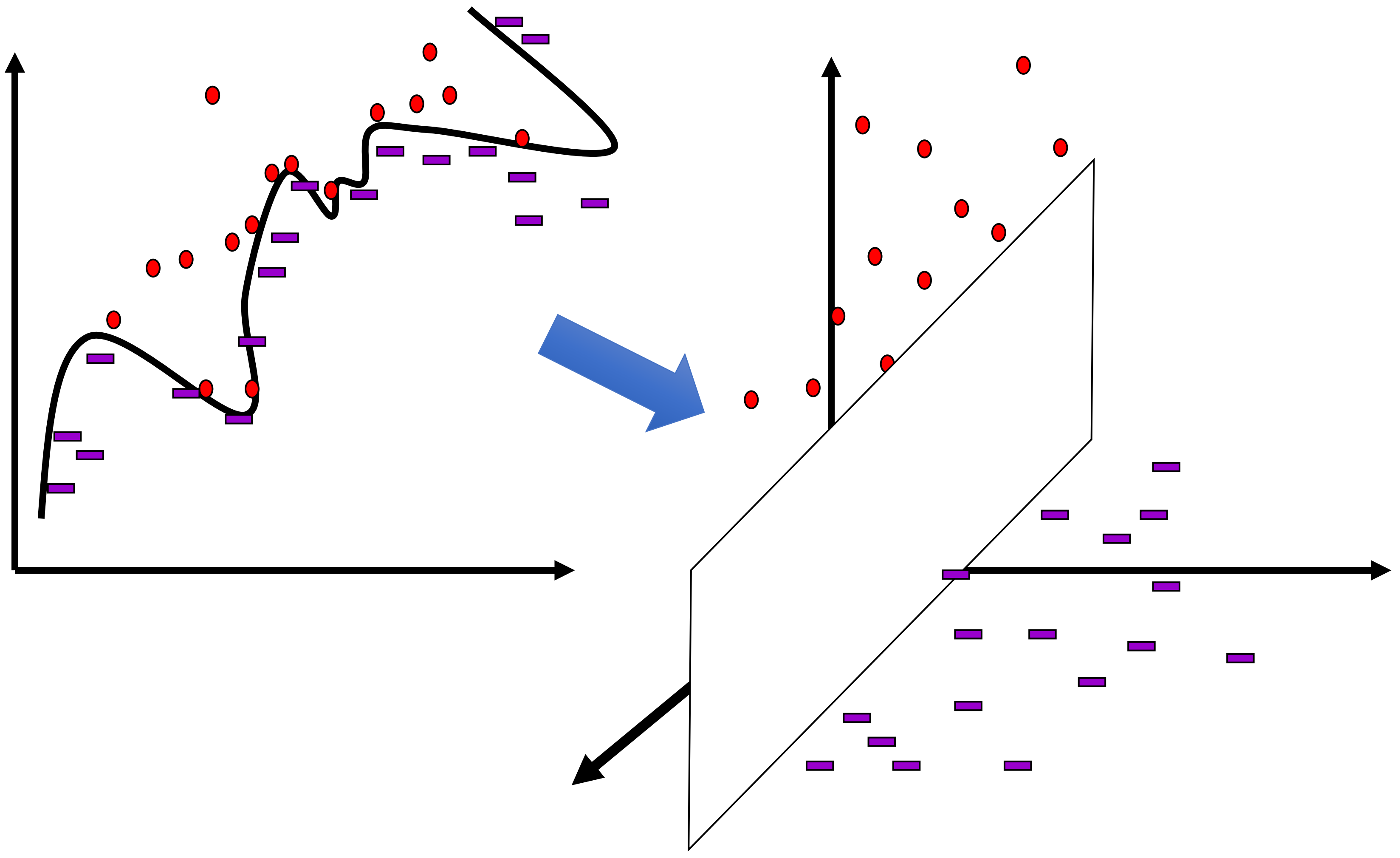
- **This function is not linearly separable**
- **Solution**: *use feature conjunctions*
  - Represent each point in 2 dimensions (x, x$^2$)



# Expressivity of Linear Functions

- This function is not linearly separable
- Solution: use feature conjunctions
  - Represent each point in 2 dimensions (x, $x^2$)



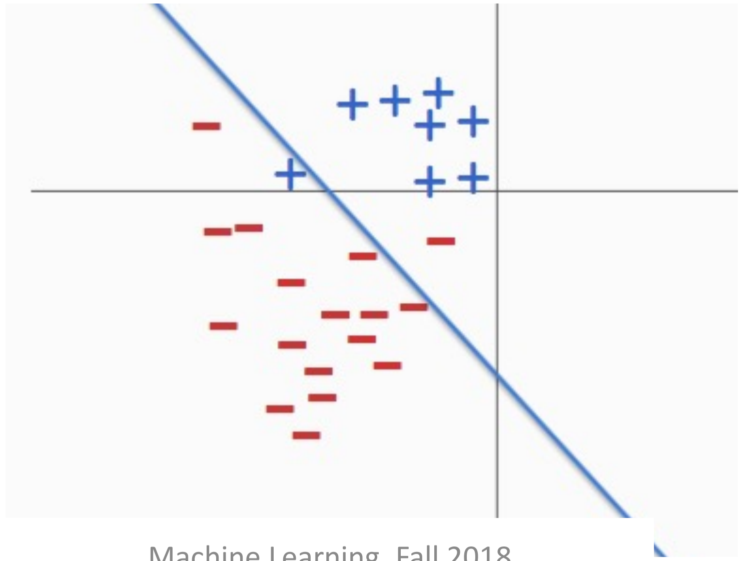


## Question

*Which feature transformation would you use to make 2d* ***XOR*** *into a linearly separable function?*

## Almost Linearly Separable Data

- In many cases the data is “almost” linearly separable
  - We can think about these examples as “noise”
  - *How much noise should we allow?*
  - *When should we change the expressivity of the learned function to account for it?*



## Note on the Bias Term

- *To simplify notation we will include the bias term as an "always on" feature*
  - Instead of $b+w^T x$ we concatenate b to x and set its value to 1
  - Increase the space dimension by 1
  - We can learn the right bias by tuning the value of that feature

***Even if we don't explicitly mention it, remember that it is still there!***

## Learning Linear Functions

- We will look into several ways of learning linear functions
  - Both batch and online

- We will start with two *mistake driven* algorithms for linear functions
  - Winnow and Perceptron
  - Fun Fact*:*
    - *Perceptron is one of the first learning algorithms!*

## Winnow

Only update the active features!

$$\textbf{Initialize}: \theta = n;\ \mathbf{w}_i = 1$$
$$\textbf{Prediction is 1 iff}\quad \mathbf{w} \bullet \mathbf{x} \geq \theta$$
$$\textbf{If no mistake: do nothing}$$
$$\textbf{If } f(x) = 1 \textbf{ but } \mathbf{w} \bullet \mathbf{x} < \theta,\quad \mathbf{w}_i \leftarrow 2\mathbf{w}_i \quad (\text{if } x_i = 1)\ (\text{promotion})$$
$$\textbf{If } f(x) = 0 \textbf{ but } \mathbf{w} \bullet \mathbf{x} \geq \theta,\quad \mathbf{w}_i \leftarrow \mathbf{w}_i/2 \quad (\text{if } x_i = 1)\ (\text{demotion})$$

- The Winnow Algorithm learns Linear Threshold Functions.
    - We will prove a mistake bound for learning K-disjunctions with Winnow

## Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

$\text{Initialize}: \quad \theta = 1024; w = (1,1,\ldots,1)$

Initialize weight to 1, threshold to *n*

## Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

Initialize : $\theta = 1024; w = (1,1,\ldots,1)$

$<(1,1,\ldots,1),+>$ $w \bullet x \geq \theta$ $w = (1,1,\ldots,1)$ ok

$<(0,0,\ldots,0),->$ $w \bullet x < \theta$ $w = (1,1,\ldots.,1)$ ok

So far no update..

# Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

Initialize: $\theta = 1024; w = (1,1,\ldots,1)$

| | | | |
|---|---|---|---|
| $<(1,1,\ldots,1),+>$ | $w \bullet x \geq \theta$ | $w = (1,1,\ldots,1)$ | ok |
| $<(0\ ,0,\ldots,0),->$ | $w \bullet x < \theta$ | $w = (1,1,\ldots.,1)$ | ok |
| $<(0,0,111,,,,0),->$ | $w \bullet x < \theta$ | $w = (1,1,\ldots.,1)$ | ok |
| $<(1,0,0,\ldots,0),+>$ | $w \bullet x < \theta$ | $w = (2,1,\ldots.,1)$ | mistake |

**Mistake!**
Dot product is **positive**, **but below the threshold**
(only x_1 is active..)

# Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

Initialize : $\theta = 1024; w = (1,1,...,1)$

| | | | |
|---|---|---|---|
| $<(1,1,...,1),+>$ | $w \bullet x \geq \theta$ | $w = (1,1,...,1)$ | ok |
| $<(0\ ,0,...,0),->$ | $w \bullet x < \theta$ | $w = (1,1,....,1)$ | ok |
| $<(0,0,111,,,,0),->$ | $w \bullet x < \theta$ | $w = (1,1,....,1)$ | ok |
| $<(1,0,0,....,0),+>$ | $w \bullet x < \theta$ | $w = (2,1,....,1)$ | mistake |
| $<(1,0,1,1,0..,0),+>$ | $w \bullet x < \theta$ | $w = (4,1,2,2...,1)$ | mistake |

Many variables are active now, but still not enough to go over the threshold

# Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

Initialize : $\theta = 1024;\ w = (1,1,\ldots,1)$

$<(1,1,\ldots,1),+>$ $w \bullet x \geq \theta$ $w = (1,1,\ldots,1)$ ok

$<(0\ ,0,\ldots,0),->$ $w \bullet x < \theta$ $w = (1,1,\ldots.,1)$ ok

$<(0,0,111,,,,0),->$ $w \bullet x < \theta$ $w = (1,1,\ldots.,1)$ ok

$<(1,0,0,\ldots.,0),+>$ $w \bullet x < \theta$ $w = (2,1,\ldots.,1)$ mistake

$<(1,0,1,1,0..,0),+>$ $w \bullet x < \theta$ $w = (4,1,2,2\ldots,1)$ mistake

$<(1,0,1,0,0..,1),+>$ $w \bullet x < \theta$ $w = (8,1,4,2\ldots,2)$ mistake

…………………… log(n/2) (for each good variable)

$w = (512,1,256,256,\ldots.,256)$

**After log(n) mistakes for each "good" variable we stop making mistakes on positives**

$<(1,0,1,0\ldots,1),+>$ $w \bullet x \geq \theta$ $w = (512,1,256,256,\ldots.,256)$ ok

$<(0,0,1,0.111..,0),->$ $w \bullet x \geq \theta$ $w = (512,1,0,..0,\ldots,256)$ mistake **(elimination version)**

………………………

$w = (1024,1024,0,0,0,1,32,\ldots,1024,1024)$ **(final hypothesis)**

# Winnow Example

$f = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$

Initialize : $\theta = 1024; w = (1,1,\ldots,1)$

$<(1,1,\ldots,1),+>$ $w \bullet x \geq \theta$ $w = (1,1,\ldots,1)$ ok

$<(0\ ,0,\ldots,0),->$ $w \bullet x < \theta$ $w = (1,1,\ldots.,1)$ ok

$<(0,0,111,,,,0),->$ $w \bullet x < \theta$ $w = (1,1,\ldots.,1)$ ok

$<(1,0,0,\ldots,0),+>$ $w \bullet x < \theta$ $w = (2,1,\ldots.,1)$ mistake

$<(1,0,1,1,0..,0),+>$ $w \bullet x < \theta$ $w = (4,1,2,2\ldots,1)$ mistake

$<(1,0,1,0,0..,1),+>$ $w \bullet x < \theta$ $w = (8,1,4,2\ldots,2)$ mistake

...................... log(n/2) (for each good variable)

$w = (512,1,256,256,\ldots..,256)$

**And.. Mistakes on negatives (eliminations) don't effect the weights of "good" variables (why?)**

$<(1,0,1,0\ldots,1),+>$ $w \bullet x \geq \theta$ $w = (512,1,256,256,\ldots..,256)$ ok

$<(0,0,1,0.111..,0),->$ $w \bullet x \geq \theta$ $w = (512,1,0,..0,\ldots,256)$ mistake **(elimination version)**

.........................

$w = (1024,1024,0,0,0,1,32,\ldots,1024,1024)$ **(final hypothesis)**

# Winnow – Mistake Bound

**Claim**: Winnow makes O(k log n) mistakes on k-disjunctions

```
Initialize : $\theta = n$; $w_i = 1$
Prediction is 1 iff $w \bullet x \geq \theta$
If no mistake : do nothing
If $f(x) = 1$ but $w \bullet x < \theta$ , $w_i \leftarrow 2w_i$ (if $x_i = 1$) (promotion)
If $f(x) = 0$ but $w \bullet x \geq \theta$ , $w_i \leftarrow w_i/2$ (if $x_i = 1$) (demotion)
```

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

## 1. <u>u < k log(n)</u>

A weight that corresponds to a good variable is only promoted

When these weights get to n there will be no more mistakes on positives

# Winnow – Mistake Bound

**Claim**: Winnow makes O(k log n) mistakes on k-disjunctions

```
Initialize : $\theta = n$;  $w_i = 1$
Prediction is 1 iff $w \bullet x \geq \theta$
If no mistake : do nothing
If $f(x) = 1$ but $w \bullet x < \theta$ ,  $w_i \leftarrow 2w_i$  (if $x_i = 1$) (promotion)
If $f(x) = 0$ but $w \bullet x \geq \theta$ ,  $w_i \leftarrow w_i/2$  (if $x_i = 1$) (demotion)
```

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

**2. v < 2(u + 1)**

**Total weight** (TW)= **n *initially*** (*we initialize weights to 1*)

*Mistake on positive*: TW(t+1) < TW(t) + n

*Mistake on negative*: TW(t+1) < TW(t) - n/2

$0 < TW < n + u\,n - v\,n/2 \Rightarrow v < 2(u+1)$

# Winnow – Mistake Bound

**Claim**: Winnow makes O(k log n) mistakes on k-disjunctions

**Initialize** : $\theta = n; \; w_i = 1$
**Prediction is 1 iff** $w \bullet x \geq \theta$
**If no mistake : do nothing**
**If** $f(x) = 1$ **but** $w \bullet x < \theta$ , $w_i \leftarrow 2w_i$ **(if** $x_i = 1$**) (promotion)**
**If** $f(x) = 0$ **but** $w \bullet x \geq \theta$ , $w_i \leftarrow w_i/2$ **(if** $x_i = 1$**) (demotion)**

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

**2. v < 2(u + 1)**

**Total weight** (TW)= n initially

**Mistake on positive: TW(t+1) < T**

Updates after a mistake on positive:
***Can promote less than n***, *otherwise no mistake in the previous step*

Mistake on negative: TW(t+1) < TW(t) - n/2

0 < TW < n + u n - v n/2 ➔ v < 2(u+1)

# Winnow – Mistake Bound

**Claim**: Winnow makes O(k log n) mistakes on k-disjunctions

**Initialize** : $\theta = n$; $w_i = 1$

**Prediction is 1 iff** $w \bullet x \geq \theta$

**If no mistake : do nothing**

**If** $f(x) = 1$ **but** $w \bullet x < \theta$ , $w_i \leftarrow 2w_i$ **(if** $x_i = 1$**) (promotion)**

**If** $f(x) = 0$ **but** $w \bullet x \geq \theta$ , $w_i \leftarrow w_i/2$ **(if** $x_i = 1$**) (demotion)**

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

## 2. v < 2(u + 1)

***Mistakes on negative examples have to demote more than n/2,*** *otherwise the dot product will be below the threshold and we wouldn't make a mistake)*

**Total weight** (TW)= n initially

Mistake on positive: TW(t+1) < T

**Mistake on negative: TW(t+1) < TW(t) - n/2**

0 < TW < n + u n - v n/2 ➔ v < 2(u+1)

## Winnow – Mistake Bound

**Claim**: Winnow makes O(k log n) mistakes on k-disjunctions

```
Initialize : $\theta = n;\ w_i = 1$
Prediction is 1 iff $w \bullet x \ge \theta$
If no mistake : do nothing
If $f(x) = 1$ but $w \bullet x < \theta$, $w_i \leftarrow 2w_i$ (if $x_i = 1$) (promotion)
If $f(x) = 0$ but $w \bullet x \ge \theta$, $w_i \leftarrow w_i/2$ (if $x_i = 1$) (demotion)
```

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

**2. v < 2(u + 1)**

**Total weight** (TW)= n initially

Mistake on positive: TW(t+1) < TW(t) + n

Mistake on negative: TW(t+1) < TW(t) - n/2

**0 < TW < n + u n - v n/2** ➔ v < 2(u+1)

TW is always positive which allow us to bound the number of negative mistakes

# Winnow – Mistake Bound

**Claim**: Winnow makes O(k log n) mistakes on k-disjunctions

```
Initialize : $\theta = n$; $w_i = 1$
Prediction is 1 iff $w \bullet x \ge \theta$
If no mistake : do nothing
If $f(x) = 1$ but $w \bullet x < \theta$, $w_i \leftarrow 2w_i$ (if $x_i = 1$) (promotion)
If $f(x) = 0$ but $w \bullet x \ge \theta$, $w_i \leftarrow w_i/2$ (if $x_i = 1$) (demotion)
```

- **u** - # of mistakes on positive examples (promotions)
- **v** - # of mistakes on negative examples (demotions)

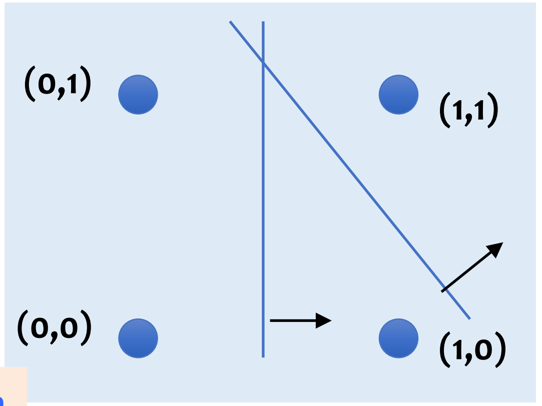
**# of mistakes:** u+ v < 3u + 2 = O(k log n)

**Mission Accomplished!** Efficient algorithm with similar mistake bound as the Halving algorithm

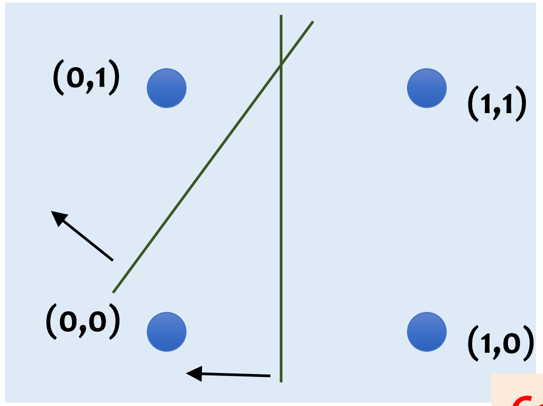
➔ **What did we just show?** *Recall definition of Mistake bounds*

# What can Winnow represent?

- The version of Winnow we saw cannot represent all disjunctions. **Why?**
- In fact, it can only represent monotone functions
  - **Multiplicative updates cannot change the sign of the weights (no negative weights)**



Can learn
$x_1 \vee x_2$
$x_2$



*Cannot* learn
$x_1 \vee !x_2$
$!x_2$

## Winnow Extension

- The algorithm we saw learns **monotone functions**
- For the general case: *Duplicate variables*
- For the negation of variable x, introduce a new variable $x^{neg}$.
- Learn monotone functions over 2n variables ***(down side?)***
- Balanced version:
  - Keep two weights for each variable; effective weight is the difference

Update Rule :

$$\text{If } f(x)=1 \text{ but } (w^+ - w^-)\bullet x \le \theta, \quad w_i^+ \leftarrow 2w_i^+ \quad w_i^- \leftarrow \frac{1}{2}w_i^- \text{ where } x_i = 1 \text{ (promotion)}$$

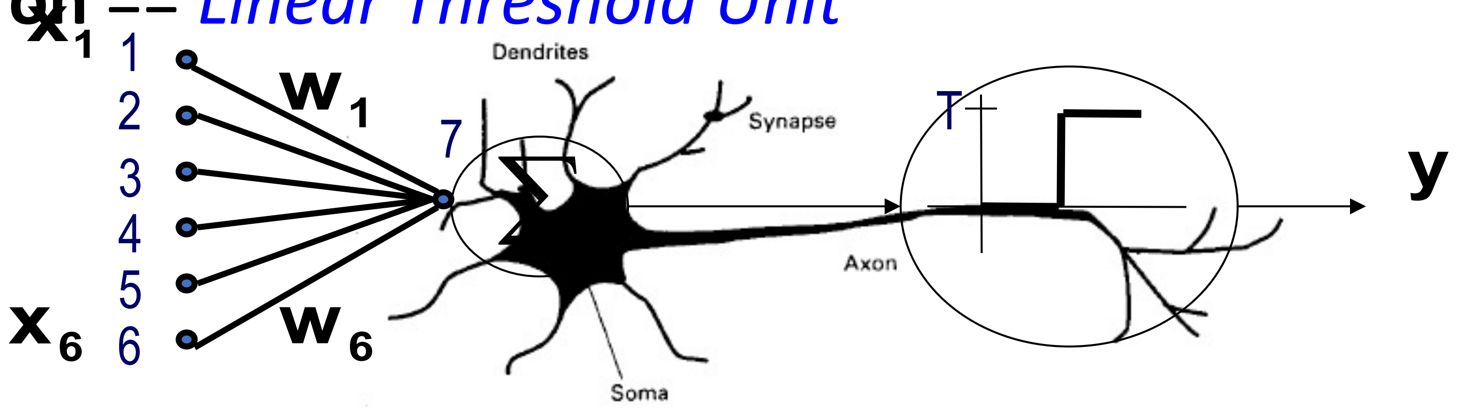
$$\text{If } f(x)=0 \text{ but } (w^+ - w^-)\bullet x \ge \theta, \quad w_i^+ \leftarrow \frac{1}{2}w_i^+ \quad w_i^- \leftarrow 2w_i^- \text{ where } x_i = 1 \text{ (demotion)}$$

## Summary

- **Learning Linear Function**
  - Very popular choice when the number of attributes is high
  - Expressive, but not all functions can be represented
    - Real world examples?
    - Several ways to deal with limited expressivity
      - Simplest: change the input space, rethink feature choices
- **Winnow**
  - First Learning algorithm for linear functions
    - Multiplicative updates
  - Applicable when concept depends on few relevant attributes
  - Nice theoretical properties: mistake bound only weakly depends on number of attributes

# Perceptron Learning Algorithm

- On-line, mistake driven algorithm.
- **Rosenblatt** (1959) suggested that when a target output value is provided for a single neuron with fixed input, it can incrementally change weights and learn to produce the output using the Perceptron learning rule
- **Perceptron** == *Linear Threshold Unit*



## Perceptron

- We learn f:X→ {-1,+1} represented as f =sgn{w•x)
- Where X= {0,1}$^n$ or X= R$^n$ and w ² R$^n$
- Given Labeled examples: $\{(x_1, y_1), (x_2, y_2),\ldots(x_m, y_m)\}$

```
1. Initialize w=0 ∊ $\mathbf{R}^n$
2. Cycle through all examples
     a. Predict the label of instance x to be y' = sgn{w•x)
     b. If y'≠y, update the weight vector:
        w = w + r y x     (r - a constant, learning rate)
        Otherwise, if y'=y, leave weights unchanged.
```

# Next time – *More on perceptron*

- We will see that the small change has strong implications
  - Multiplicative vs. additive update
  - **Mistake bound for perceptron**
- Also-
  - **Practical considerations**
    - How to get perceptron to work? What can be tuned?