

Sample dataset: predict if game was played

- **Class label** denotes whether tennis game was played.
- Columns denote **features/attributes/predictors** and **labels/targets/response**
- Rows denote **instance-label** pairs (x_n, y_n) .

Predictors				Response
Outlook	Temperature	Humidity	Wind	Class
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

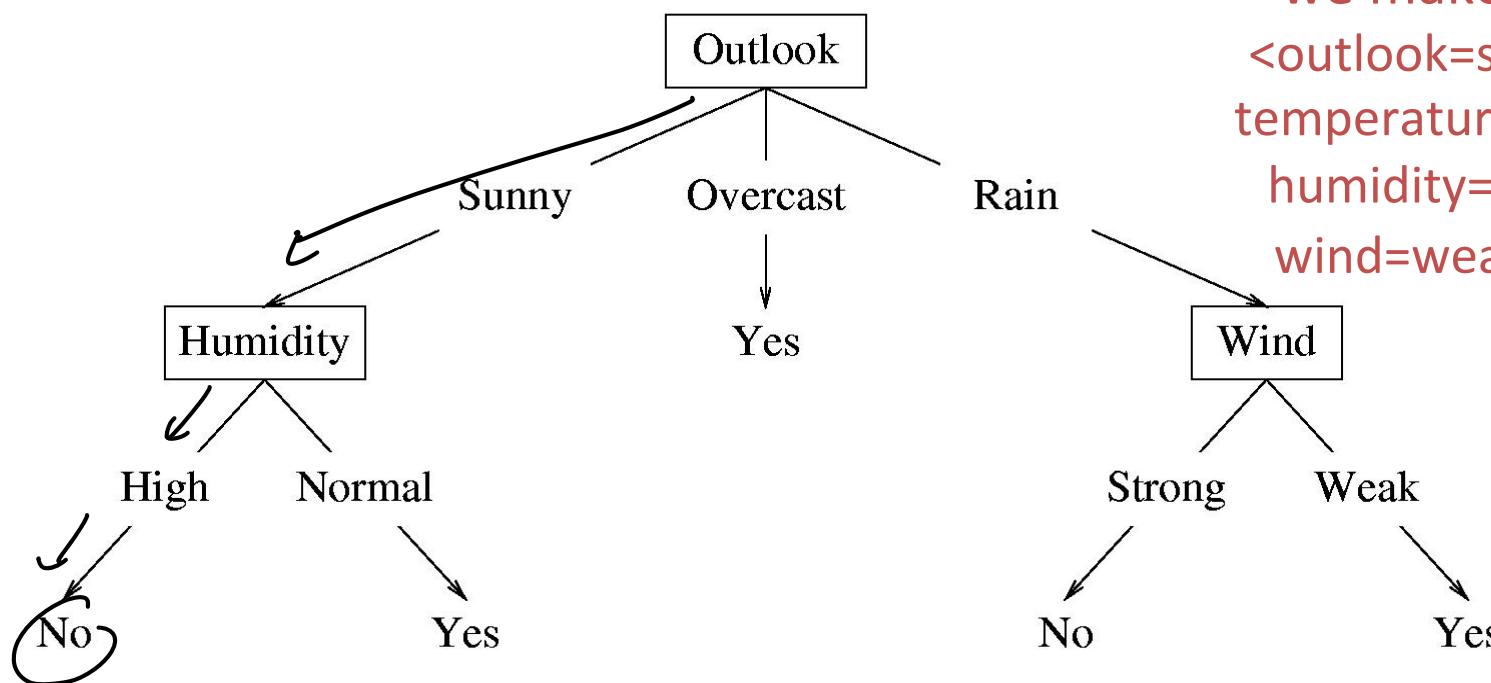
sunny, cool, normal,
weak?

overcast, cool, high,
weak?

Can you describe a
“model” that could be
used to make
decisions in general?

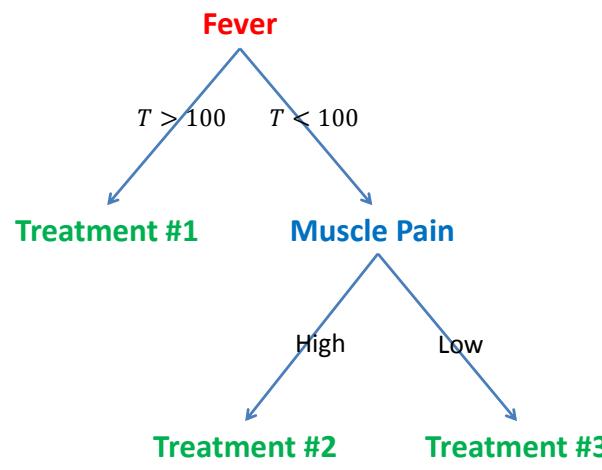
A possible decision tree

- Each internal node : test one feature
- Each edge : select one value for the feature
- Each leaf: predict class

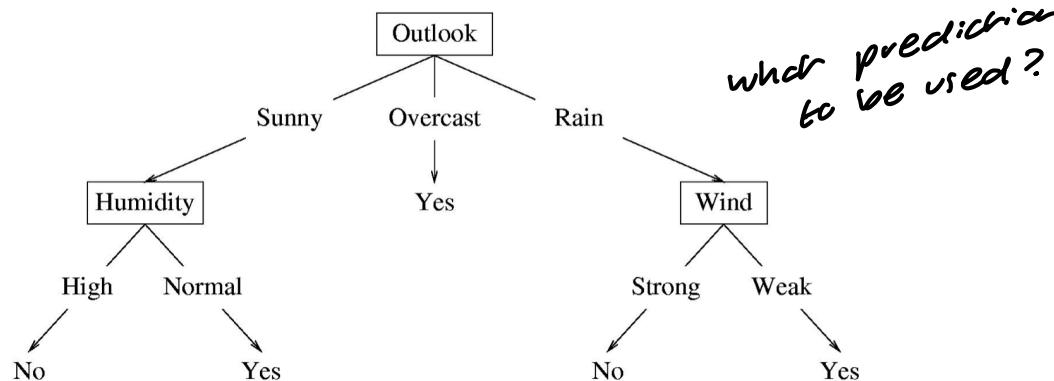
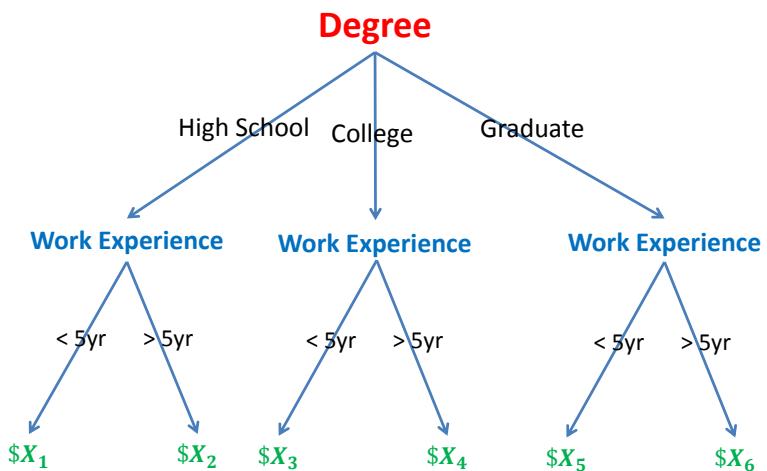


Many decisions are tree structures

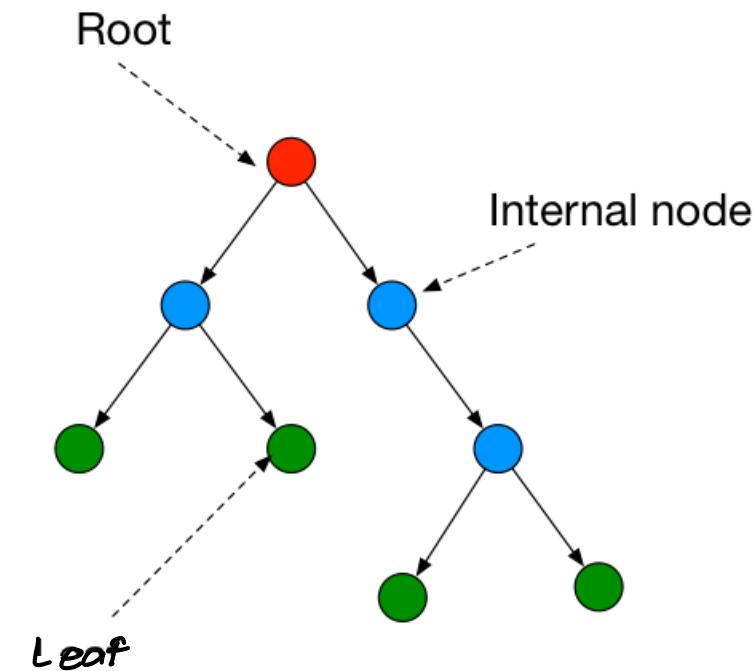
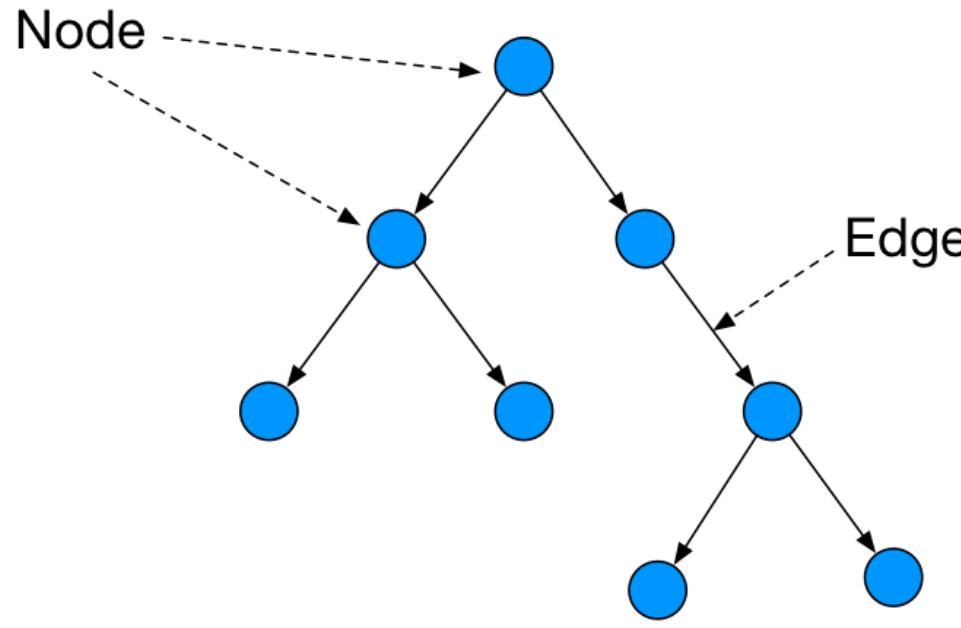
Medical treatment



Salary in a company



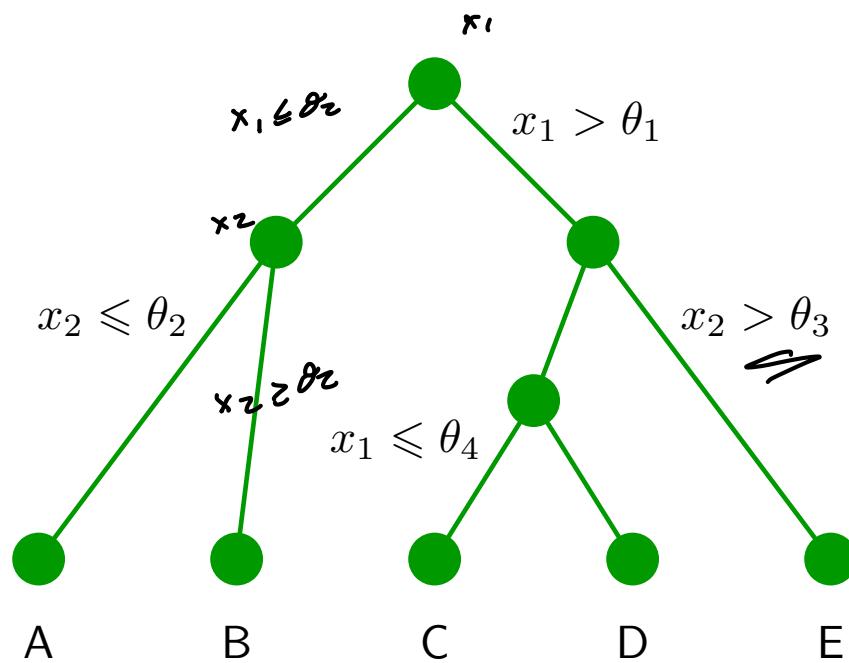
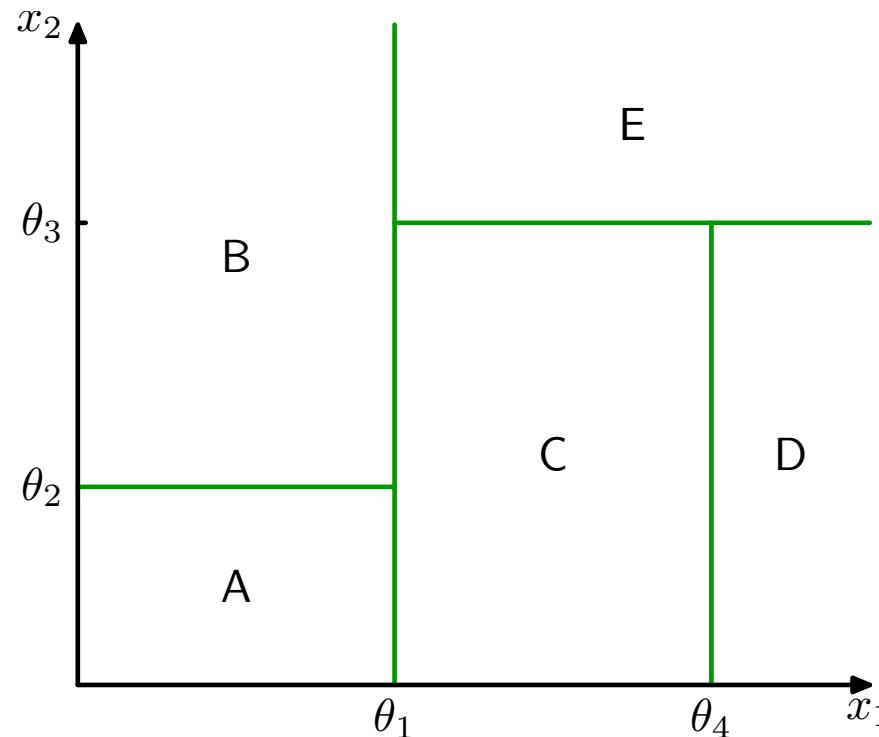
What is a Tree?



- Each node is a function
- Each edge is a feature
- Each leaf: predict class

A tree partitions the feature space

useful when comparing



Decision tree learning

learn model from data

Setup

- Set of possible **instances** \mathbb{X}
 - ▶ Each instance $x \in \mathbb{X}$ is a feature vector *discrete values*
 - ▶ (humidity=low, wind=weak, outlook=rain, temp=hot)
- Set of possible **labels** \mathbb{Y} *unknown target function*
- \mathbb{Y} is discrete valued
- Unknown target function $f : \mathbb{X} \rightarrow \mathbb{Y}$ *we don't know this function*
- **Model/Hypotheses:** $H = \{h | h : \mathbb{X} \rightarrow \mathbb{Y}\}$. *hypothesize multiple*
- Each hypothesis h is a decision tree

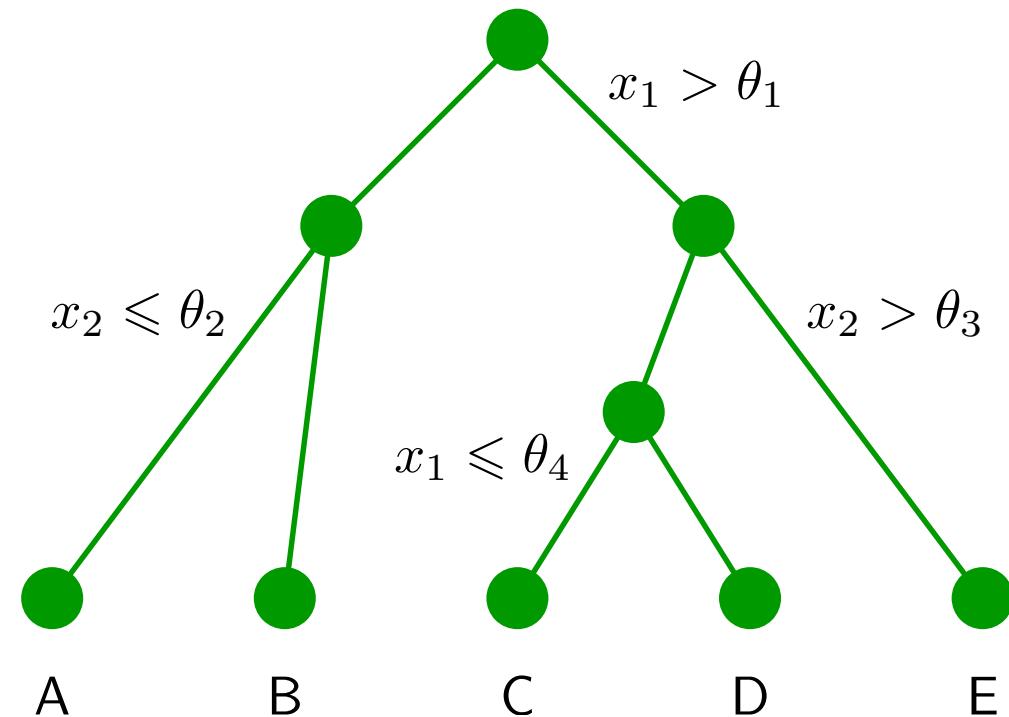
Goal: Train/induce/learn a function h that maps instance to label.

- *give all features of each node*
- *structure of tree - what is the root?*
 - ↳ *what are the values along the edges*

Learning a tree model

Three things to learn:

- ① The structure of the tree.
- ② The threshold values (θ_i).
- ③ The values for the leaves (A, B, \dots).



A tree model for deciding where to eat

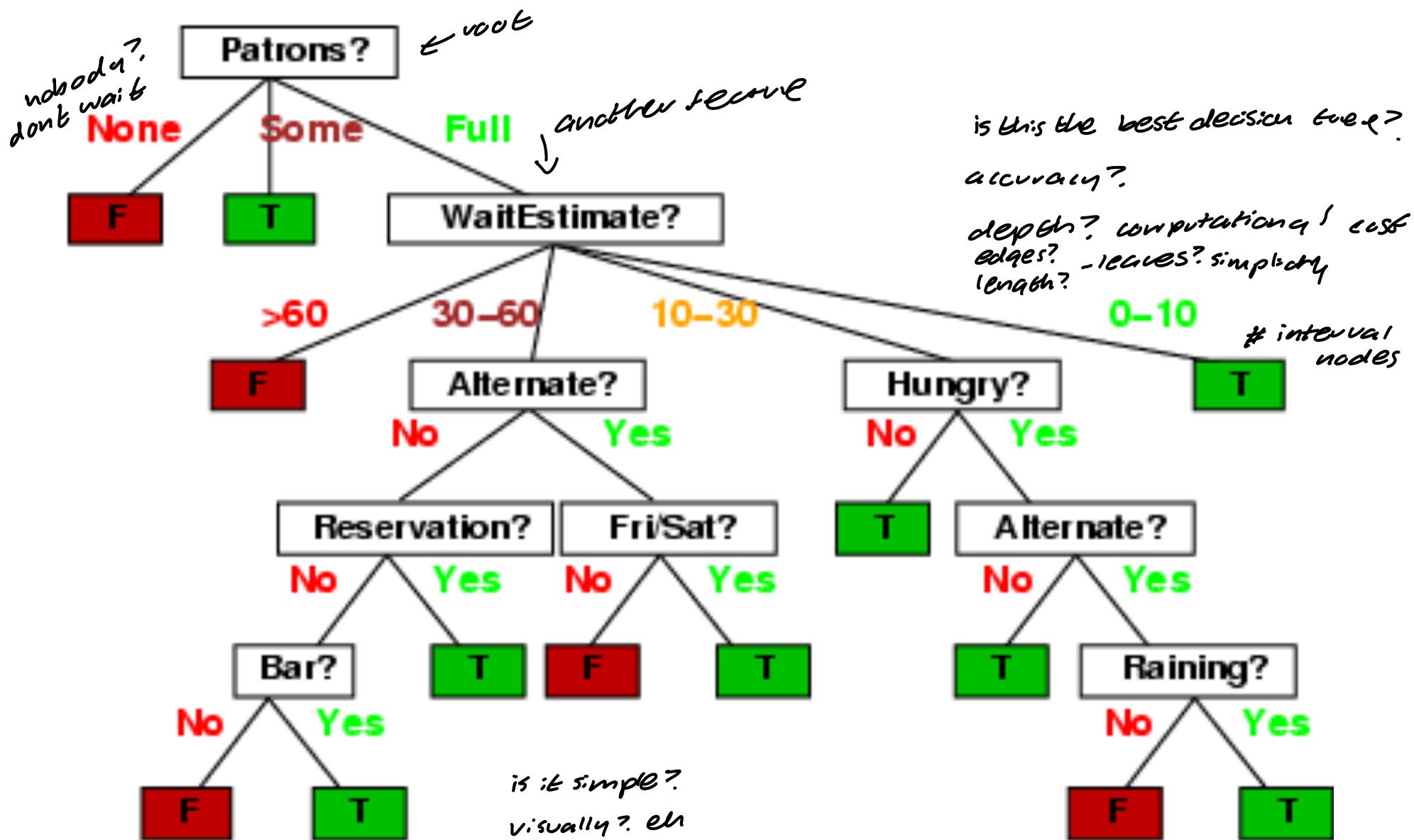
Choosing a restaurant

(Example from Russell & Norvig, AIMA)

Example	Attributes											Target <i>WillWait</i>
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>		
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T	{}
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F	
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T	
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T	
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F	
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T	
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F	
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T	
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F	
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F	
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F	
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T	

Classification of examples is positive (T) or negative (F)

A possible decision tree



Is this the best decision tree?

Ockham's razor

- when selecting a model, we should choose the simpler model that fits the data well
simpler model \rightarrow less likely to overfit & more likely to generalize well to new data
- choose subset of features that provides best performance on test data while minimizing # of features used

The simplest consistent explanation is best

- A form of **inductive bias**.
-bias that doesn't care much about data
- Find smallest decision tree that correctly classifies all training examples
characteristic
-accurate & simple decisions
- NP-hard
- Instead, greedily construct a decision tree that is pretty small.

computationally expensive

find a tree that does well in the dataset
preference for simplicity

Decision tree training/learning

For simplicity assume each feature is binary (takes values YES/NO)

Algorithm 1 DecisionTreeTrain (*data, features*)

```

1: guess  $\leftarrow$  the most frequent label in data           learning algorithm:  

2: if all labels in data are the same then      everything is binary  

3:   return LEAF (guess)                         working at pieces & putting  
them in smaller pieces.  

4: else if features is empty then          not be put at root? no, o, o  

5:   return LEAF (guess)                         take features           one smaller subsets  

6: else                                         lets assume all labels take  

7:   f  $\leftarrow$  the “best” feature  $\in$  features          the same value  

8:   NO  $\leftarrow$  the subset of data on which f = NO  

9:   YES  $\leftarrow$  the subset of data on which f = YES  

10:  left  $\leftarrow$  DecisionTreeTrain (NO, features – {f})  

11:  right  $\leftarrow$  DecisionTreeTrain (YES, features – {f})  

12:  return NODE(f, left, right)  

13: end if

```

Choosing the best feature

select the feature that gives the best output

- Possibilities
 - ▶ Random: select a feature at random
 - ▶ Highest accuracy: select feature with largest accuracy
 - ▶ Max-gain: select feature with largest information gain (to be explained shortly).
- **ID3 algorithm:** One algorithm for decision tree learning
 - ▶ Select the feature with largest information gain.

How to measure information gain?

Idea:

Gaining information reduces uncertainty

Measures the amount of uncertainty of a random variable with a specific probability distribution. Higher it is, less confident we are in its outcome.

Use entropy to measure uncertainty

If a random variable X has K different values, a_1, a_2, \dots, a_K , its entropy is given by

$$H[X] = - \sum_{k=1}^K P(X = a_k) \log P(X = a_k)$$

entropy

$P(X = a_1)$
 $P(X = a_2)$

$= - \sum [\log P(x)]$ ↑ & ↑ uncertainty
 ↓ over base? \geq

the base can be 2, though it is not essential (if the base is 2, the unit of the entropy is called "bit")



Examples of computing entropy

$$\begin{aligned} x = 0 &= 1-p \\ P(x=1) &= p \end{aligned}$$

only 2 values
binary r.v.

what is the entropy of x

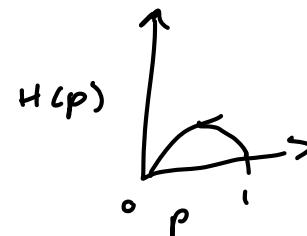
$$\begin{aligned} H[x] &= - [P(x=0) \log_2 P(x=0) + P(x=1) \log_2 P(x=1)] \\ &\approx -(1-p) \log_2(1-p) + p \log_2 p \\ &\approx - (0 \log_2 0 + 1 \log_2 1) \\ &= 0 \end{aligned}$$

$$p = 1$$

if value of p is 0, value would be 0

$$\begin{aligned} p &= \frac{1}{2} \\ P(x=0) &= \frac{1}{2} \\ P(x=1) &= \frac{1}{2} \end{aligned}$$

$$\begin{aligned} H[x] &= - \left[\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2} \right] \\ \log_2 \left(\frac{1}{2} \right) &= -\log_2(2) = 1 \\ &= 1 \end{aligned}$$



decision trees - algorithm

use entropy to see what to split

Which attribute to split?

what goes @ the root?

look @ patterns



Patrons? is a better choice—gives **information** about the classification
use information gain to choose which to split

Patron vs. Type?

By choosing Patron, we end up with a partition (3 branches) with smaller entropy, ie,
smaller uncertainty (0.45 bit)
Entropy → less uncertainty

By choosing Type, we end up with uncertainty of 1 bit.

Thus, we choose Patron over Type.

uncertainty



$$\begin{aligned} & -1.322 \\ & -\left(\frac{4}{12} \log_2 \frac{2}{5} + \frac{3}{12} \log_2 \left(\frac{3}{5}\right)\right) = .97 \\ & \quad \underbrace{-\frac{4}{12} \log_2 \frac{2}{5}}_{-0.722} \end{aligned}$$

Uncertainty if we go with “Patron”

For “None” branch

$$-\left(\frac{0}{0+2} \log \frac{0}{0+2} + \frac{2}{0+2} \log \frac{2}{0+2}\right) = 0$$

(-1.322)

For “Some” branch

$$-\left(\frac{4}{4+0} \log \frac{4}{4+0} + \frac{4}{4+0} \log \frac{4}{4+0}\right) = 0$$

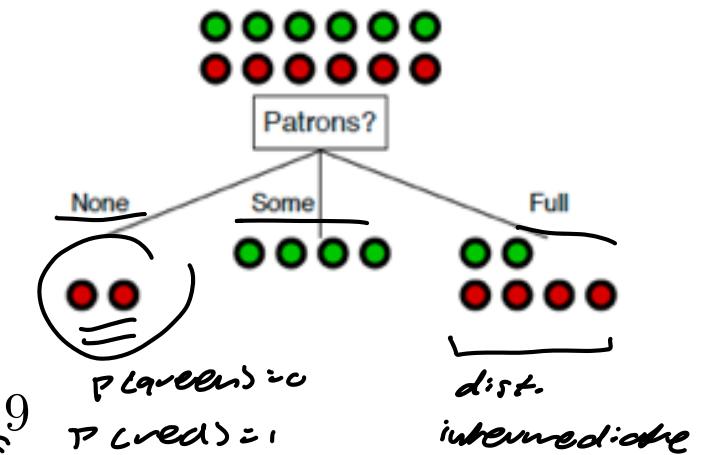
For “Full” branch

$$-\left(\frac{2}{2+4} \log \frac{2}{2+4} + \frac{4}{2+4} \log \frac{4}{2+4}\right) \approx 0.9$$

For choosing “Patrons”

weighted average of each branch: this quantity is called **conditional entropy**

$$\frac{2}{12} * 0 + \frac{4}{12} * 0 + \frac{6}{12} * 0.9 = 0.45$$



Conditional entropy

Definition. Given two random variables X and Y

condition of

$$H[Y|X] = \sum_k P(X = a_k) H[Y|X = a_k]$$

In our example

X: the attribute to be split

Y: Wait or not

x-attribute *average*

When $H[Y]$ is fixed, we need only to compare conditional entropy

Relation to information gain

$$\text{GAIN} = \underbrace{H[Y]}_{\text{reduction in uncertainty}} - \underbrace{H[Y|X]}_{}$$

Information gain in Decision trees

- The expected reduction in entropy of the target variable Y due to sorting on the feature X .
- Also called the **mutual information** between Y and X .

Conditional entropy for Type

For “French” branch

$$-\left(\frac{1}{1+1} \log \frac{1}{1+1} + \frac{1}{1+1} \log \frac{1}{1+1}\right) = 1$$

For “Italian” branch

$$-\left(\frac{1}{1+1} \log \frac{1}{1+1} + \frac{1}{1+1} \log \frac{1}{1+1}\right) = 1$$

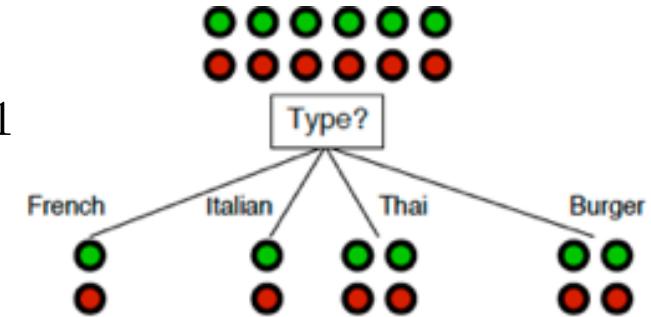
For “Thai” and “Burger” branches

$$-\left(\frac{2}{2+2} \log \frac{2}{2+2} + \frac{2}{2+2} \log \frac{2}{2+2}\right) = 1$$

For choosing “Type”

weighted average of each branch:

$$\frac{2}{12} * 1 + \frac{2}{12} * 1 + \frac{4}{12} * 1 + \frac{4}{12} * 1 = 1$$



Comparing the features/attributes

$$H[Y] = 1$$

G_T	G_F
$P(Y=T) = \frac{1}{2}$	$P(Y=F) = \frac{1}{2}$

Patrons vs Type

anonymity card. entropy of types of patrons

$$\begin{aligned}
 GAIN[Y, \text{Patrons}] &= H[Y] - H[Y|\text{Patrons}] \\
 &= 1 - 0.45 \\
 &= 0.55
 \end{aligned}$$

$$\begin{aligned}
 GAIN[Y, \text{Type}] &= H[Y] - H[Y|\text{Type}] \\
 &= 1 - 1 \\
 &= 0
 \end{aligned}$$

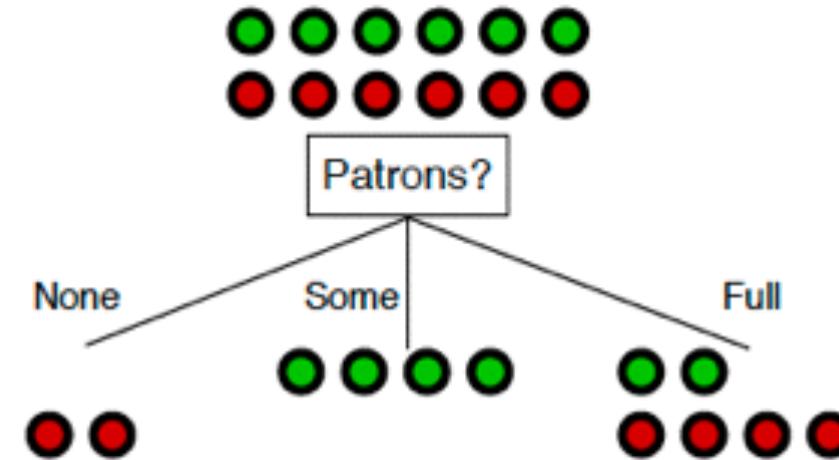
next split?

We will look only at the 6 instances with
Patrons == Full

Example	Attributes											WillWait
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est		
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0-10		T
X_2	T	F	F	T	Full	\$	F	F	Thai	30-60		F
X_3	F	T	F	F	Some	\$	F	F	Burger	0-10		T
X_4	T	F	T	T	Full	\$	F	F	Thai	10-30		T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60		F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0-10		T
X_7	F	T	F	F	None	\$	T	F	Burger	0-10		F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0-10		T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60		F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10-30		F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0-10		F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30-60		T

Classification of examples is positive (T) or negative (F)

Do we split on “Non” or “Some”?



No, we do not

The decision is deterministic, as seen from the training data

Decision tree training/learning

For simplicity assume each feature is binary (takes values YES/NO)

Algorithm 2 DecisionTreeTrain (*data, features*)

1: *guess* \leftarrow the most frequent label in *data* *given a decision tree*
2: **if** all labels in *data* are the same **then**
3: **return** LEAF (*guess*)
4: **else if** *features* is empty **then**
5: **return** LEAF (*guess*)
6: **else**
7: *f* \leftarrow the “best” feature \in *features*
8: *NO* \leftarrow the subset of *data* on which *f* = NO
9: *YES* \leftarrow the subset of *data* on which *f* = YES
10: *left* \leftarrow DecisionTreeTrain (*NO, features* − {*f*})
11: *right* \leftarrow DecisionTreeTrain (*YES, features* − {*f*})
12: **return** NODE(*f, left, right*)
13: **end if**

Decision tree training/learning

For simplicity assume each feature is binary (takes values YES/NO)

Algorithm 3 DecisionTreePredict (tree, instance)

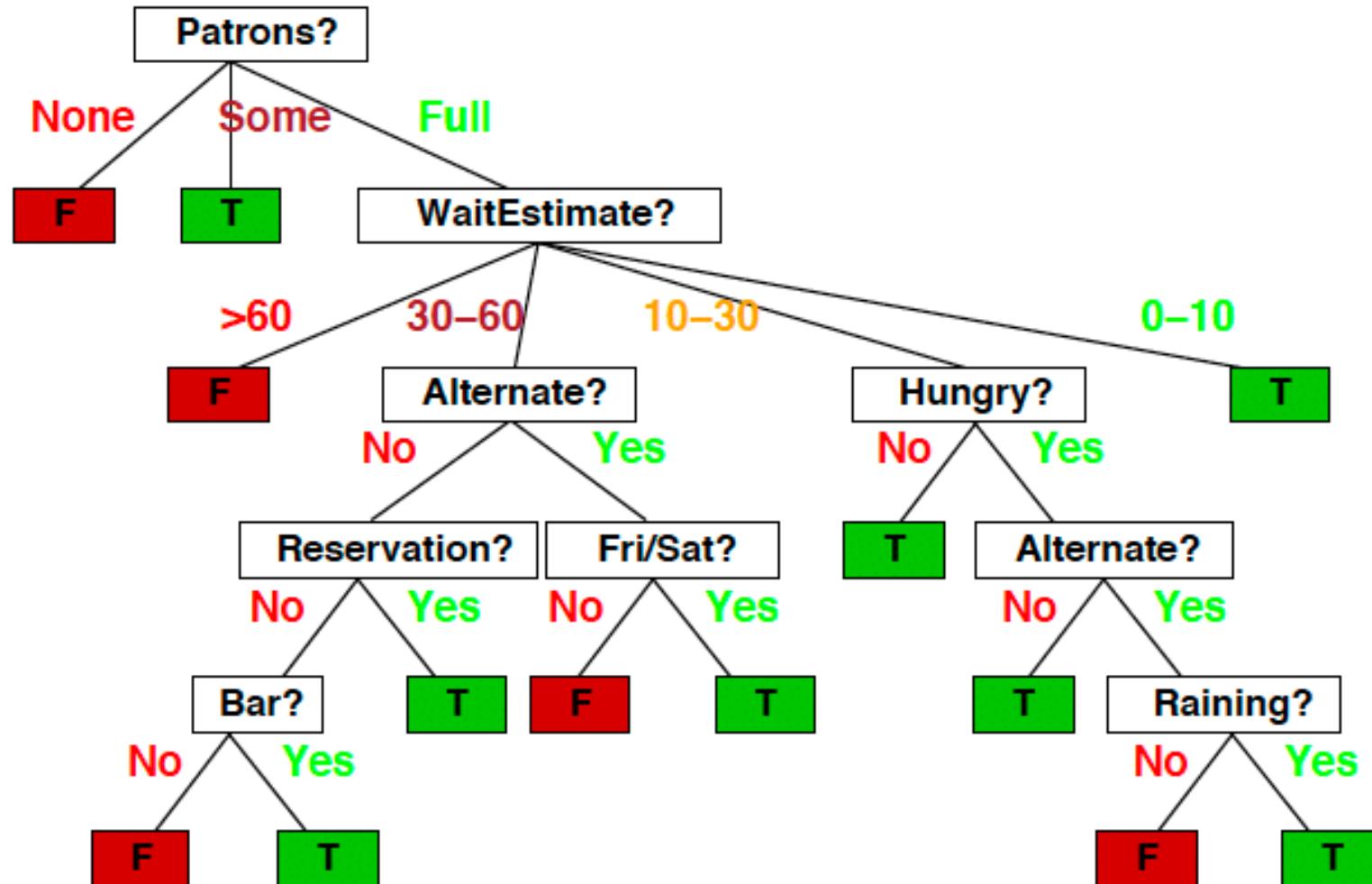
```

1: if tree has form LEAF (guess) then
2:   return guess
3: else if tree has form NODE(f,left,right) then
4:   if f = NO in instance then
5:     return DecisionTreeTest(left,instance)
6:   else
7:     return DecisionTreeTest(right,instance)
8:   end if
9: end if

```

tree \Rightarrow leaf
 same pred.

Greedily we build the tree and get this



Require the tree is not too deep

fix depth parameter

- part test or train data - averaging

Prefering shallow trees: a form of inductive bias

- We need to be careful to pick an appropriate tree depth
- If the tree is too deep, we can overfit (memorize the training data).
- If the tree is too shallow, we underfit (not learn enough).
- Max depth is a **hyperparameter** that should be tuned by the data
 - ▶ A parameter that controls the other parameters of the tree.
 - ▶ Max depth of 0: underfitting
 - ▶ Max depth of ∞ : overfitting.

causes over parameters

imp. to find max depth

no root?

not learning from data

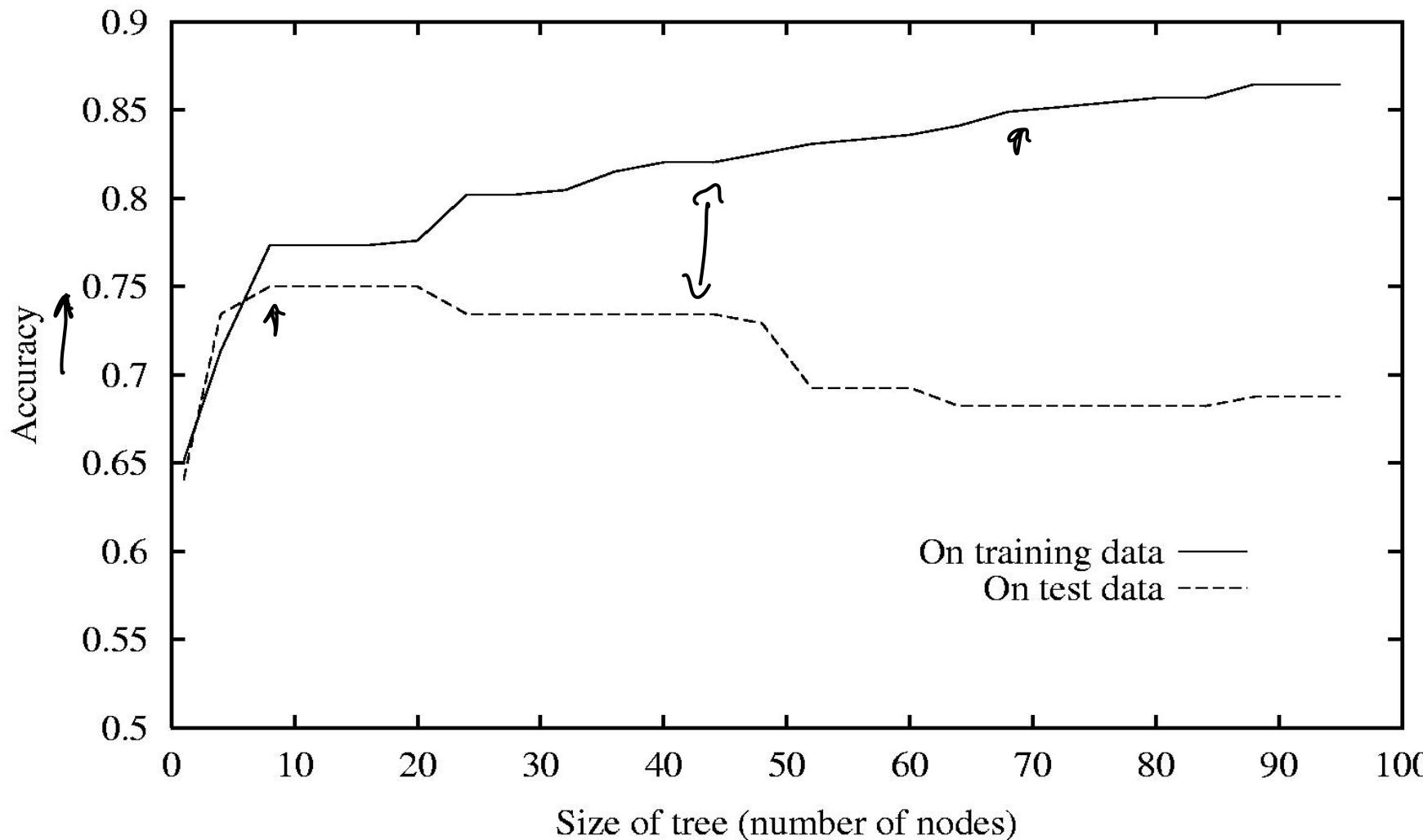
Overfitting

Reasons for overfitting

- Noisy data
 - ▶ Two instances have the same feature values but different class labels
 - ▶ Some of the feature or label values are incorrect
- Some features are irrelevant to classification.
- Target variable is non-deterministic in the features.
 - ▶ In general, we cannot measure all the variables needed to predict. So target variable is not uniquely determined by the input feature values.

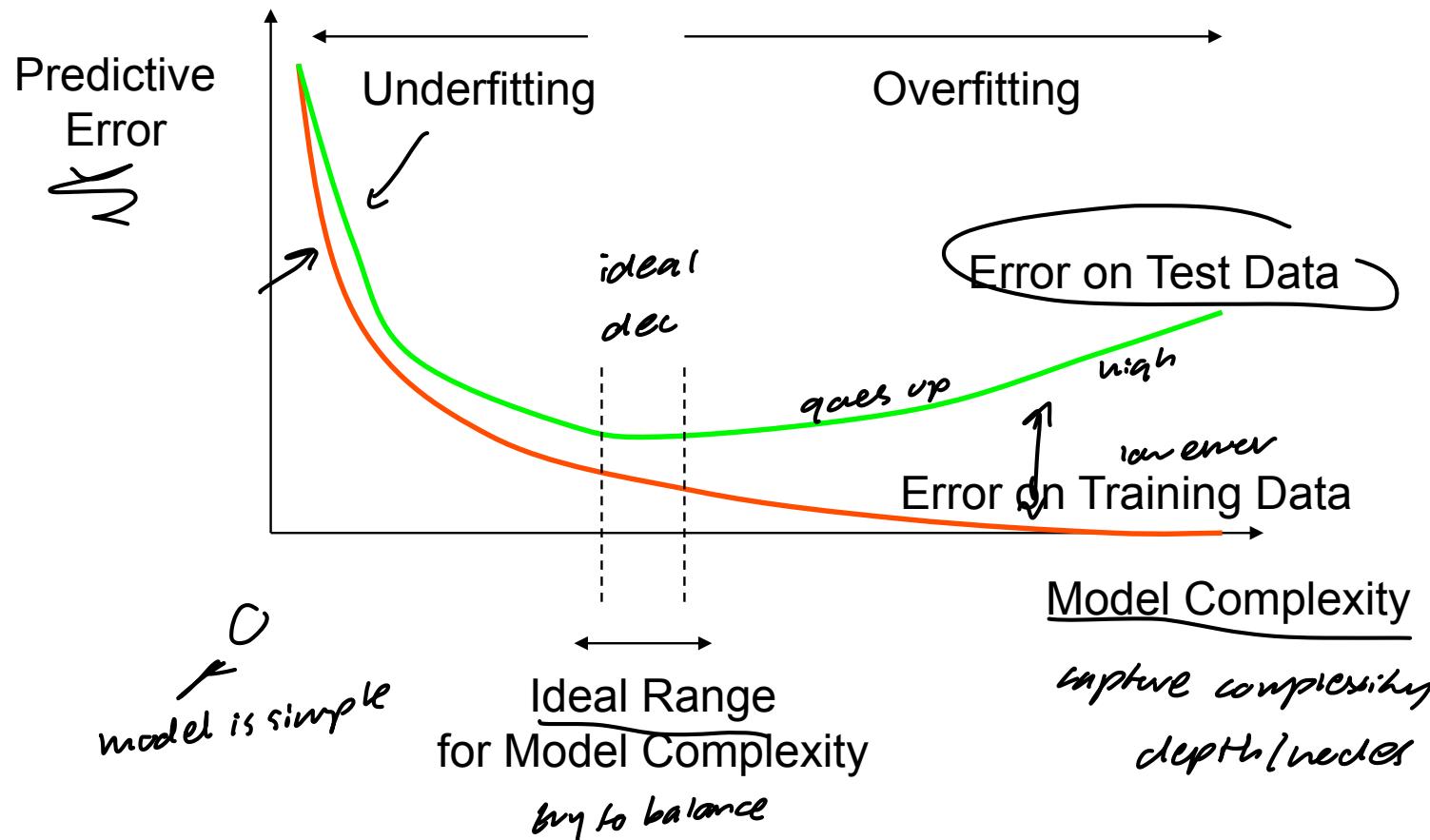
⇒ training error is not guaranteed to be zero

Overfitting in decision trees



Our decision tree learning procedure always increases training set accuracy.
Though training accuracy is increasing, test accuracy can decrease.

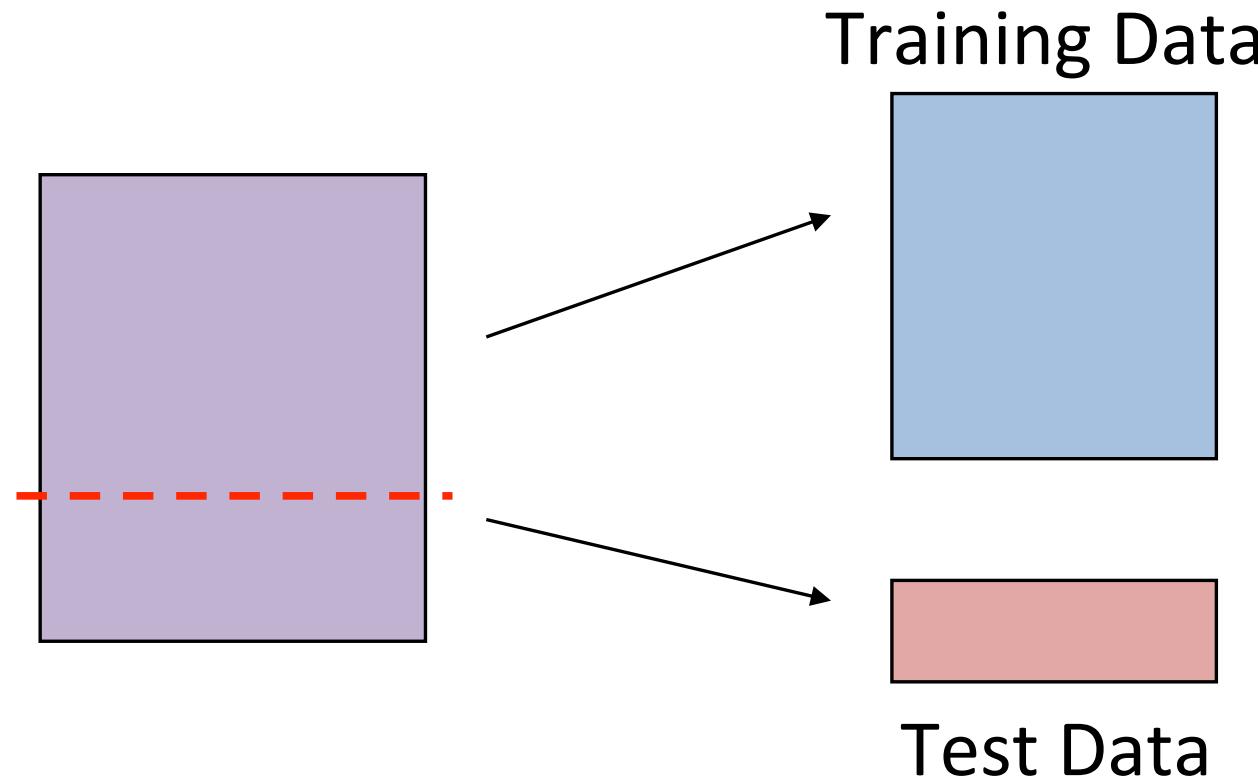
decision trees - a learning paradigm
Overfitting paradigm
Learn



How to detect overfitting?

More generally, how to get a realistic estimate of accuracy

Train and test data



- Train model on training data
- Compute accuracy on test data
- **Downside:** Throwing out data.

Cross-validation CV

N instances - fold

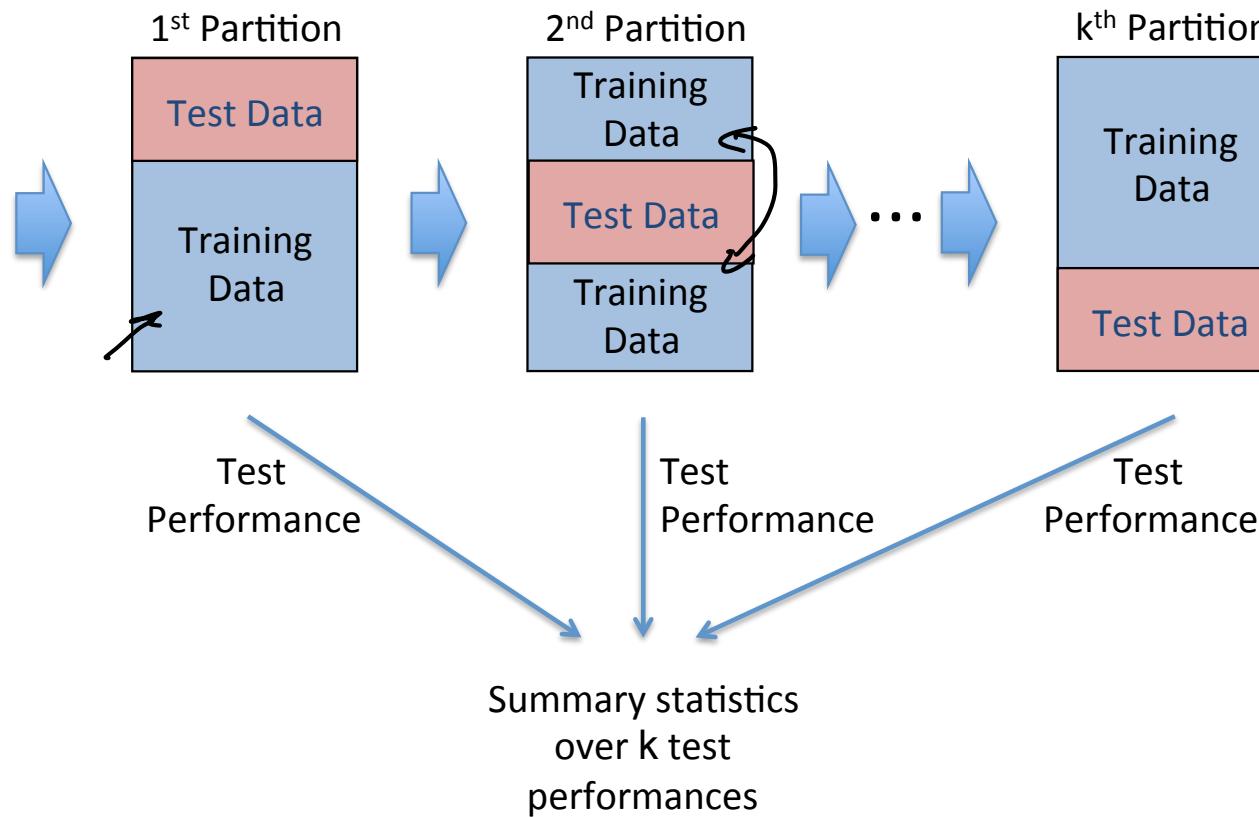
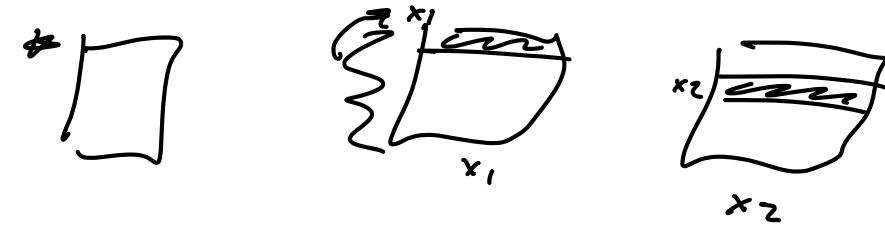
Why just choose one particular split of the data?

- In principle, we should do this multiple times since performance may be different for each split
- K-Fold Cross-Validation (e.g., $K = 10$)
 - ▶ Randomly partition full data set of N instances into K disjoint subsets (each roughly of size $\frac{N}{K}$)
 - ▶ Choose each fold in turn as the test set; train model on the other folds and evaluate.
fold becomes less
 - ▶ Compute average statistics over K test performances. Can also do leave-one-out CV where $K = N$.
average stats

decision trees - algorithms

Cross-validation CV

3-fold CV

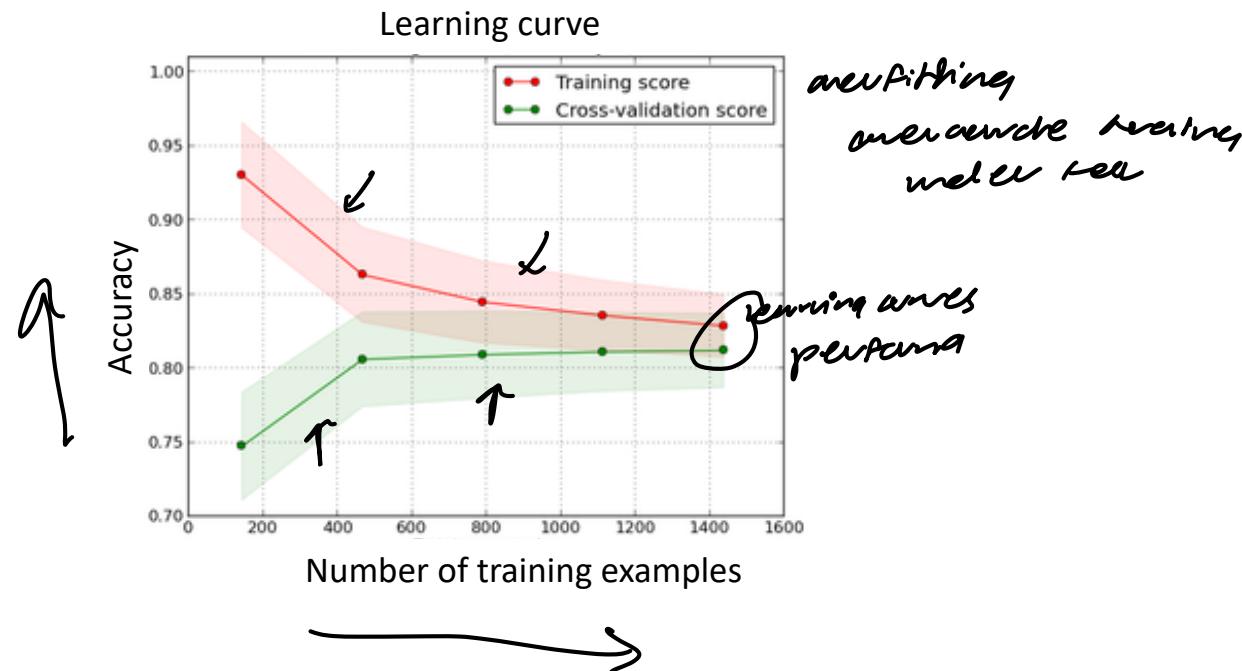


Recipe for K-fold CV

- Split the training data into K equal parts. Denote each part as $\mathcal{D}_k^{\text{TRAIN}}$
- for every $k \in [1, K]$
 - ▶ Train a model using $\mathcal{D}_{\setminus k}^{\text{TRAIN}} = \mathcal{D}_{\text{TRAIN}}^{\text{TRAIN}} - \mathcal{D}_k^{\text{TRAIN}}$
all of training data \setminus k
 - ▶ Evaluate the performance of the model on $\mathcal{D}_k^{\text{TRAIN}}$
- Average the K performance metrics

Visualizing overfitting: Learning curves

- Shows performance versus number of training instances.
 - Compute over a single training/test split.
 - Average over multiple trials of CV

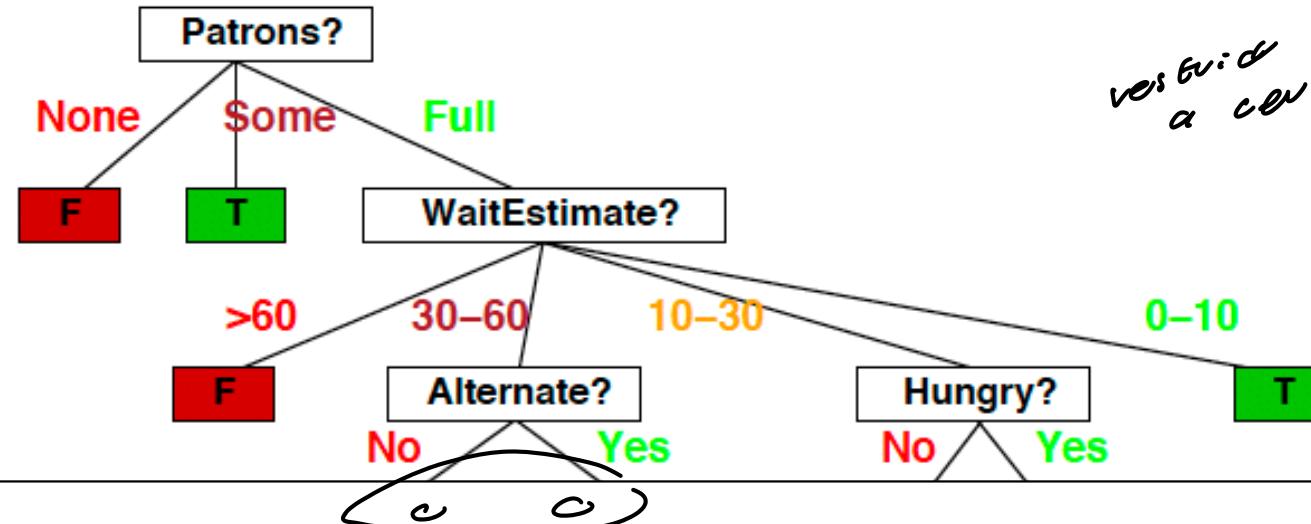


Avoiding overfitting

- Get more training data
- Remove irrelevant features
- Force the decision tree to be simple : Decision tree pruning
 - ▶ Prune while building tree (early stopping)
 - ▶ Prune after building tree (post-pruning)

Control the size of the tree

We would prune to have a smaller one



versatile to
a certain degree.

If we stop here, not all training sample would be classified correctly.

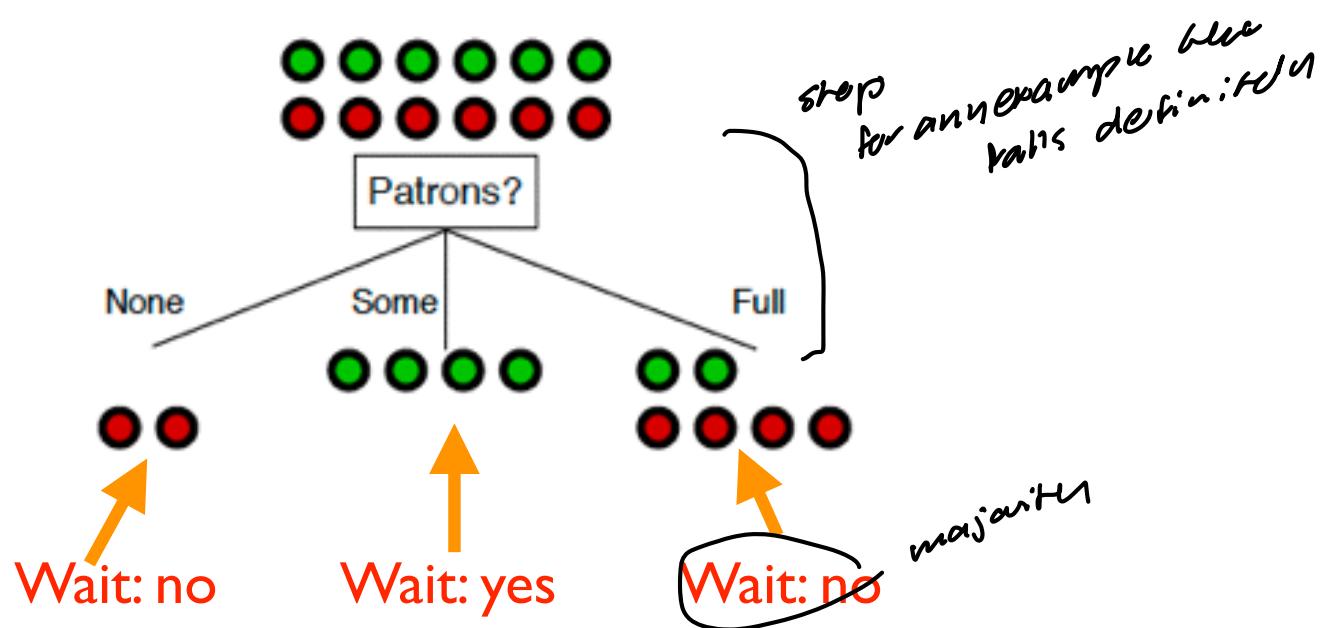
More importantly, how do we classify a new instance?

We label the leaves of this smaller tree with the majority of training samples' labels

Example

Example

We stop after the root (first node)



Hyperparameters in a Decision tree

Maximum depth of the tree

- Critical to choose max depth carefully for good generalization.
- Choosing max depth (as well as any other hyperparameters) based on empirical performance.

How do we choose max depth?

ML have diff. problems

Hyperparameter selection (tuning) by using a validation dataset

common paradigm

Training data (set) validation set

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$
- They are used for learning $h(\cdot)$ assesses model on unseen data

Test (evaluation) data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(x_1, y_1), (x_2, y_2), \dots, (x_M, y_M)\}$
- They are used for assessing how well $h(\cdot)$ will do in predicting an unseen $x \notin \mathcal{D}^{\text{TRAIN}}$

Validation (or development) data

- L samples/instances: $\mathcal{D}^{\text{DEV}} = \{(x_1, y_1), (x_2, y_2), \dots, (x_L, y_L)\}$
- They are used to optimize hyperparameter(s).
no overlap

Training data, validation and test data should **not** overlap!

Recipe for hyperparameter selection

- For each possible value of the hyperparameter (say Max depth = 1, 3, ⋯)
 - ▶ Train a model using $\mathcal{D}^{\text{TRAIN}} - \text{train}$
 - ▶ Evaluate the performance of the model on $\mathcal{D}^{\text{DEV}} - \text{eval}$.
- Choose the model with the best performance on \mathcal{D}^{DEV}
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

Z

Cross-validation (CV)

What if we do not have validation data?

- We split the training data into K equal parts (termed **folds** or **splits**).
- We use each part *in turn* as a validation dataset and use the others as a training dataset.
- We choose the hyperparameter such that ***on average***, the model performing the best

$K = 5$: 5-fold cross validation



Special case: when $K = N$, this will be leave-one-out (LOO).

Recipe

mc models

- Split the training data into K equal parts. Denote each part as $\mathcal{D}_k^{\text{TRAIN}}$
- For each possible value of the hyperparameter (say Max depth = 1, 3, \dots)
 - ▶ for every $k \in [1, K]$
 - ★ Train a model using $\mathcal{D}_{\setminus k}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_k^{\text{TRAIN}}$
 - ★ Evaluate the performance of the model on $\mathcal{D}_k^{\text{TRAIN}}$
 - ▶ Average the K performance metrics
- Choose the hyperparameter corresponding to the best averaged performance
- Use the best hyperparameter to train on a model using all $\mathcal{D}^{\text{TRAIN}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

Summary

Advantages of using trees

- Easily interpretable by humans (as long as the tree is not too big)
- Computationally efficient
- Handles both numerical and categorical data

Disadvantages

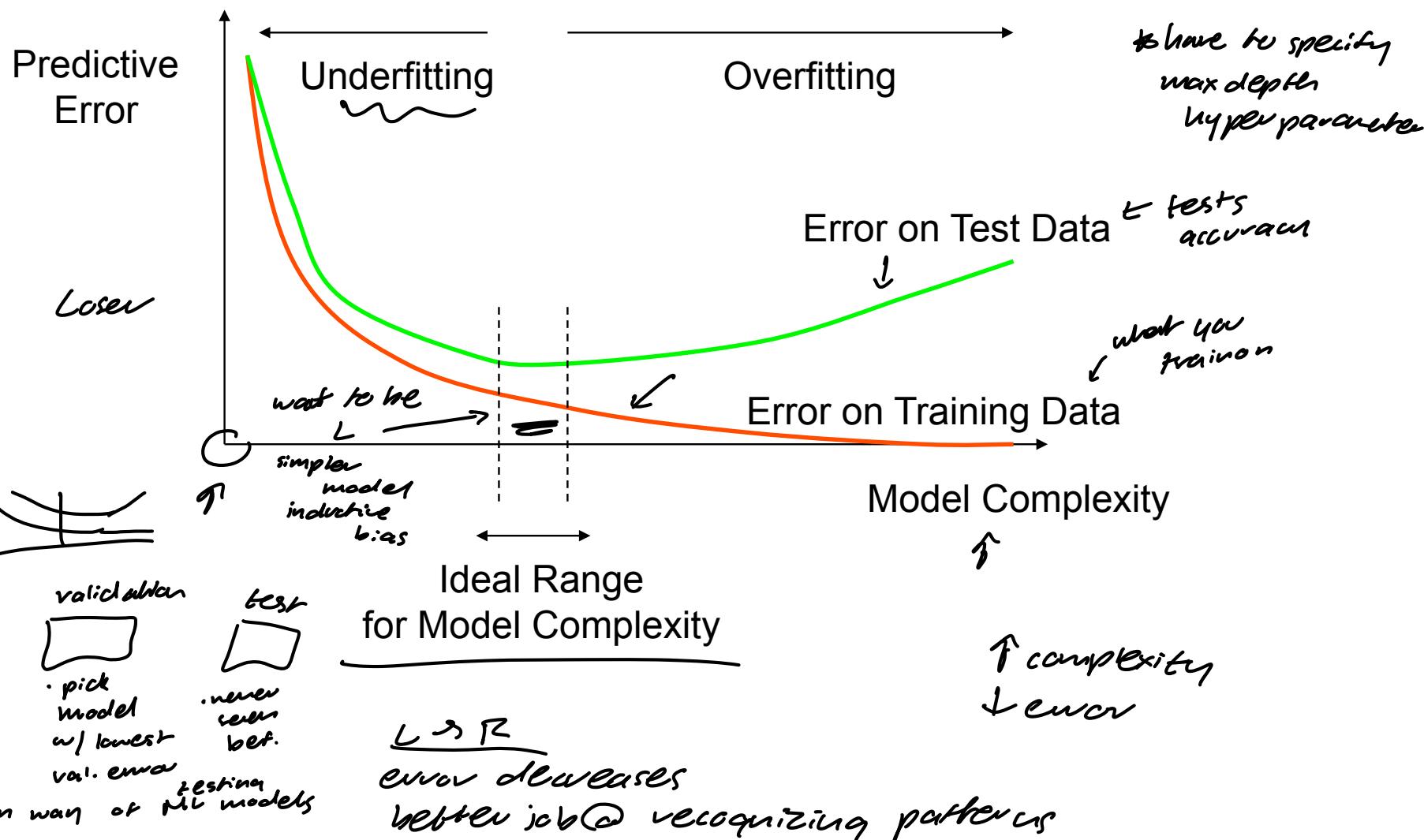
- Heuristic training techniques
 - ▶ Finding a tree that minimizes empirical error is NP-hard
 - ▶ Resort to greedy approaches.
- You should now be able to use decision trees to do machine learning.
- Given data, use training, development and test splits (or cross-validation).
- Use training and development to tune for Max depth that trades off overfitting and underfitting.
- Use test to get an estimate of generalization or accuracy on unseen data.
 - Model/hypotheses: Trees
 - Parameters: Features at each internal node, feature values along edges and labels at leaves.
 - Hyperparameters: max depth

What is the optimal Tree Depth?

- We need to be careful to pick an appropriate tree depth
- If the tree is too deep, we can overfit (memorize the training data).
- If the tree is too shallow, we underfit (not learn enough).
- Max depth is a hyperparameter that should be tuned by the data
 - ▶ A parameter that controls the other parameters of the tree.
 - ▶ Max depth of 0: underfitting
 - ▶ Max depth of ∞ : overfitting.

Overfitting

overfit
• error on training & test (higher)



Hyperparameters in Decision tree

Max depth of the tree

- Critical to choose max depth carefully for good generalization.
- Choosing max depth (as well as any other hyperparameters) based on empirical studies.

How do we choose max depth?

Hyperparameter selection (tuning) by using a validation dataset

nonoverlapping

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$
- They are used for learning $h(\cdot)$

Test (evaluation) data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_M, y_M)\}$
- They are used for assessing how well $h(\cdot)$ will do in predicting an unseen $\mathbf{x} \notin \mathcal{D}^{\text{TRAIN}}$

Validation (or development) data

- L samples/instances: $\mathcal{D}^{\text{DEV}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_L, y_L)\}$
- They are used to optimize hyperparameter(s).

Training data, validation and test data should *not* overlap!

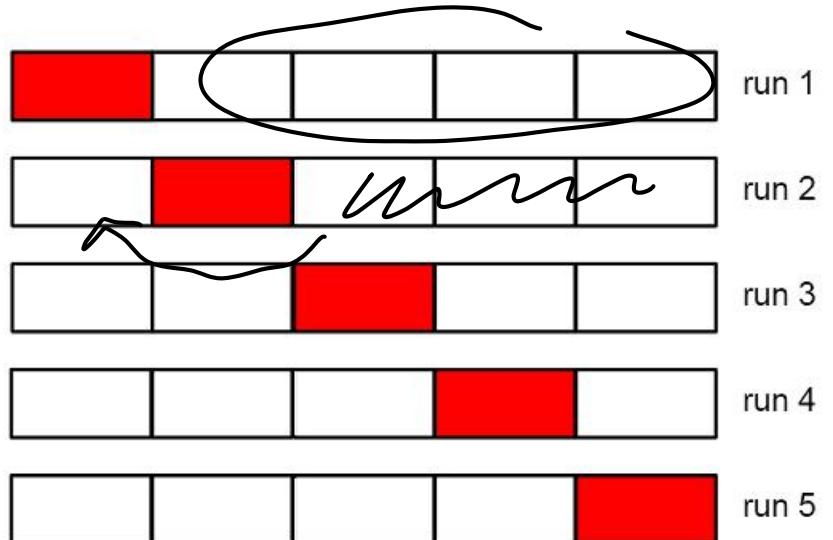
Cross-validation (CV)

split data in K equal parts
1 round of model

What if we do not have validation data?

- We split the training data into K equal parts (termed **folds** or **splits**).
- We use each part *in turn* as a validation dataset and use the others as a training dataset.
- We choose the hyperparameter such that ***on average***, the model performing the best

$K = 5$: 5-fold cross validation

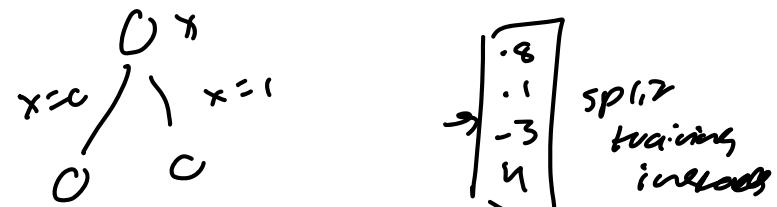


Special case: when $K = N$, this will be leave-one-out (LOO).

Computational Considerations

numerical + categorical features

threshold
 $x \in \{0, 1\}$
training



Numerical Features $x \in \{0, 1\}$

- We could split on any feature, with any threshold $\rightarrow 1.1 / .9 / 4$
- However, for a given feature, the only split points we need to consider are the n values in the training data for this feature.
- If we sort each feature by these n values, we can quickly compute our metric of interest (gain or conditional entropy)

Computational Considerations

Numerical Features

- We could split on any feature, with any threshold
- However, for a given feature, the only split points we need to consider are the n values in the training data for this feature.
- If we sort each feature by these n values, we can quickly compute our metric of interest (gain or conditional entropy)
 - ▶ This takes $O(dn \log n)$ time

Outline

1 Review of previous lecture

2 Nearest neighbor classifier

- Example
- General setup for classification
- Algorithm
- Some practical sides of NNC
- Preprocessing data

3 What we have learned

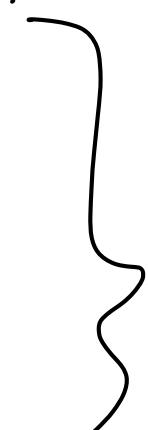
Snapshot of Iris data

Iris data

- 4 features
- 3 classes

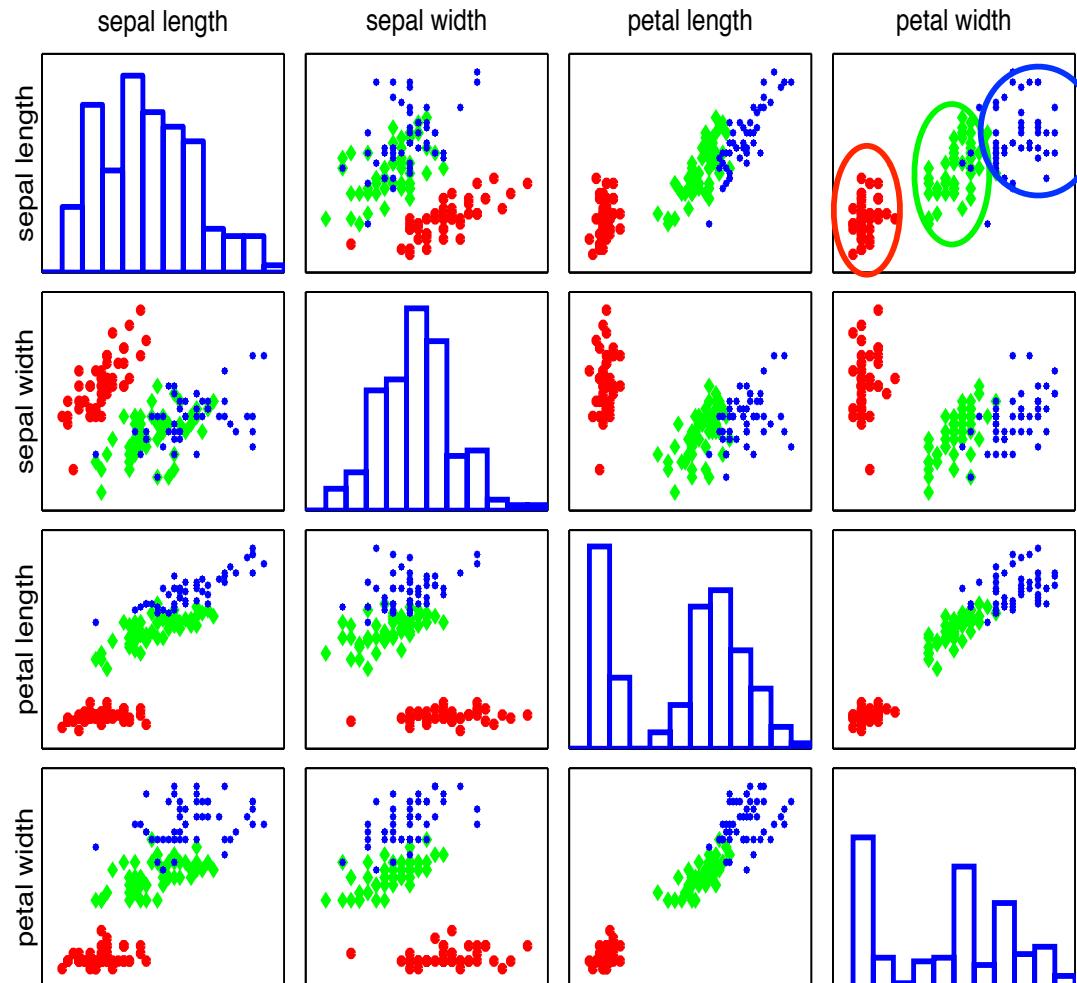
Fisher's Iris Data

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	<i>I. setosa</i>
4.9	3.0	1.4	0.2	<i>I. setosa</i>
4.7	3.2	1.3	0.2	<i>I. setosa</i>
4.6	3.1	1.5	0.2	<i>I. setosa</i>
5.0	3.6	1.4	0.2	<i>I. setosa</i>
5.4	3.9	1.7	0.4	<i>I. setosa</i>
4.6	3.4	1.4	0.3	<i>I. setosa</i>
5.0	3.4	1.5	0.2	<i>I. setosa</i>
4.4	2.9	1.4	0.2	<i>I. setosa</i>
4.9	3.1	1.5	0.1	<i>I. setosa</i>

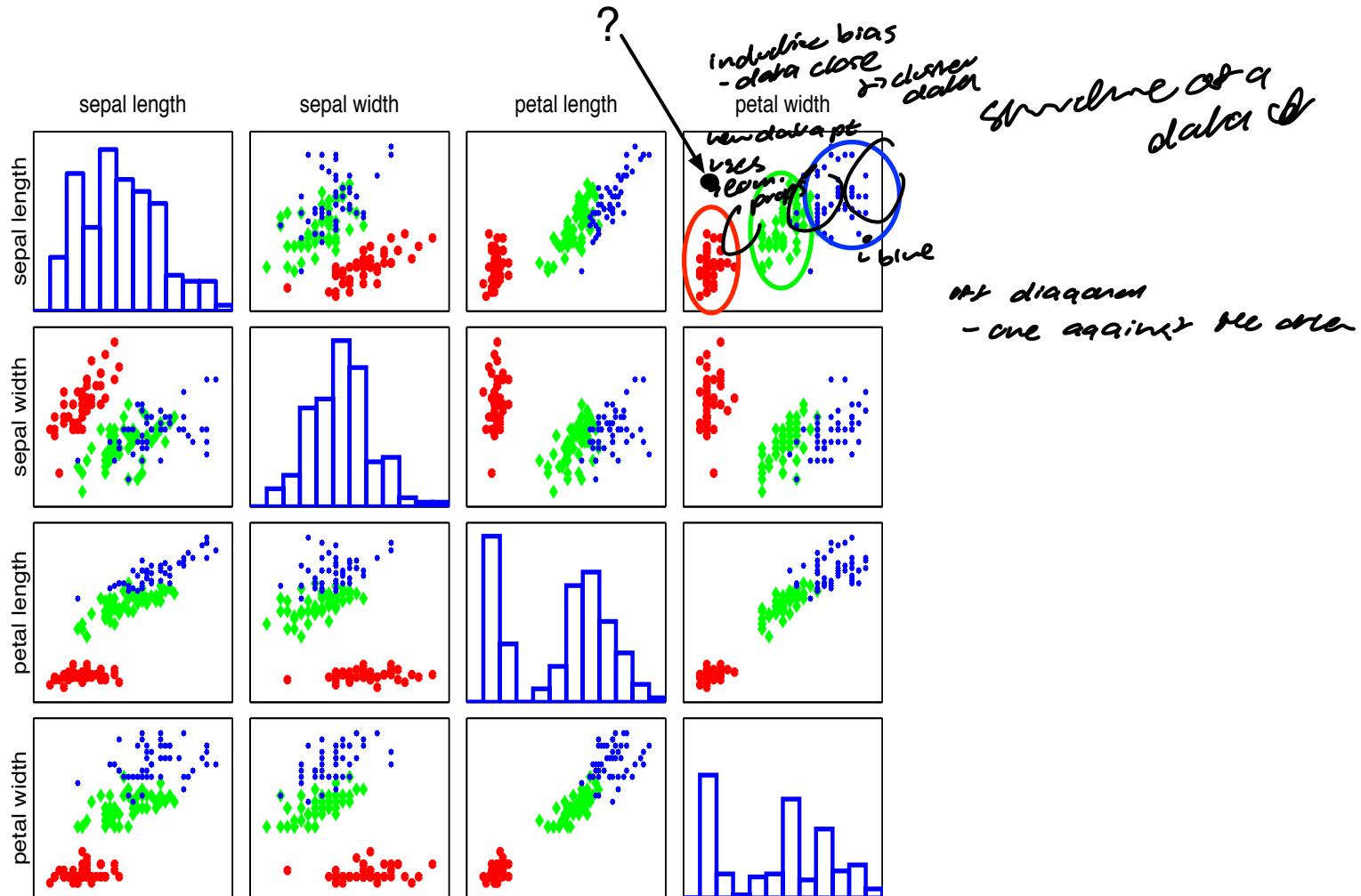


Different types seem well-clustered and separable

Using two features: petal width and sepal length



Labeling an unknown flower type



Closer to red cluster: so labeling it as setosa

Inductive bias

- Label of point (instance) is similar to the label of nearby points.

Multi-class classification

Classify data into one of the multiple categories ↑ even in 3 categories - RGTS

- Instance (feature vectors): $x \in \mathbb{R}^D$ - D dimensional $x = \begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix}$
- Label: $y \in [C] = \{1, 2, \dots, C\}$
- Learning goal: $y = h(x)$ maps x into y

Special case: binary classification

- Number of classes: $C = 2$
- Labels: $\{0, 1\}$ or $\{-1, +1\}$

Terminology

Training data (set)

n pairs or instances & labels

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$
- They are used for learning $h(\cdot)$

Test (evaluation) data

m possible pairs

how well it is generalized

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(x_1, y_1), (x_2, y_2), \dots, (x_M, y_M)\}$
- They are used for assessing how well $h(\cdot)$ will do in predicting an unseen $x \notin \mathcal{D}^{\text{TRAIN}}$

Training data and test data should *not* overlap: $\mathcal{D}^{\text{TRAIN}} \cap \mathcal{D}^{\text{TEST}} = \emptyset$

Nearest neighbor classification (NNC)

randomly assign classifications

Training

- Store the entire training set.

Testing/Prediction

$$x(1) = x_{\text{nn}(x)}$$

where $\text{nn}(x) \in [N] = \{1, 2, \dots, N\}$, i.e., the index to one of the training instances

$$\text{nn}(x) = \arg \min_{n \in [N]} \|x - x_n\|_2^2 = \arg \min_{n \in [N]} \sum_{d=1}^D (x_d - x_{nd})^2$$

euclidean distance

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix} \quad x_n = \begin{pmatrix} x_{n1} \\ x_{n2} \\ \vdots \\ x_{nD} \end{pmatrix}$$
$$\|x - x_n\|_2^2 = \sum_{d=1}^D (x_d - x_{nd})^2$$
$$\|x - x_n\|_2^2 = \sqrt{\sum_{d=1}^D (x_d - x_{nd})^2}$$

Nearest neighbor classification (NNC)

Testing/Prediction

$$\mathbf{x}(1) = \mathbf{x}_{\text{nn}(\mathbf{x})}$$

where $\text{nn}(\mathbf{x}) \in [N] = \{1, 2, \dots, N\}$, i.e., the index to one of the training instances

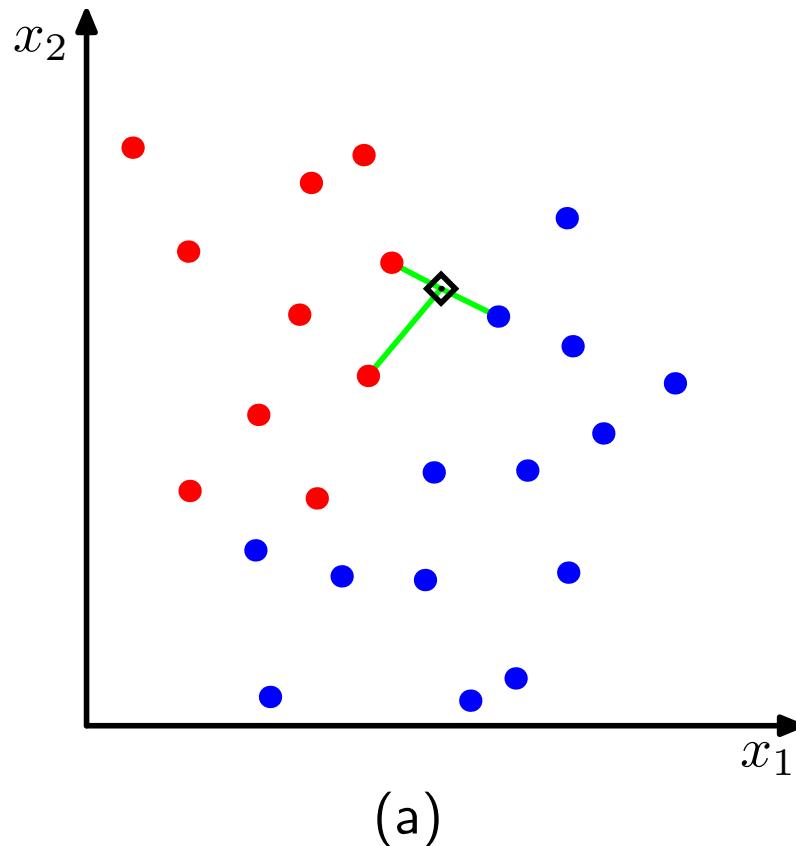
$$\text{nn}(\mathbf{x}) = \arg \min_{n \in [N]} \|\mathbf{x} - \mathbf{x}_n\|_2^2 = \arg \min_{n \in [N]} \sum_{d=1}^D (x_d - x_{nd})^2$$

Classification rule

$$y = \underbrace{h(\mathbf{x})}_{\text{randomization}} = \underbrace{y_{\text{nn}(\mathbf{x})}}_{\text{randomization}}$$

Visual example

In this 2-dimensional example, the nearest point to x is a **red training instance**, thus, x will be labeled as **red**.



Example: classify Iris with two features

Training data

ID (n)	petal width (x_1)	sepal length (x_2)	category (y)
1	0.2	5.1	setosa
2	1.4	7.0	versicolor
3	2.5	6.7	virginica

Flower with unknown category

petal width = 1.8 and sepal length = 6.4

Calculating distance = $\sqrt{(x_1 - x_{n1})^2 + (x_2 - x_{n2})^2}$

ID	distance
1	1.75
2	0.72
3	0.76

Thus, the category is *versicolor* (the real category is *virginica*)

How to measure nearness with other distances?

	$\sqrt{2}$	\sqrt{n}	$(x_1, y_1), (x_2, y_2)$
$(-1, 0)$	1 - Θ	$(1, 0)$	$ x_2 - x_1 + y_2 - y_1 $
$(0, 1)$	2 - Θ	$(2, 0)$	
$(0, 2)$	2 - Θ	$(2, 2)$	
$(1, -1)$	5	$(2, 3)$	
$(1, 0)$	4	.	

Previously, we use the Euclidean distance

$$\text{nn}(x) = \arg \min_{n \in [N]} \|x - x_n\|_2^2$$

We can also use alternative distances

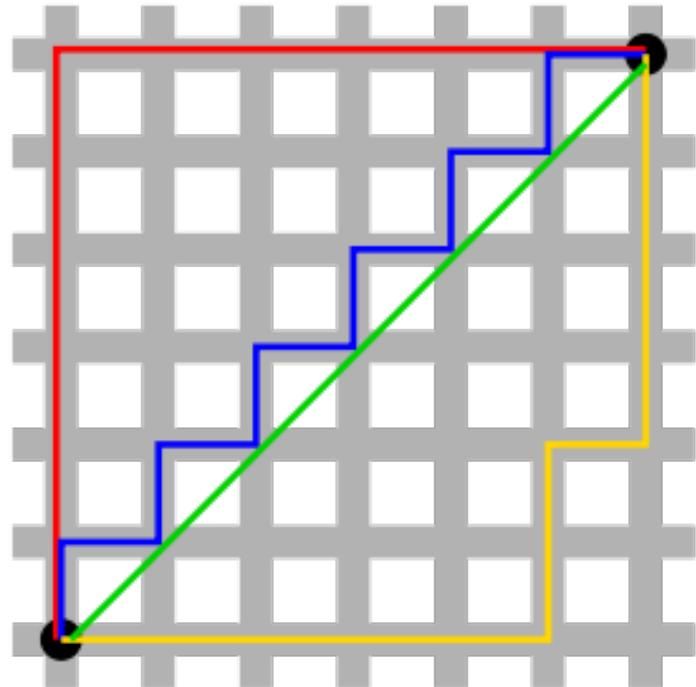
E.g., the following L_1 distance (i.e., city block distance, or Manhattan distance)

$$\text{nn}(x) = \arg \min_{n \in [N]} \|x - x_n\|_1$$

$$\|x - x_n\|_1 = \arg \min_{n \in [N]} \sum_{d=1}^D |x_d - x_{nd}|$$

$$\|x - x_n\|_2 = \sqrt{\sum (x_d - x_{nd})^2}$$

$\|x - x_n\|_1$
 $\|x - x_n\|_2$



Green line is Euclidean distance.
Red, Blue, and Yellow lines are L_1 distance

euclidean-sensitive to outliers
 L_1 -more robust
better HR parameters

How to compute distances with other types of features?

binarizing

map to 3 binary features

euclidian dist.

- subtleties

- preserve data and not visual colors

- Real-valued features (discussion so far).
- Binary features : map to 0 and 1.
 - ▶ Example: Is flower white or colored ?
- Categorical features with V values: map to V -binary features.
 - ▶ Example: Color (white, blue, pink): map to three binary features IsItWhite?, IsItBlue? IsItPink?
 - ▶ Termed **one-hot** encoding.

categorical

white or blue

- looks certain values

w	b	P
0	1	2
001	000	100

we prefer ~~1-hot~~ encoding

1-hot

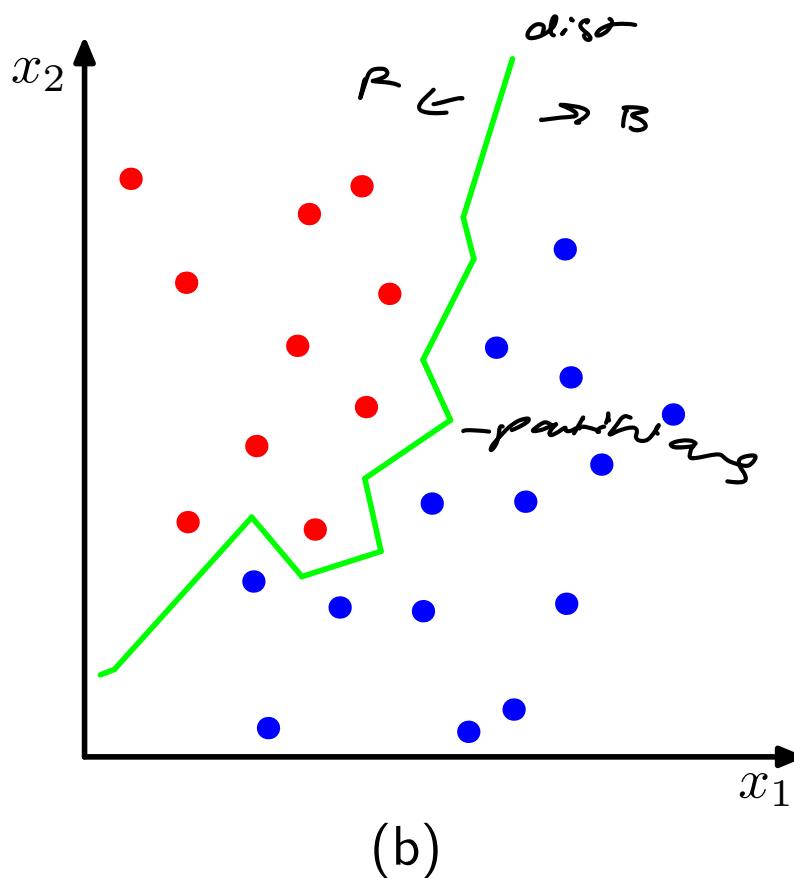


categorical
asymptotic

Decision boundary

1
1st Q) what can we say
about the classifier

For every point in the space, determine its label using the NNC rule. Gives rise to a **decision boundary** that partitions the space into different regions.



Compare to decision boundary of decision trees

K-nearest neighbor (KNN) classification

1 data pt closer to us

Increase the number of nearest neighbors to use?

- 1-nearest neighbor: $\text{nn}_1(\mathbf{x}) = \arg \min_{n \in [N]} \|\mathbf{x} - \mathbf{x}_n\|_2^2$
- 2nd-nearest neighbor: $\text{nn}_2(\mathbf{x}) = \arg \min_{n \in [N] - \text{nn}_1(\mathbf{x})} \|\mathbf{x} - \mathbf{x}_n\|_2^2$
- 3rd-nearest neighbor: $\text{nn}_3(\mathbf{x}) = \arg \min_{n \in [N] - \text{nn}_1(\mathbf{x}) - \text{nn}_2(\mathbf{x})} \|\mathbf{x} - \mathbf{x}_n\|_2^2$
*look at k nearest neighbors
use individual bins*

K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use?

- 1-nearest neighbor: $\text{nn}_1(\mathbf{x}) = \arg \min_{n \in [N]} \|\mathbf{x} - \mathbf{x}_n\|_2^2$
- 2nd-nearest neighbor: $\text{nn}_2(\mathbf{x}) = \arg \min_{n \in [N] - \text{nn}_1(\mathbf{x})} \|\mathbf{x} - \mathbf{x}_n\|_2^2$
- 3rd-nearest neighbor: $\text{nn}_3(\mathbf{x}) = \arg \min_{n \in [N] - \text{nn}_1(\mathbf{x}) - \text{nn}_2(\mathbf{x})} \|\mathbf{x} - \mathbf{x}_n\|_2^2$

The set of K-nearest neighbors

$$\text{knn}(\mathbf{x}) = \{\text{nn}_1(\mathbf{x}), \text{nn}_2(\mathbf{x}), \dots, \text{nn}_K(\mathbf{x})\}$$

Let $\mathbf{x}(k) = \mathbf{x}_{\text{nn}_k(\mathbf{x})}$, then

$$\|\mathbf{x} - \mathbf{x}(1)\|_2^2 \leq \|\mathbf{x} - \mathbf{x}(2)\|_2^2 \dots \leq \|\mathbf{x} - \mathbf{x}(K)\|_2^2$$

How to classify with K neighbors?

Classification rule

- Every neighbor votes: suppose y_n (the label) for \mathbf{x}_n is c , then
 - ▶ vote for c is 1
 - ▶ vote for $c' \neq c$ is 0

We use the *indicator function* $\mathbb{I}(y_n == c)$ to represent.

- Aggregate everyone's vote

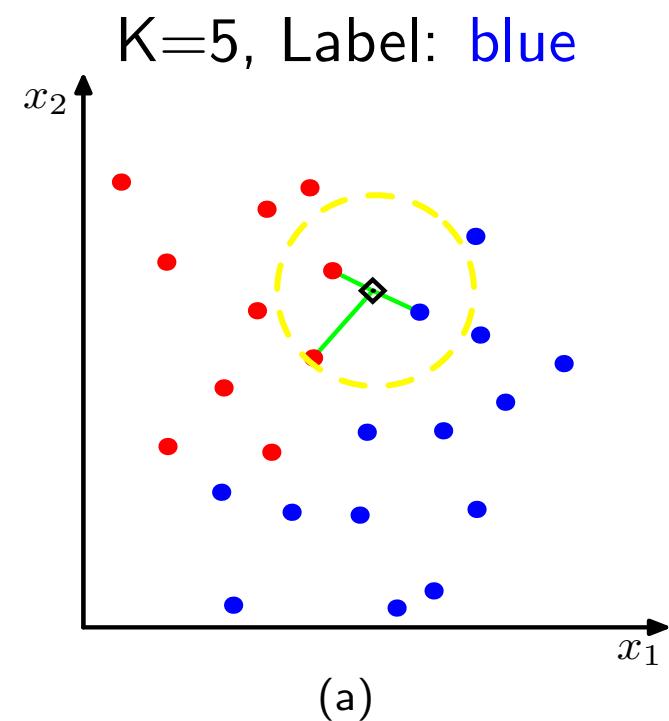
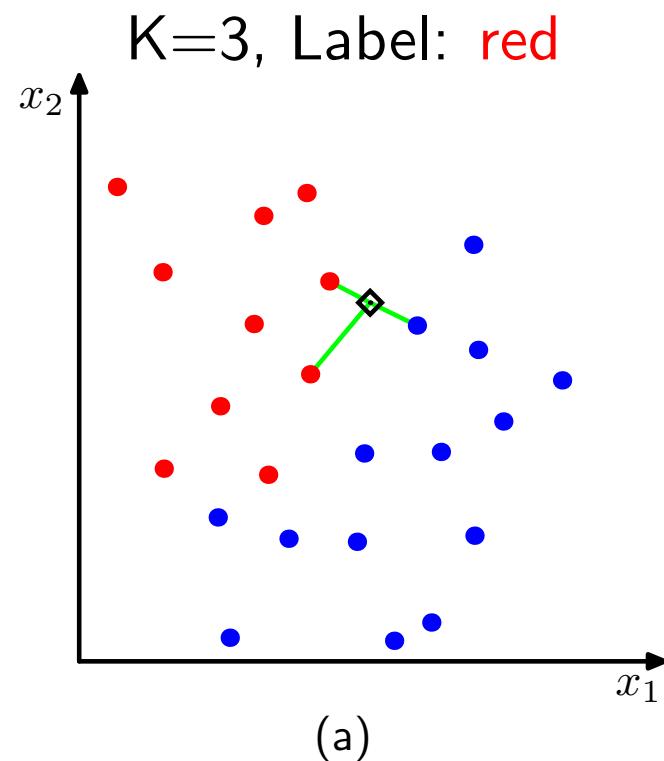
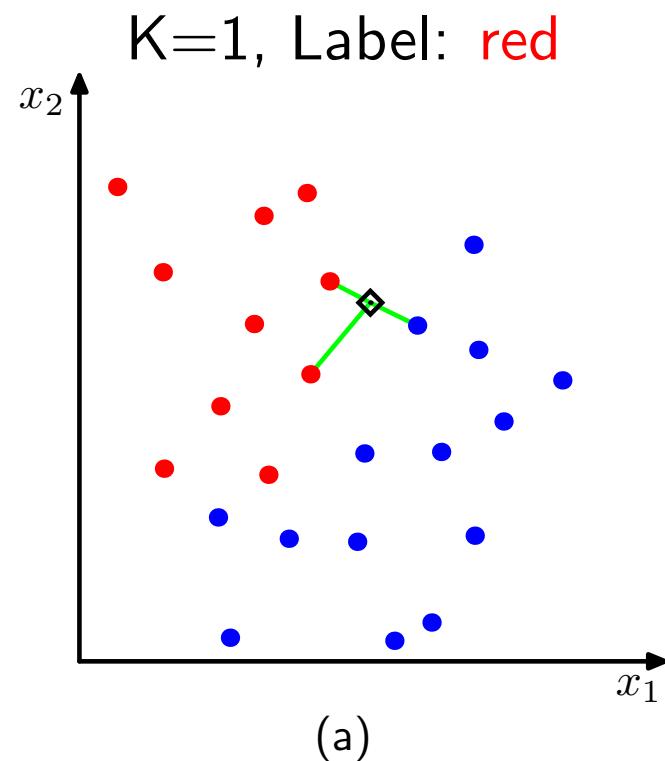
$$\begin{cases} 1 & \text{if } y_n = c \\ 0 & \text{else} \end{cases}$$

$$v_c = \sum_{n \in \text{knn}(\mathbf{x})} \mathbb{I}(y_n == c), \quad \forall c \in [C]$$

- Label with the majority

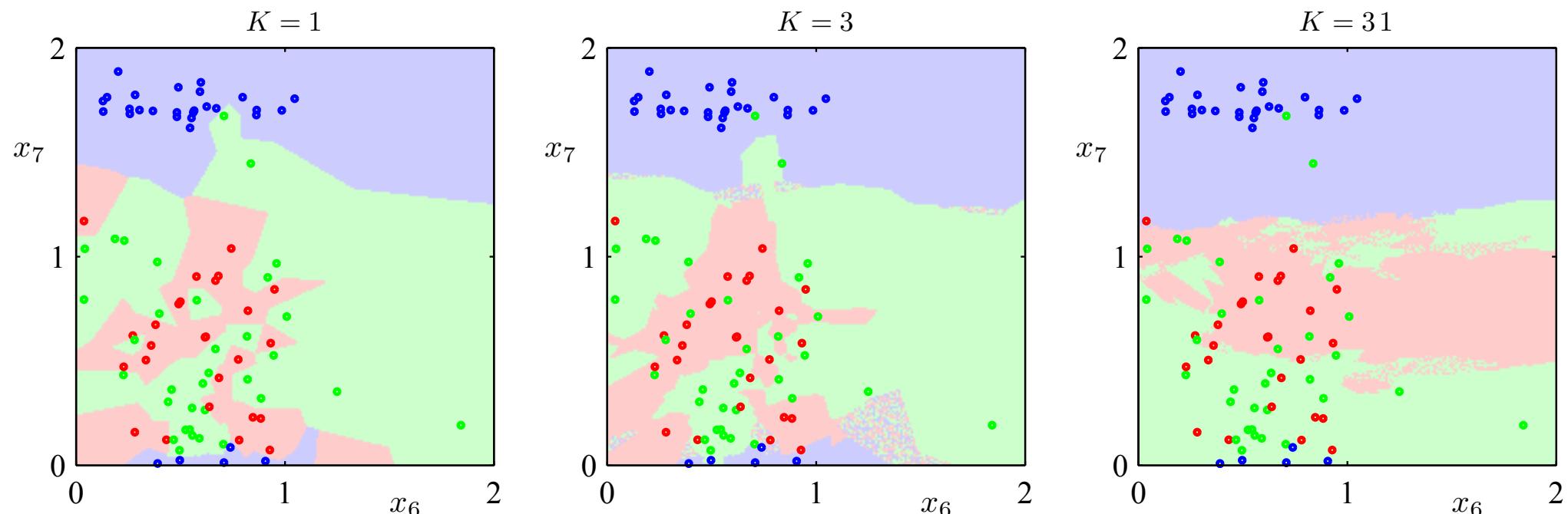
$$\underline{y = h(\mathbf{x})} = \arg \max_{c \in [C]} v_c$$

Example



\Rightarrow complexity ↓

Decision boundary as a function of K



When K increases, the decision boundary becomes smooth.

Mini-summary

Advantages of NNC

- Computationally, simple and easy to implement – just computing the distance

Disadvantages of NNC

- Computationally intensive for large-scale problems: $O(ND)$ for labeling a data point
- We need to “carry” the training data around. Without it, we cannot do classification. This type of method is called *nonparametric*.
- Choosing the right distance measure and K can be involved.

Hyperparameters in NNC

Two practical issues about NNC

- Choosing K , i.e., the number of nearest neighbors (default is 1)
- Choosing the right distance measure (default is Euclidean distance), for example, from the following generalized distance measure

$$\|x - x_n\|_p = \left(\sum_d |x_d - x_{nd}|^p \right)^{1/p}$$

for $p \geq 1$.

$p = 2 \rightarrow$ euclidean

$p = 1 \rightarrow L_1$

Tuning by using a validation dataset

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$
- They are used for learning $h(\cdot)$

Test (evaluation) data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_M, y_M)\}$
- They are used for assessing how well $h(\cdot)$ will do in predicting an unseen $\mathbf{x} \notin \mathcal{D}^{\text{TRAIN}}$

Development (or validation) data

- L samples/instances: $\mathcal{D}^{\text{DEV}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_L, y_L)\}$
- They are used to optimize hyperparameter(s).

Training data, validation and test data should *not* overlap!

Recipe

- For each possible value of the hyperparameter (say $K = 1, 3, \dots, 100$)
 - ▶ Train a model using $\mathcal{D}^{\text{TRAIN}}$
 - ▶ Evaluate the performance of the model on \mathcal{D}^{DEV}
- Choose the model with the best performance on \mathcal{D}^{DEV}
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

What if we do not have validation data?

- Use cross-validation.

Yet, another practical issue with NNC

Assumes all features are equally important!

- Distances depend on units of the features.

$$d(x_1, x_2) = \sqrt{(x_1 - x_{n_1})^2 + (x_2 - x_{n_2})^2}$$

(x_1, x_2)
 $x_1 \text{ cm}$ $x_2 \text{ cm}$

(x_1, x_2)
 $x_1 \text{ mm}$ $x_2 \text{ mm}$
 $x_1 \text{ m}$ $x_2 \text{ m}$
feature

Preprocess data

Normalize data to have zero mean and unit standard deviation in each dimension

- Compute the means and standard deviations in each feature

$$\bar{x}_d = \underbrace{\frac{1}{N} \sum_n x_{nd}}_{\text{mean}}, \quad s_d^2 = \underbrace{\frac{1}{N-1} \sum_n (x_{nd} - \bar{x}_d)^2}_{\text{variance}}$$

- Scale the feature accordingly

$$x_{nd} \leftarrow \frac{x_{nd} - \bar{x}_d}{s_d}$$

Many other ways of normalizing data — you would need/want to try different ones and pick them using (cross)validation

Summary so far

- Decision trees and Nearest Neighbor Classifier: two simple learning algorithm
 - ▶ Used intensively in practical applications.
 - Discussed a few practical aspects, such as tuning hyperparameters, with (cross)validation
-
- You should now be able to use decision trees and nearest neighbors to do machine learning.
 - Given data, use training, development and test splits (or cross-validation).
 - Use training and development to tune hyperparameters that trades off overfitting and underfitting.
 - Use test to get an estimate of generalization or accuracy on unseen data.

Key issues in machine learning

- Modeling
 - ▶ How to formulate the problem ?
- Representation
 - ▶ What is the input/output space ?
 - ▶ What is the model/ hypothesis space?
- Algorithms
 - ▶ How to find the best hypothesis?

learning

Supervised
learning

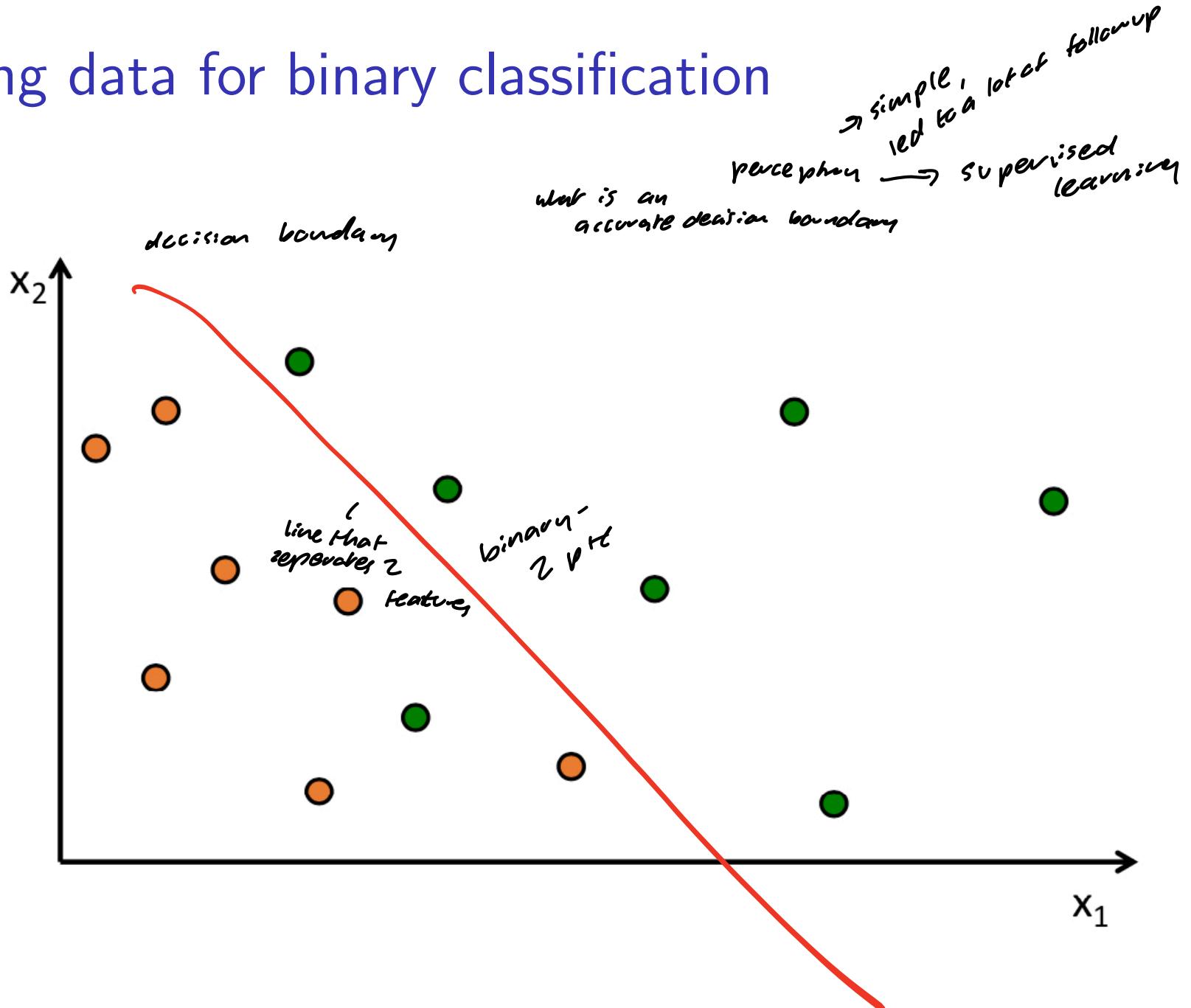
$$h(\underline{x}) \rightarrow *$$

features & labels

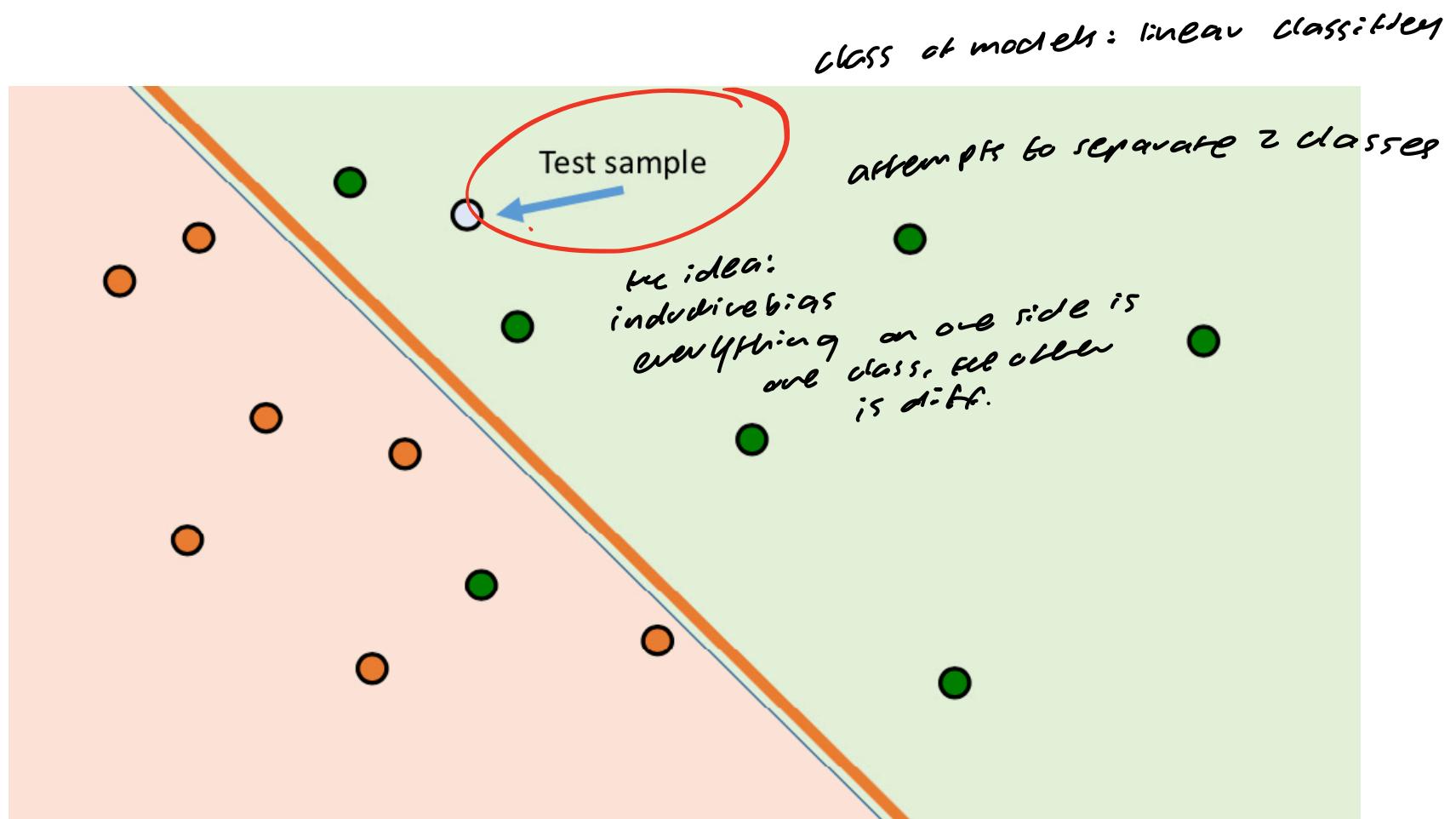
Decision Tree
Nearest neighbor

once we have made a choice
figuring out best hypothesis
search algorithm

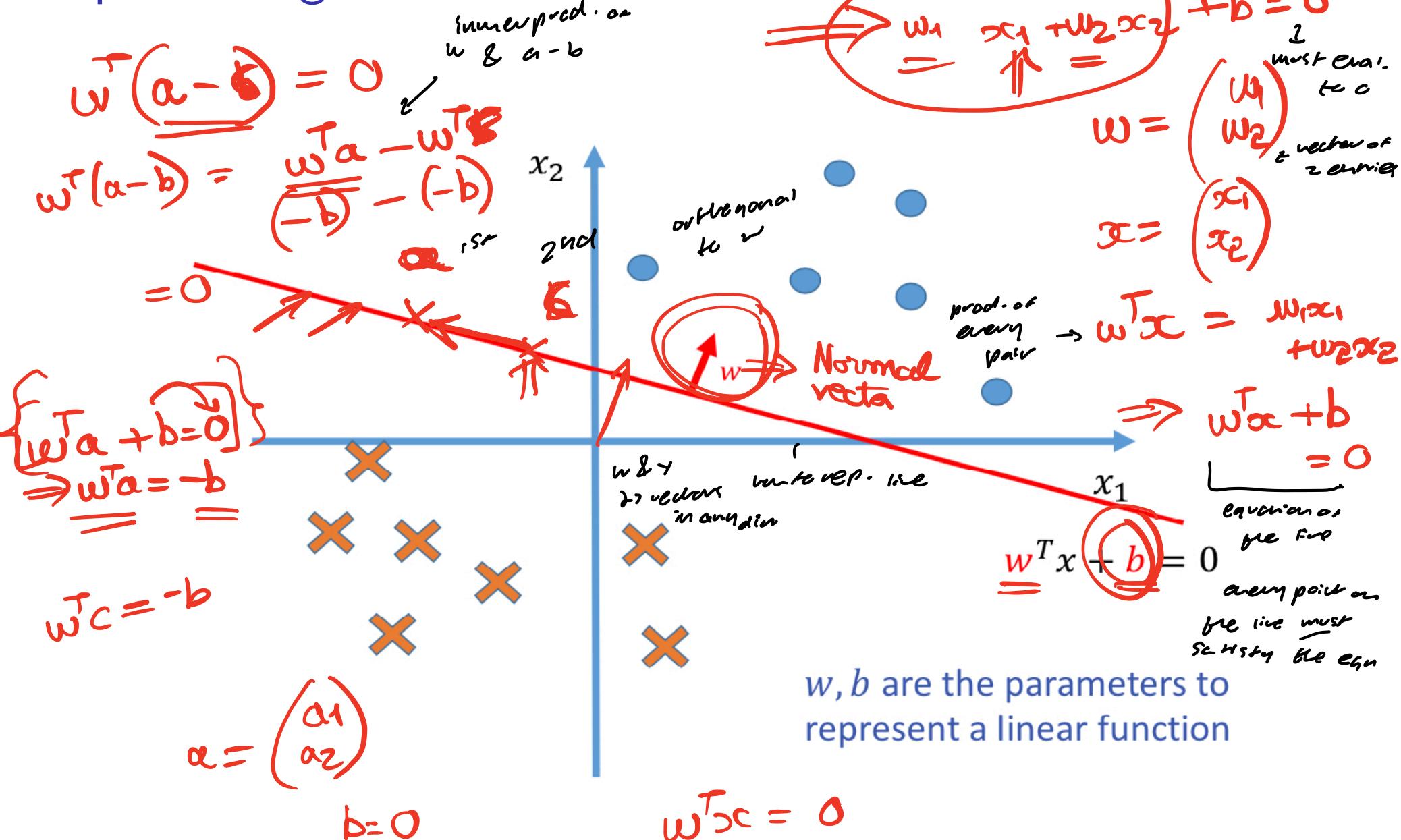
Training data for binary classification



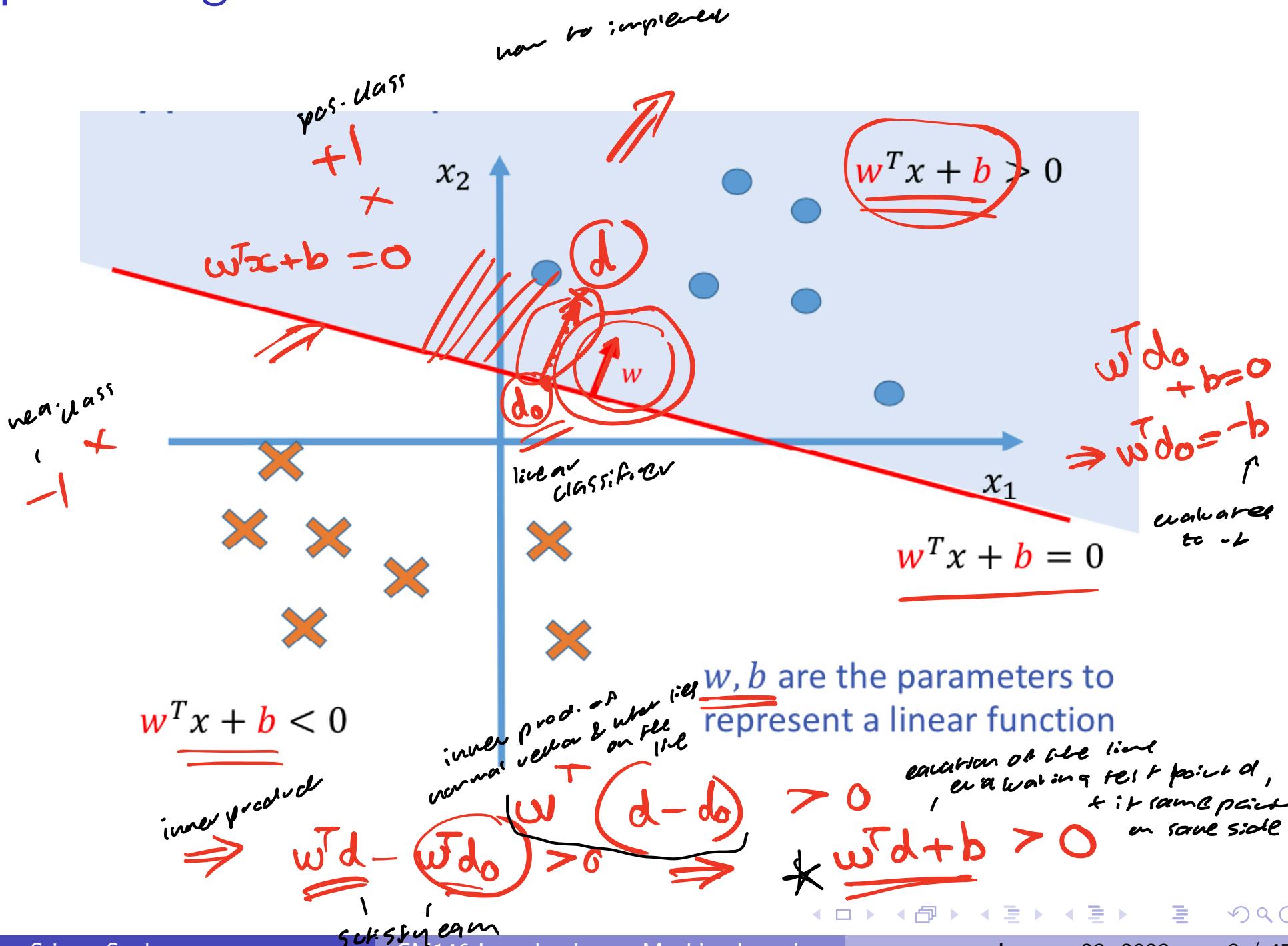
Prediction using a linear classifier



Representing a linear classifier



Representing a linear classifier



Perceptron learning

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ \text{not determined} & \text{if } x = 0 \end{cases}$$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix}$$

sign

Binary classification

- Instance (feature vectors): $x \in \mathbb{R}^D$

- Label: $y \in \{-1, +1\}$

- Model/Hypotheses:

$$H = \{h | h : \mathbb{X} \rightarrow \mathbb{Y}, h(x) = \text{sign}(\sum_{d=1}^D w_d x_d + b)\}.$$

- Learning goal: $\hat{y} = h(x)$

- Learn w_1, \dots, w_D, b .

- Parameters: w_1, \dots, w_D, b .

- w : weights, b : bias

bias

input is a vector x

combine term

1st feature 2nd feature

$(w_1 x_1 + w_2 x_2 + \dots + w_D x_D + b)$

weights

Perceptron predict

- Input: $\underline{x} \in \mathbb{R}^D$, $\underline{w} \in \mathbb{R}^D$, $b \in \mathbb{R}$.
- weights - vector
bias - #
 d dim.

$$a = \sum_{d=1}^D w_d x_d + b = \underline{w}^T \underline{x} + b$$

$\hat{y} = \text{sign}(a) \in \{+1, -1\}$

$x = \begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix}$

take all x_1, \dots, x_D

$$(w_1 x_1 + w_2 x_2 + \dots + w_D x_D + b)$$

- Output: \hat{y} .
- $\sum_{d=1}^D w_d x_d + b = \underline{w}^T \underline{x} + b = 0$: hyperplane in D dimensions with parameters (\underline{w}, b) .
- w : weights, b : bias
- a : activation \rightarrow a is linear-comb. before v compute sign
study how the neuron functions
- $\text{sign}(\sum_{d=1}^D w_d x_d + b)$: Linear Threshold Unit (LTU) - compute linear funcn, +1/-1
— compute sign of the activation

Hyperplanes through the origin

Consider \underline{x} that satisfies $g(\underline{x}) = \underline{w}^T \underline{x} + b = 0$. These \underline{x} define a hyperplane in D dimensions.

We can always write this as a hyperplane passing through the origin in $D + 1$ dimensions. *Follow this hyperplane in D dim.*

can write hyperplane

in $D+1$ dim = 0

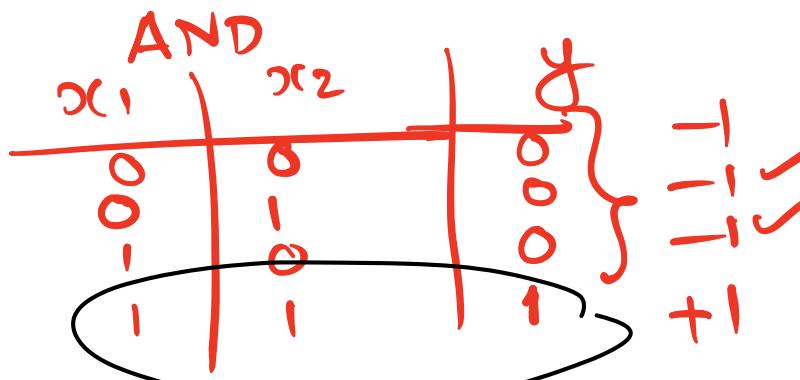
take old weights &
append bias term

$$\begin{aligned}\tilde{\underline{x}} &\equiv \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_D \end{pmatrix} & \tilde{\underline{w}} &\equiv \begin{pmatrix} b \\ w_1 \\ \vdots \\ w_D \end{pmatrix} \\ \tilde{g}(\tilde{\underline{x}}) &= \tilde{\underline{w}}^T \tilde{\underline{x}} = 0 \\ &= \sum_{d=1}^D \underline{w}_d \underline{x}_d + b \\ &= g(\underline{x}) = 0\end{aligned}$$

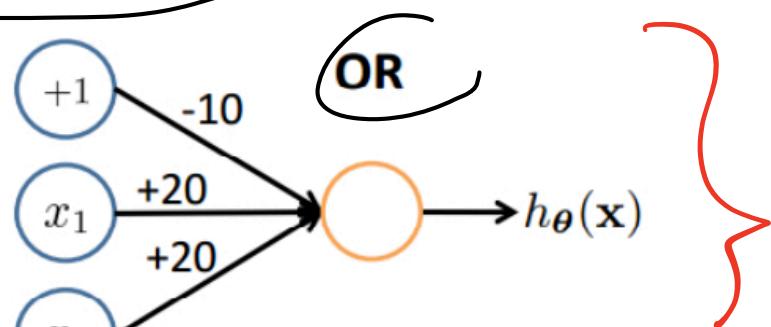
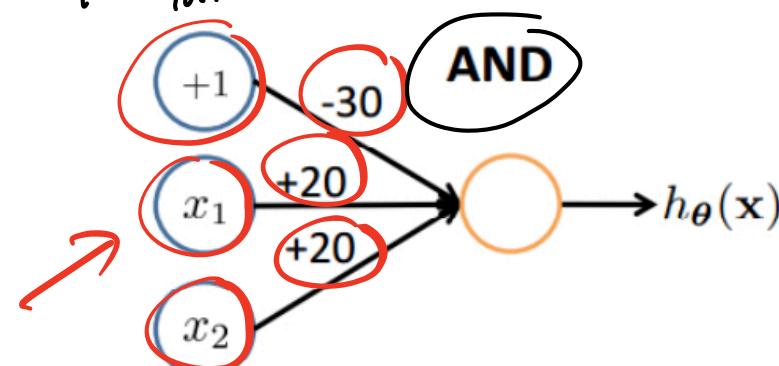
For simplicity, I may write $\tilde{\underline{w}}$ and $\tilde{\underline{x}}$ as w and x when there is no confusion

Representing Boolean functions

what kind of functions



take each input



activation

$$a = 20x_1 + 20x_2 - 30$$

weight
bias

$$y = \text{sign}(a)$$

both inputs 1:

$$x_1=1, x_2=1$$

$$a = 20 + 20 - 30 = 10$$

$$y = +1$$

bias
1

pass through sigmoid tan

inputs

$$x_1=0, x_2=1$$

$$a = 20 - 30 = -10$$

$$y = -1$$

neg. number

$$x_1=0, x_2=0$$

$$a = -30$$

$$y = -1$$

pred: check
val o: get -1

$$a = -10$$

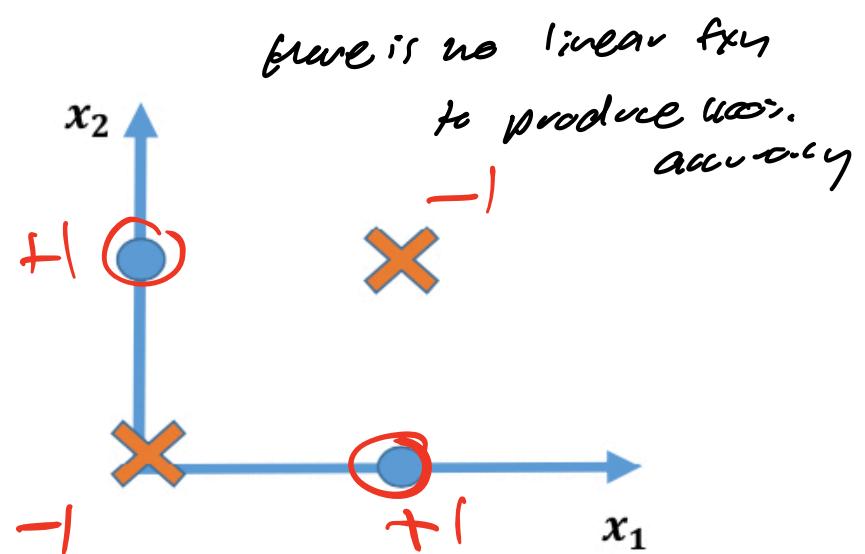
$$y = -1$$

Representing Boolean functions

Can linear model represent XOR?

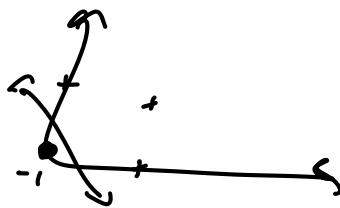
nope, cannot model XNOR

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



XNOR:

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1



Learning a linear classifier

Figure of a linear classifier

Several algorithms

- Perceptron
- Logistic regression
- (Linear) Support Vector Machines

Based on different assumptions

perceptron learning:
learn a linear classifier

mistake
+ correction
= learning

Perceptron learning

If we have only one training example (x_n, y_n) .

Assume $b = 0$. \underline{w}

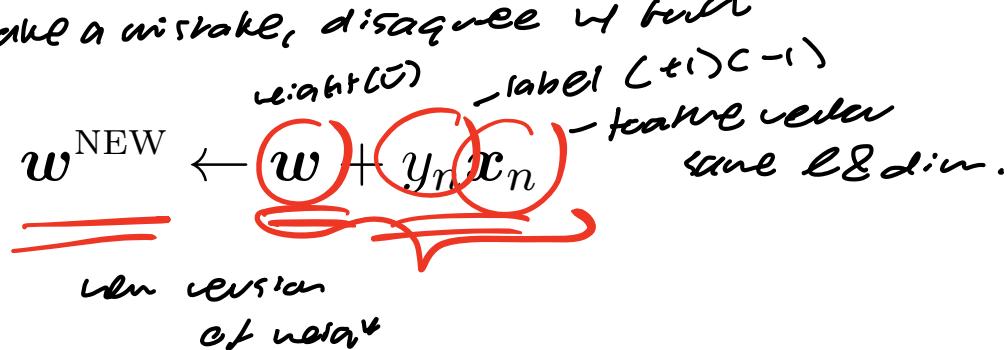
How can we change \underline{w} such that

$$\underline{y_n} = \text{sign}(\underline{\underline{w}}^T \underline{x_n})$$

2 cases:

Two cases

- If $y_n = \text{sign}(\underline{w}^T \underline{x_n})$, do nothing. *pred. matches truth*
- If $y_n \neq \text{sign}(\underline{w}^T \underline{x_n})$, *- make a mistake, disagree w/ truth*



Perceptron learning

If we have only one training example (x_n, y_n) .

Assume $b = 0$.

How can we change w such that

$$y_n = \text{sign}(w^T x_n)$$

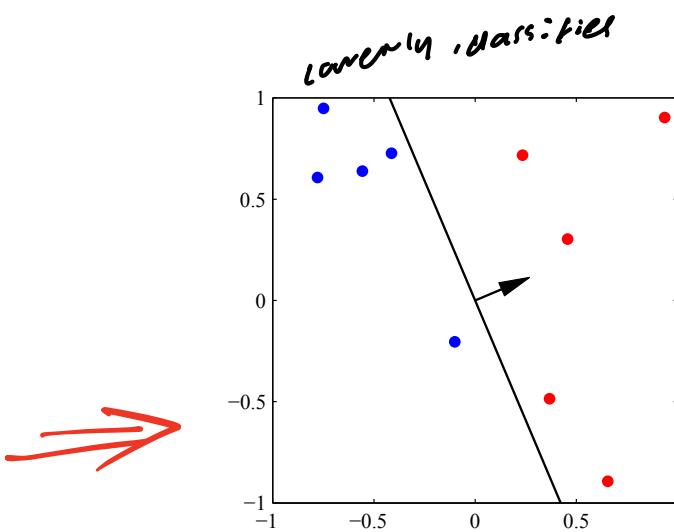
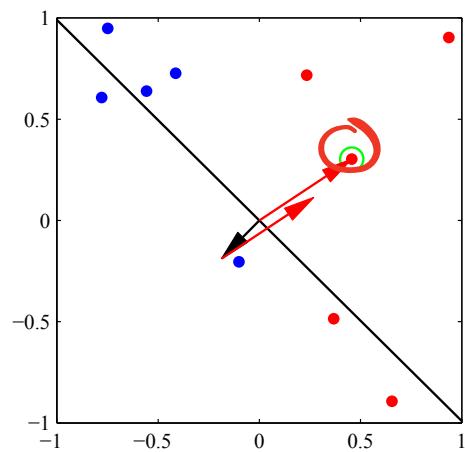
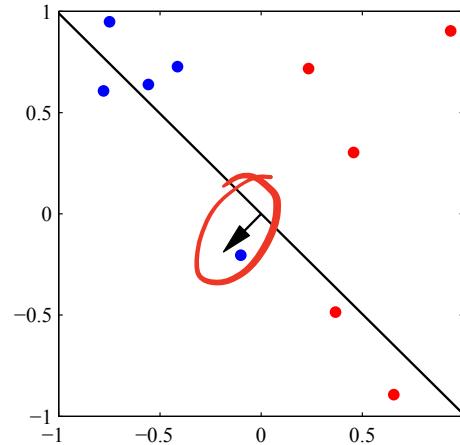
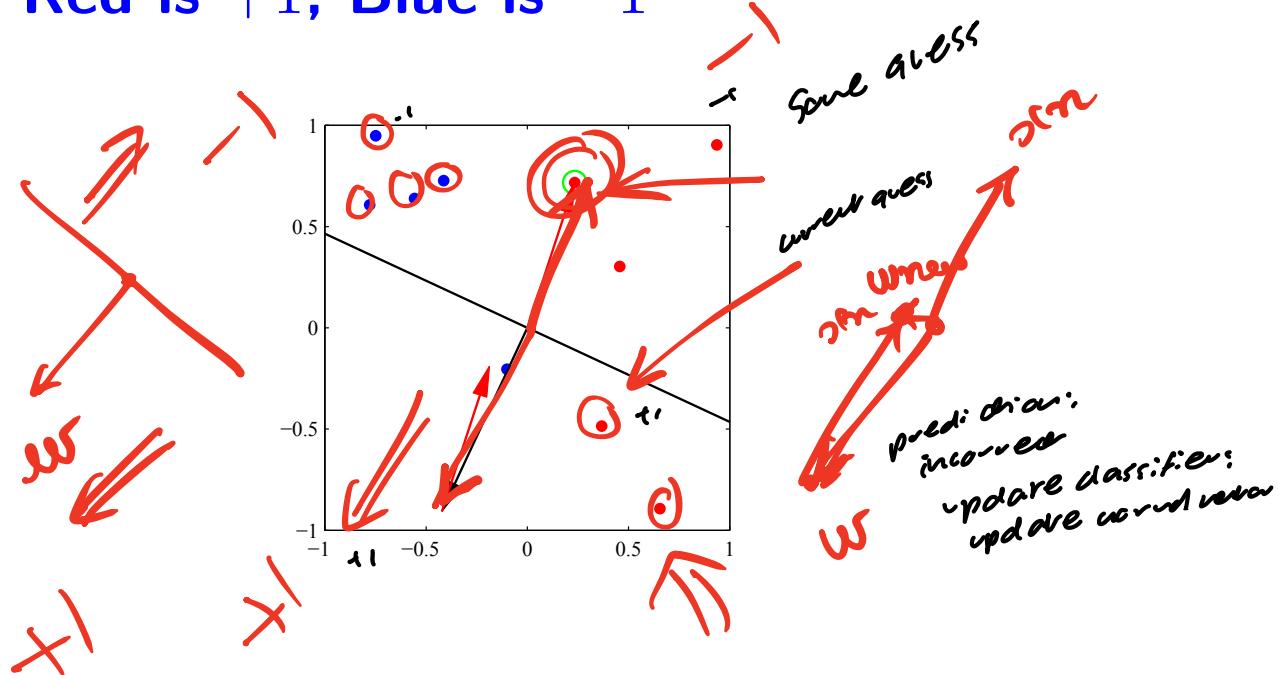
Another way of saying the same thing

- $a = \cancel{w^T x_n}$ activity inner prod.
 - If $y_n a > 0$, do nothing. sign of training prediction
 - If $\cancel{\cancel{y_n a}} \leq 0$,
- $a > 0$ pos $\Rightarrow y_n a > 0$
 $a < 0$ neg $\Rightarrow y_n a < 0$
- $$w^{\text{NEW}} \leftarrow w + y_n x_n$$

Example of perceptron update

Red is +1, Blue is -1

$$w_{\text{new}} \leftarrow w + y_n x_n$$
$$(w + \alpha n)$$
$$y_n = +1$$



Why would it work?

$$\lambda = w^T x_n$$

$$y_n \in \{+1, -1\}$$

If $y_n a \leq 0$, then

$$\underline{y_n(w^T x_n) \leq 0}$$

$$y_n^2 \overset{\text{dim } x_n}{=} x_n^T x_n \geq 0$$

What would happen if we change to new $w^{\text{NEW}} = w + y_n x_n$?

$$y_n [(\underline{w + y_n x_n})^T x_n] = \underbrace{y_n}_{\text{just a scalar}} (\underline{w_{\text{new}}^T x_n})$$

$$= y_n w^T x_n + y_n^2 x_n^T x_n \leq 0$$

$$(\underline{w + y_n x_n})^T x_n =$$

$$= y_n (w^T x_n + (\underline{y_n x_n})^T x_n)$$

$$= y_n (w^T x_n + y_n x_n^T x_n)$$

$$= y_n w^T x_n + \underline{y_n y_n x_n^T x_n}$$

Why would it work?

If $y_n a \leq 0$, then

$$y_n(\mathbf{w}^T \mathbf{x}_n) \leq 0$$

What would happen if we change to new $\mathbf{w}^{\text{NEW}} = \mathbf{w} + y_n \mathbf{x}_n$?

$$y_n[(\mathbf{w} + y_n \mathbf{x}_n)^T \mathbf{x}_n] = y_n \mathbf{w}^T \mathbf{x}_n + y_n^2 \mathbf{x}_n^T \mathbf{x}_n$$

We are adding a positive number, so it is possible that

$$y_n(\mathbf{w}^{\text{NEW}}{}^T \mathbf{x}_n) > 0$$



i.e., we are more likely to classify correctly

Perceptron learning

Iteratively solving one case at a time

- REPEAT
- Pick a data point $\underline{x_n}$
- Compute $a = \underline{w^T x_n}$ using the *current* \underline{w}
- If $\underline{ay_n} > 0$, do nothing. Else,

$$w \leftarrow w + y_n x_n$$

- UNTIL converged. $\cancel{\star}$

i) Not linearly separable

never going to give
perfect accuracy

Perceptron training/learning

data = N samples/instances: = $\{(x_1, y_1), \dots, (x_N, y_N)\}$

Algorithm 1 PerceptronTrain (*data, maxIter*)

```
1:  $w \leftarrow 0$ 
2: for iter = 1 .. MaxIter do
3:   for  $(x, y) \in \text{data}$  do
4:      $b_{\text{or}} - a \leftarrow w^T x$ 
5:     @ activation if  $ay \leq 0$  then
6:        $w \leftarrow w + yx$ 
7:     end if
8:   end for
9: end for
10: return  $w$  sign
```

*upper parameter
for vector to be trained*

*loop through
every training data point*

Prediction: $\text{sign}(w^T x)$.

Design decisions

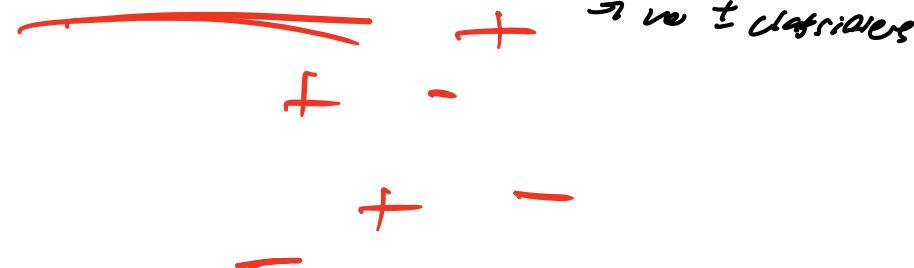
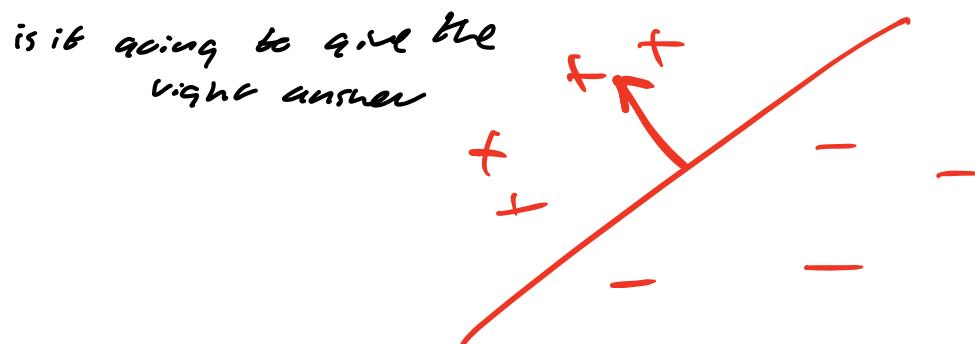
- MaxIter : Hyperparameter
- How to loop over the data?
 - ▶ Constant. ↗ *new to me?*
 - ▶ Permuting once ↙ *can be expensive when data is large*
 - ▶ Permuting in each iteration

Properties of perceptron learning

- This is an **online** algorithm – looks at one instance at a time.
learns after 1 instance at a time
- Does the algorithm terminate (**convergence**)?
small batches - need more

Properties of perceptron learning

- This is an **online** algorithm – looks at one instance at a time.
learns 1 instance & a rule
- Does the algorithm terminate (**convergence**)?
 - ▶ If training data is **not linearly separable**, the algorithm does not converge.
 - ▶ If the training data is linearly separable, the algorithm stops in a finite number of steps (**converges**).
Findign a model how long should I wait even if there exists a linear classifier
- How long to convergence?
never run to infinity - max # of iterations
 - ▶ Depends on the difficulty of the problem (**margin**).
dataset -> no. of classifiers



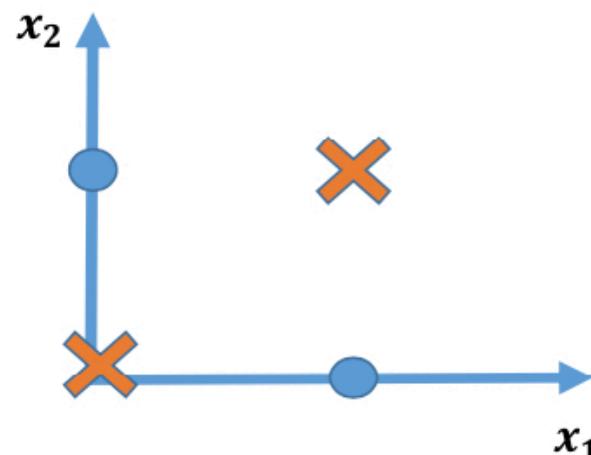
Perceptron learnability

** what is linearly separable
* can't rep.*

Perceptron cannot learn what it cannot represent

- Only linearly separable functions (Minsky and Papert 1969).
- Parity function (XOR) cannot be learned.

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Convergence

assume



Convergence theorem

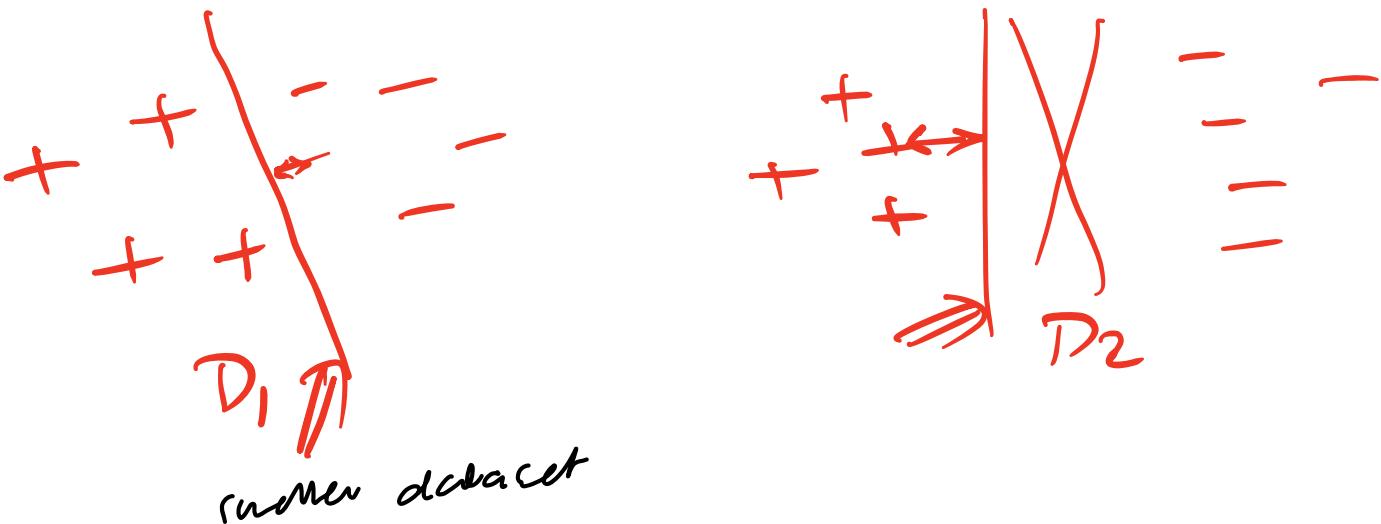
- If the data is linearly separable, the perceptron algorithm will converge after making mistakes that depend on the difficulty of the problem (margin).

Cycling theorem

- If the training data is not linearly separable, then the learning algorithm will eventually repeat the same set of weights and enter an infinite loop

,

Margin



- The margin of a separating hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.
- The margin of a data set is the maximum margin possible for that dataset using any weight vector.

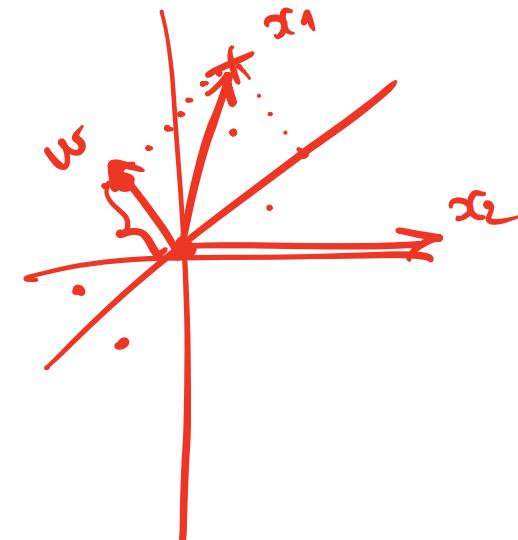
Margin

$$\mathcal{D} = \left\{ \begin{array}{l} (x_1, y_1) \\ \vdots \\ (x_N, y_N) \end{array} \right\}$$

inner prod

$$y_n \mathbf{w}^T \mathbf{x}_n$$

near
vector



$$\text{Margin}(\mathcal{D}, \mathbf{w}) = \begin{cases} \min_{(x,y) \in \mathcal{D}} y_n \mathbf{w}^T \mathbf{x}_n & \text{for a separating hyperplane } \mathbf{w} \\ -\infty & \text{else} \end{cases}$$

$$\text{Margin}(\mathcal{D}) = \sup_w \text{Margin}(\mathcal{D}, \mathbf{w})$$

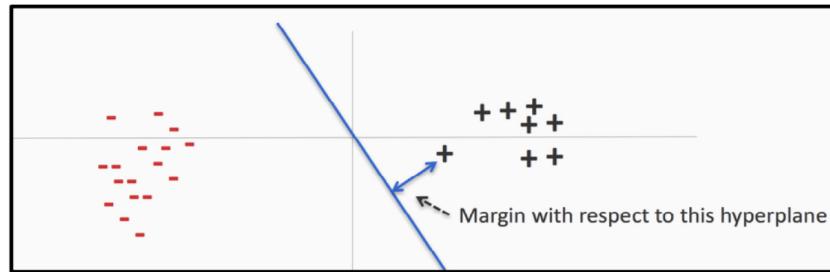
\sup = supremum

$$\mathbf{x}_1^T \mathbf{w}$$

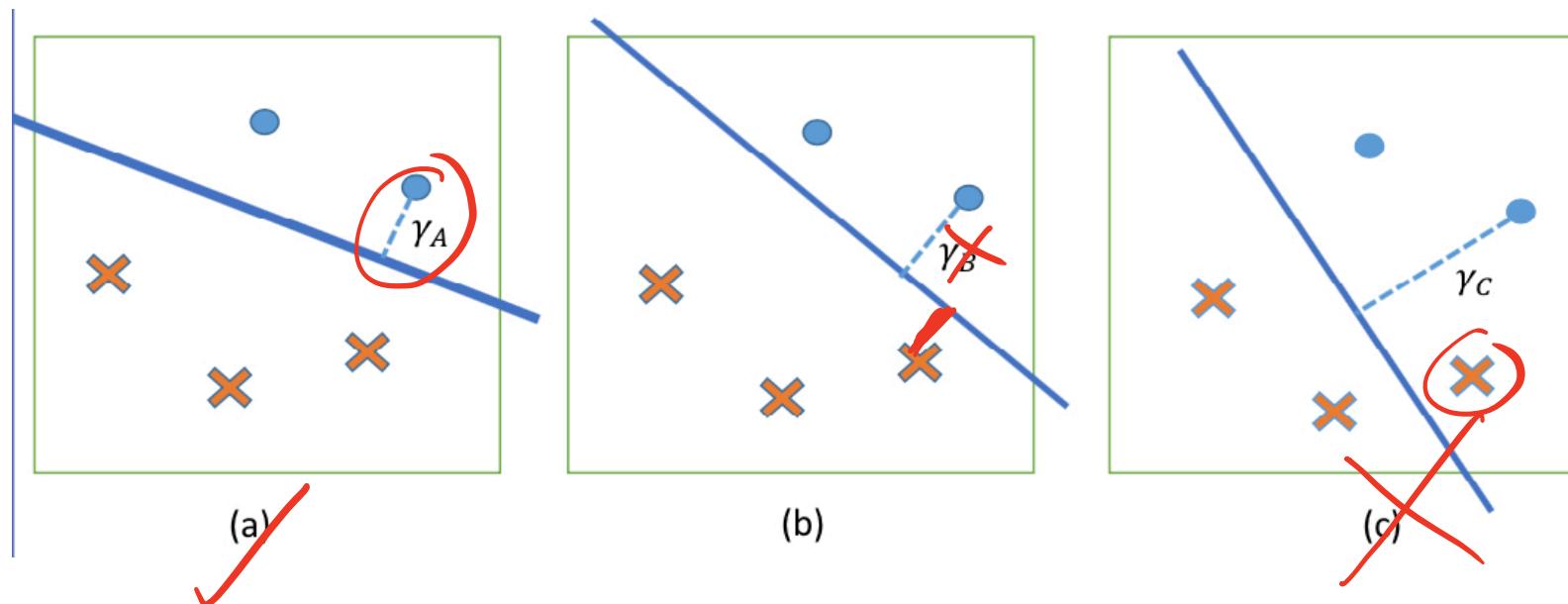
$$\mathbf{w}^T \mathbf{x} = \mathbf{x}^T \mathbf{w}$$

max over set of objects

Margin

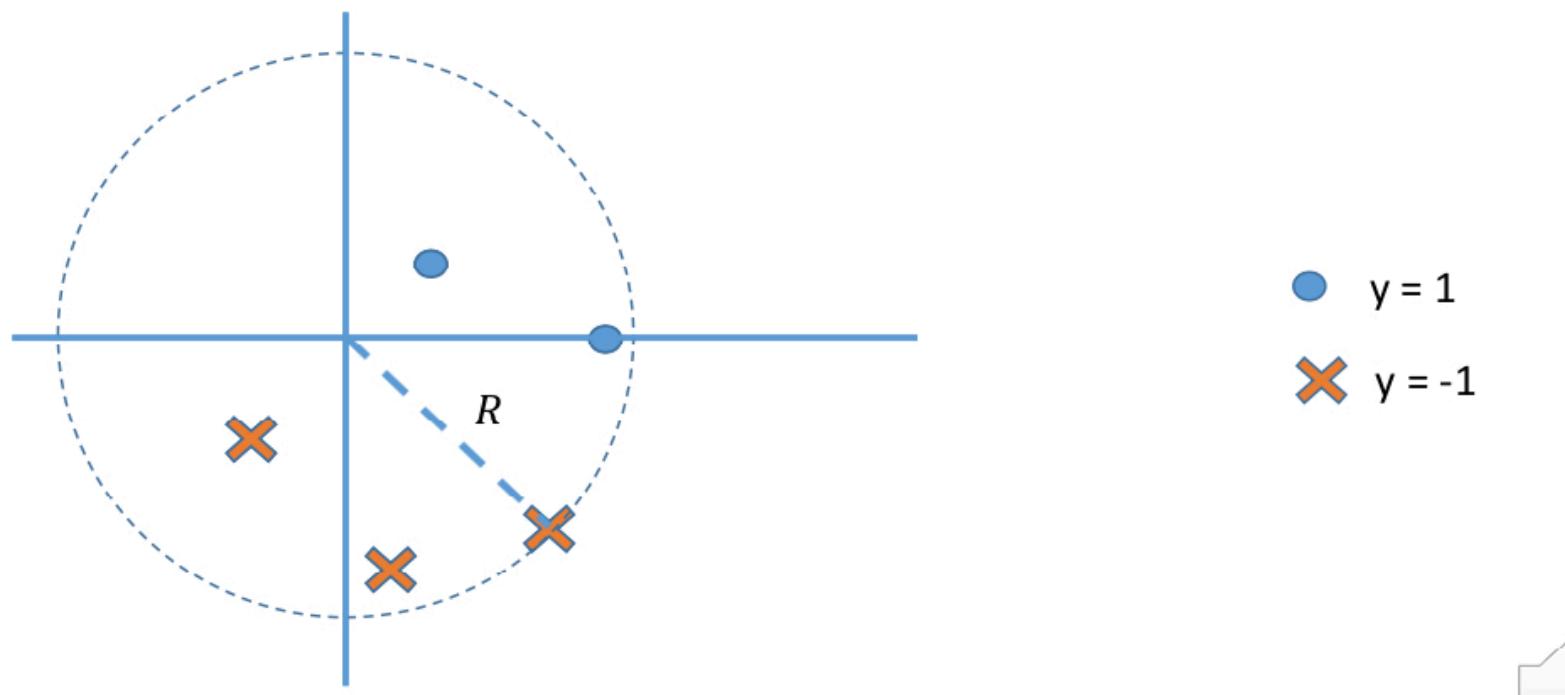


Which γ is the margin of the data?



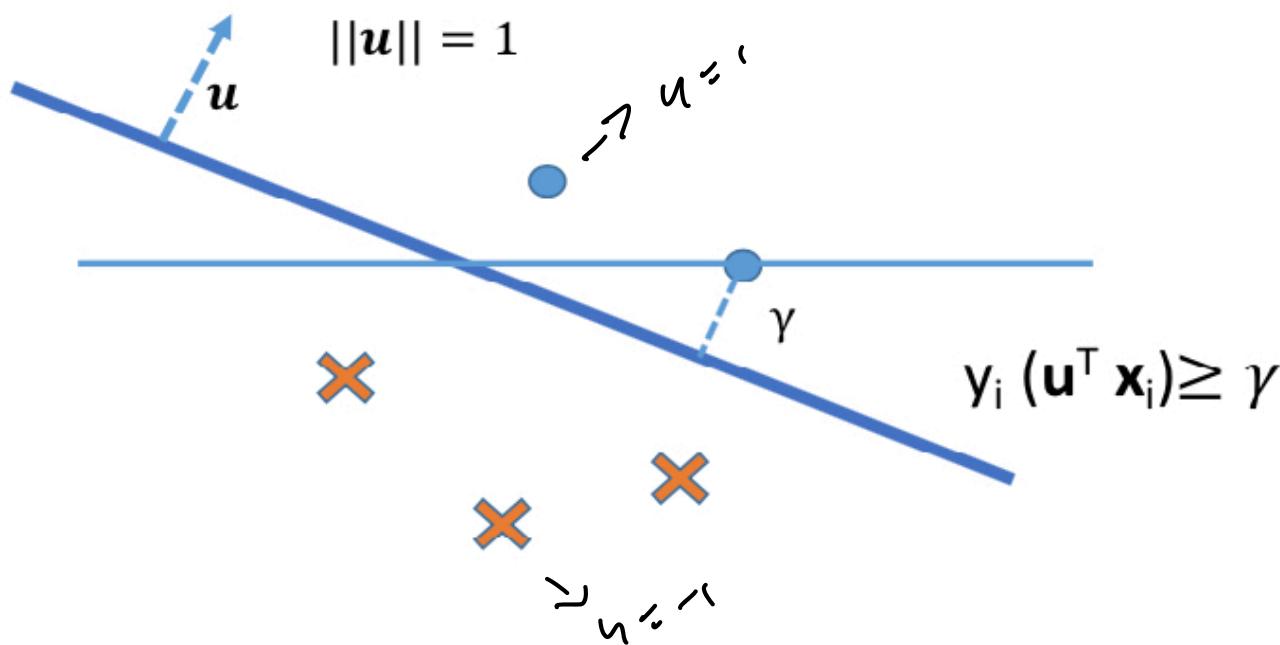
The Mistake Bound Theorem (Novikoff 1962, Block 1962)

- Let $\{(x_1, y_1), \dots, (x_N, y_N)\}$ be a sequence of training examples such that $\|x_n\|_2 \leq R$ and label $y_n \in \{-1, +1\}$.



The Mistake Bound Theorem (Novikoff 1962, Block 1962)

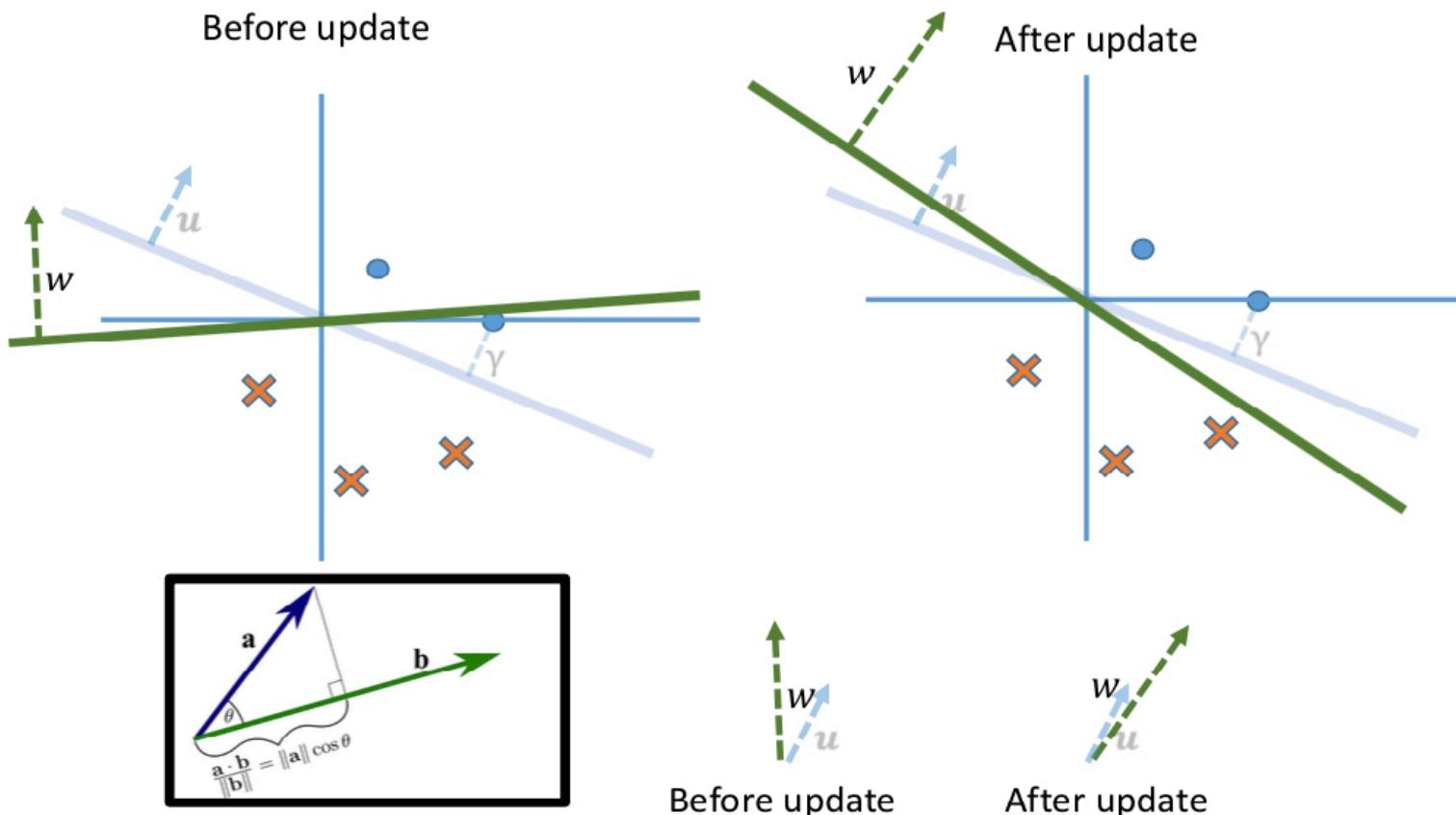
- Let $\{(x_1, y_1), \dots, (x_N, y_N)\}$ be a sequence of training examples such that $\|x_n\|_2 \leq R$ and label $y_n \in \{-1, +1\}$.
- Suppose there exists a unit vector $u \in \mathbb{R}^D$ such that for some $\gamma > 0$, we have $y_n u^T x_n \geq \gamma$.



The Mistake Bound Theorem (Novikoff 1962, Block 1962)

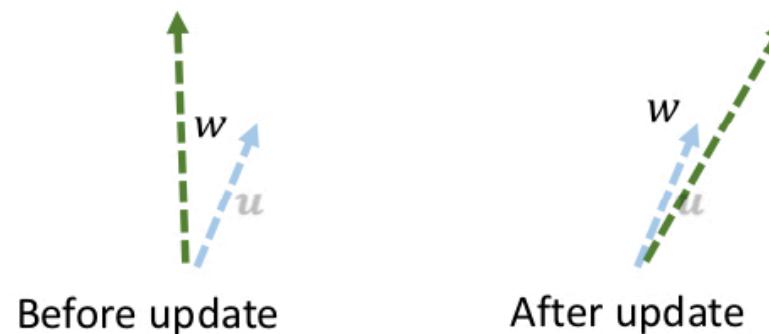
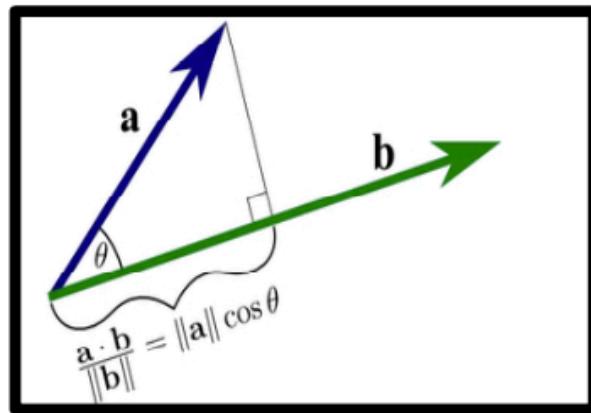
- Let $\{(x_1, y_1), \dots, (x_N, y_N)\}$ be a sequence of training examples such that $\|x_n\|_2 \leq R$ and label $y_n \in \{-1, +1\}$.
- Suppose there exists a unit vector $u \in \mathbb{R}^D$ such that for some $\gamma > 0$, we have $y_n u^T x_n \geq \gamma$.
- Then the Perceptron algorithm will make at most $\frac{R^2}{\gamma^2}$ mistakes on the training sequence.
- If the data is separable....
- then the perceptron algorithm will find a separating hyperplane after making a finite number of mistakes.

Intuition



Intuition

- After update, $u^T w_{t+1}$ is larger than $u^T w_t$.
 - ▶ After t mistakes, $u^T w_t \geq t\gamma$.
- The size of $\|w_{t+1}\|$ may increase but not too much.
 - ▶ After t mistakes, $\|w_t\|^2 \leq tR^2$.



Proof (Preliminaries)

Setting

- Initial weight vector $w_0 = \mathbf{0}$.
- All training examples are contained in a ball of size R .
 $\|x_n\| \leq R$.
- The training data is separable by a margin γ using a unit vector u .
 $y_n u^T x_n \geq \gamma$.

Proof (1/3)

Claim 1: After t mistakes, $\mathbf{u}^T \mathbf{w}_t \geq t\gamma$.

$$\begin{aligned}\mathbf{u}^T \mathbf{w}_{t+1} &= \mathbf{u}^T (\mathbf{w}_t + y_n \mathbf{x}_n) \\ &\geq \mathbf{u}^T \mathbf{w}_t + \gamma\end{aligned}$$

Because $\mathbf{w}_0 = 0$, simple induction gives us: $\mathbf{u}^T \mathbf{w}_t \geq t\gamma$.

```
a ← wTx
if ay ≤ 0 then
    w ← w + yx
```

The inner product between the true underlying model and the current model is non-decreasing after each update. This could be because the directions of \mathbf{w} and \mathbf{u} align or because the length of \mathbf{w} increases.

Proof (2/3)

Claim 2: After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$\begin{aligned}\|\mathbf{w}_{t+1}\|^2 &= \|\mathbf{w}_t + y_n \mathbf{x}_n\|^2 \\ &= \|\mathbf{w}_t\|^2 + 2\mathbf{w}_t^\top (y_n \mathbf{x}_n) + \|y_n \mathbf{x}_n\|^2 \\ &\leq \|\mathbf{w}_t\|^2 + R^2\end{aligned}$$

Because $\mathbf{w}_0 = 0$, simple induction gives us: $\|\mathbf{w}_t\|^2 \leq tR^2$.

Proof (3/3)

What we know

- ① After t mistakes, $\mathbf{u}^T \mathbf{w}_t \geq t\gamma$.
- ② After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

The inner product between the true underlying model and the current model is non-decreasing after each update. This could be because the directions of \mathbf{w} and \mathbf{u} align or because the length of \mathbf{w} increases.

But the length of \mathbf{w} does not increase too much!.

$$R\sqrt{t} \geq \|\mathbf{w}_t\| \quad \text{Number of mistakes } t \leq \frac{R^2}{\gamma^2}.$$

$$R\sqrt{t} \geq \|\mathbf{w}_t\| \geq \mathbf{u}^T \mathbf{w}_t \quad \text{Bounds the total number of mistakes!}$$

$$R\sqrt{t} \geq \|\mathbf{w}_t\| \geq \mathbf{u}^T \mathbf{w}_t \geq t\gamma$$

Beyond the separable case

- Good news
 - ▶ Perceptron makes no assumptions about the data, could be even adversarial.
 - ▶ After a fixed number of mistakes, you are done. Do not need to see any more data.
- Bad news
 - ▶ Real world data is often not linearly separable.

Voting and averaging

- Vanilla perceptron returns final weight vector.
- Might lose good weight vectors that were learned during training.
- Aggregating the models (or weight vectors) seen during training may give better results (especially when data is not separable).

Voted perceptron

- Remember every weight vector in your sequence of updates.
- At final prediction time, each weight vector gets to vote on the label.
- The number of votes it gets is the number of iterations it survived before being updated
- Comes with strong theoretical guarantees about generalization, impractical because of storage issues

Averaged perceptron

- Instead of using all weight vectors, use the average weight vector (i.e longer surviving weight vectors get more say)
- More practical alternative and widely used

Averaged Perceptron training/learning

data = N samples/instances: = $\{(x_1, y_1), \dots, (x_N, y_N)\}$

Algorithm 2 AveragedPerceptronTrain ($data, maxIter$)

```
1:  $w \leftarrow 0.$   $\mu \leftarrow 0.$ 
2: for  $iter = 1 \dots MaxIter$  do
3:   for  $(x, y) \in data$  do
4:      $a \leftarrow w^T x$ 
5:     if  $ay \leq 0$  then
6:        $w \leftarrow w + yx$ 
7:     end if
8:      $\mu \leftarrow \mu + w$ 
9:   end for
10: end for
11: return  $\mu$ 
```

Prediction: $sign(\mu^T x)$.

Perceptron

- Extensions
 - ▶ Voting
 - ▶ Averaging
- Limitations
 - ▶ Linear separability
- Interpreting the importance of features
 - ▶ The values of weight w_d tells us the importance of feature x_d .

Summary

- You should now be able to understand the differences between decision trees, perceptrons and nearest neighbors.
- Given data, use training, development and test splits (or cross-validation).
- Use training and development to tune hyperparameters that trades off overfitting and underfitting.
- Use test to get an estimate of generalization or accuracy on unseen data.

Perceptron learning

Special case: binary classification

each feature vector: D dim.

- Instance (feature vectors): $x \in \mathbb{R}^D$
- Label: $y \in \{-1, +1\}$
- Model/Hypotheses:
linear classifier, lin. comb.
 $H = \{h | h : \mathbb{X} \rightarrow \{-1, +1\}, h(x) = \text{sign}(\sum_{d=1}^D w_d x_d + b)\}.$
- Learning goal: $y = h(x)$
learn one set fun

Perceptron learning

mathematically:
figure out values for these quantities

Special case: binary classification

- Instance (feature vectors): $x \in \mathbb{R}^D$

- Label: $y \in \{-1, +1\}$

- Model/Hypotheses:

$$H = \{h | h : \mathbb{X} \rightarrow \{-1, +1\}, h(x) = \text{sign}(\sum_{d=1}^D w_d x_d + b)\}.$$

- Learning goal: $y = h(x)$

- ▶ Learn w_1, \dots, w_D, b .
- ▶ Parameters: w_1, \dots, w_D, b .
- ▶ w : weights, b : bias

Perceptron predict

- Input: $x \in \mathbb{R}^D$, $w \in \mathbb{R}^D$, $b \in \mathbb{R}$.

input x : feature vector

visualizing perceptron model

$$\xrightarrow{\text{activation (vec1 t)}} a = \sum_{d=1}^D w_d x_d + b = w^T x + b$$
$$\hat{y} = \text{sign}(a)$$

to make binary predictions

- Output: \hat{y} .
- $\sum_{d=1}^D w_d x_d + b$: hyperplane in D dimensions with parameters (w, b) .
- w : weights, b : bias
- a : activation
- $\text{sign}(\sum_{d=1}^D w_d x_d + b)$: Linear Threshold Unit (LTU)

Perceptron learning

Iteratively solving one case at a time

- REPEAT
- Pick a data point x_n
- Compute $a = \mathbf{w}^T \mathbf{x}_n$ using the *current* \mathbf{w}
- If $ay_n > 0$, do nothing. Else,

*make a mistake
-correct model*

$$\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$$

- UNTIL converged.

Properties of perceptron learning

- This is an **online** algorithm – looks at one instance at a time.
- **Convergence**
 - ▶ If training data is **not linearly separable**, the algorithm does not converge.
 - ▶ If the training data is linearly separable, the algorithm stops in a finite number of steps (**converges**).
- How long to convergence ?
 - ▶ Depends on the difficulty of the problem.

↳ ONE AND ONLY HYPOTHESIS THAT PERCEPTRON DESCRIBES DATA SET

Convergence

how to show perceptron
learning algorithm

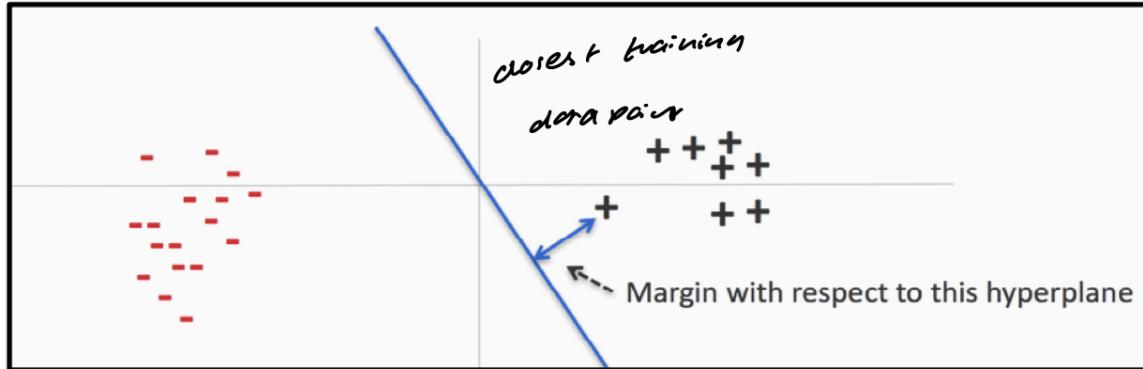
main - classify
difficulty

Convergence theorem

- If the data is linearly separable, the perceptron algorithm will converge after making mistakes that depend on the difficulty of the problem (margin).

it depends on margin

Margin



$$\text{Margin}(\mathcal{D}, \underline{\mathbf{w}}) = \begin{cases} \min_{(x,y) \in \mathcal{D}} y_n \mathbf{w}^T \mathbf{x}_n & \text{for a separating hyperplane } \mathbf{w} \\ -\infty & \text{else} \end{cases}$$

at least 1
misclassified

$$\text{Margin}(\mathcal{D}) = \sup_{\mathbf{w}} \text{Margin}(\mathcal{D}, \underline{\mathbf{w}})$$

define margin

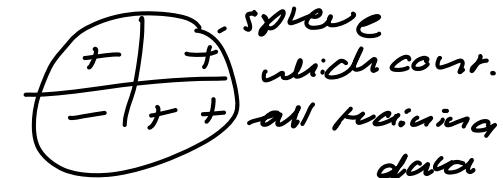
every possible hyperplane

The Mistake Bound Theorem (Novikoff 1962, Block 1962)

convergence theorem

$$\|x_n\|_2 \leq R$$

upper bound



converges after finite # of mistakes

- Let $\{(x_1, y_1), \dots, (x_N, y_N)\}$ be a sequence of training examples such that $\|x_n\|_2 \leq R$ and label $y_n \in \{-1, +1\}$. linearly separable: sole vector u , defines hyperplane $u^T x_n \geq \gamma > 0$, direction
- Suppose there exists a unit vector $u \in \mathbb{R}^D$ such that for some $\gamma > 0$, we have $y_n u^T x_n \geq \gamma$.
- Then the Perceptron algorithm will make at most $\left(\frac{R^2}{\gamma^2}\right)$ mistakes on the training sequence.

$u^T x_n - n^{th}$ data point



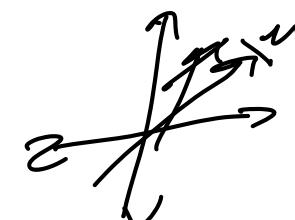
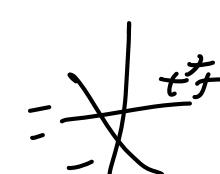
$\left(\frac{R^2}{\gamma^2}\right)$
related to margin

small margin \rightarrow ↑ mistakes

$$\text{For } u^T x_n \quad \frac{\sum_{n=1}^N y_n u^T x_n}{N} = \gamma$$

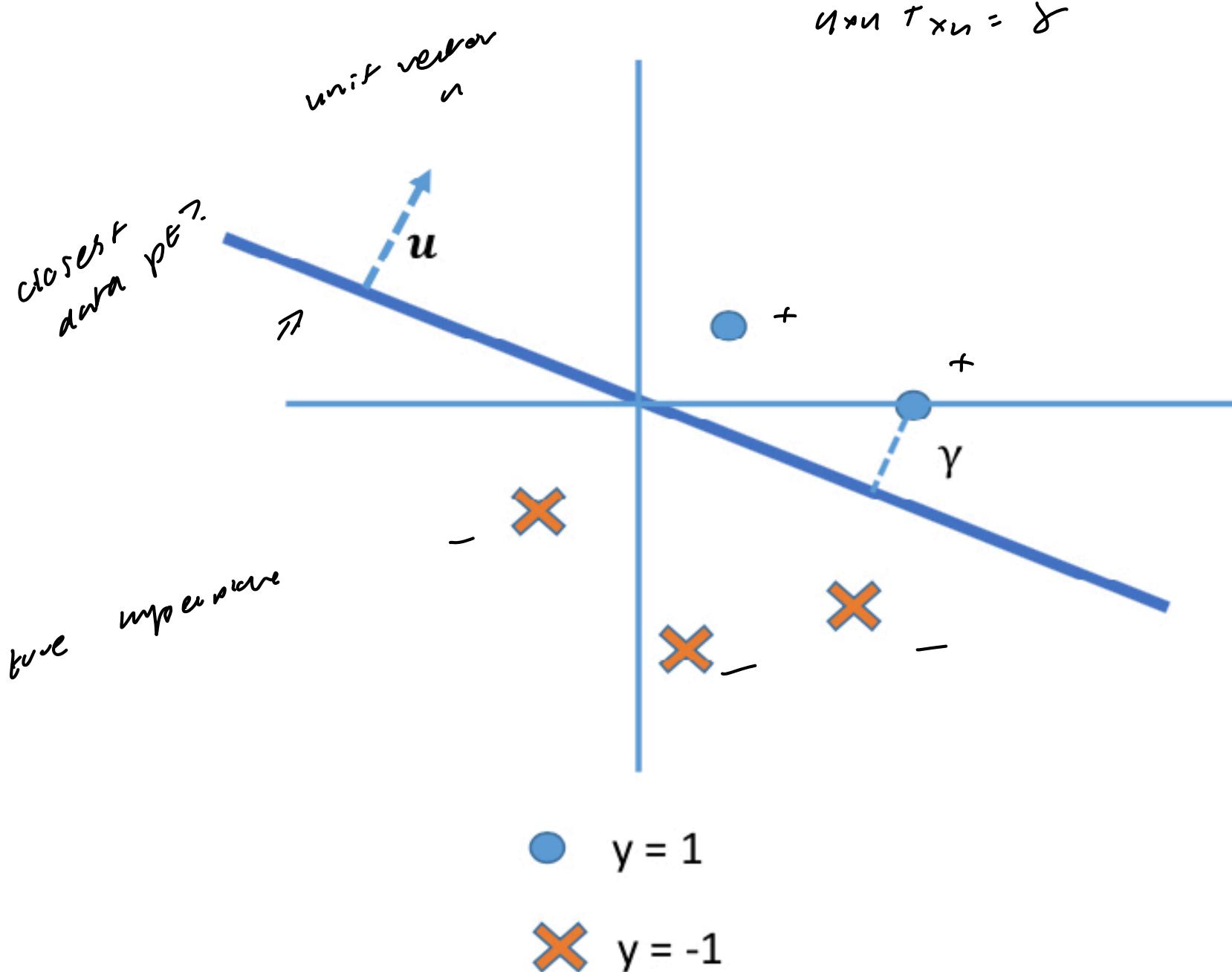
margin
closest dist. to line

hyperplane & training data

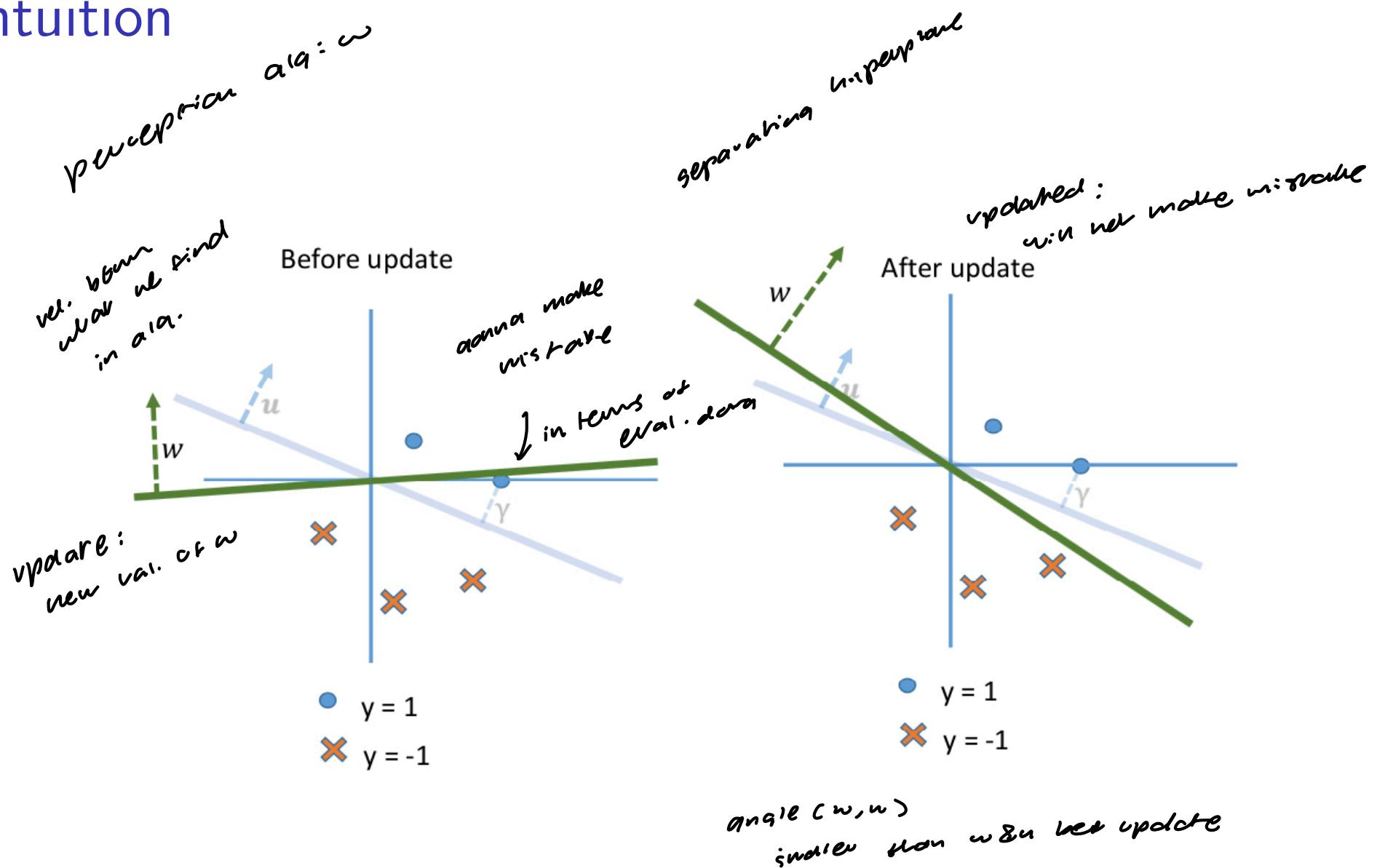


any hyperplane & normal vector

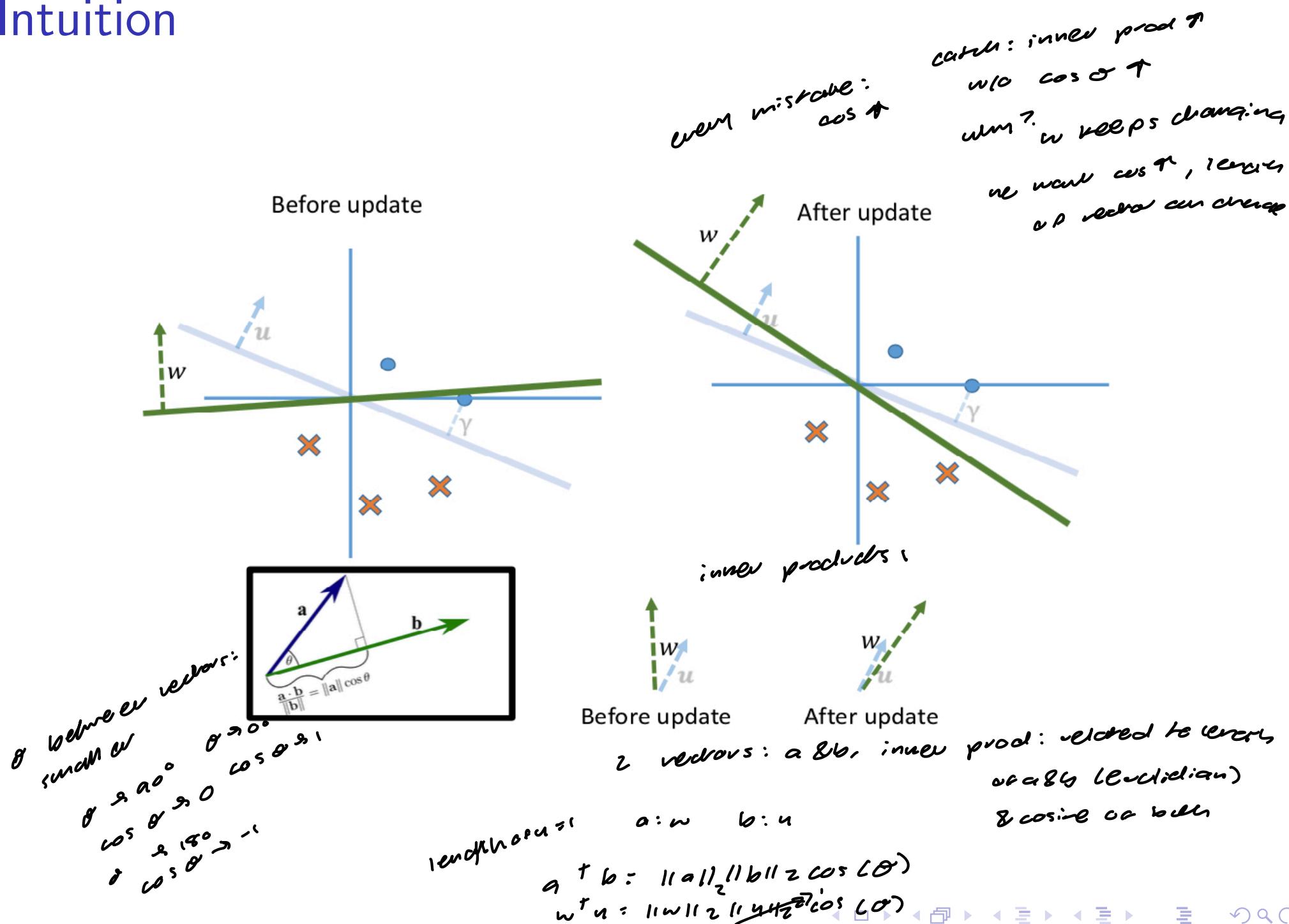
Intuition



Intuition



Intuition

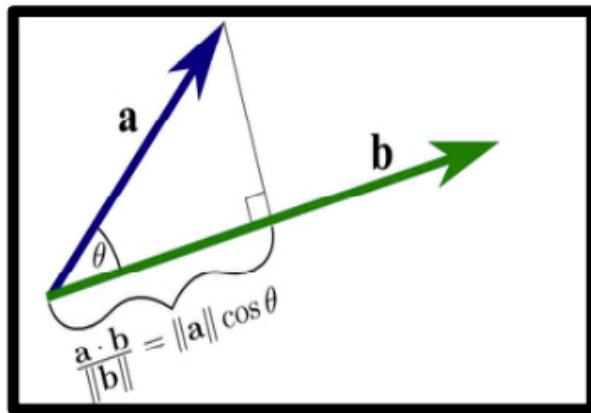


Intuition

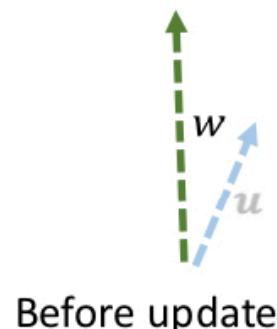
w - weight vector

$$\cos(\theta_t) = u^T w_t$$

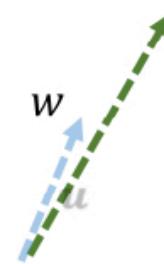
- After update, $u^T w_{t+1}$ is larger than $u^T w_t$
 - After t mistakes, $u^T w_t \geq t\gamma$
- The size of $\|w_{t+1}\|$ may increase but not too much.
 - After t mistakes, $\|w_t\|^2 \leq tR^2$.
*→ never inc.
past this point*



$R^2 -$



Before update



After update

Proof (Preliminaries)

Setting

- Initial weight vector $\underline{w_0 = 0}$. *→ init. to all 0s*
- All training examples are contained in a ball of size R .
 $\|x_n\| \leq R$.
- The training data is separable by a margin γ using a unit vector \underline{u} .
 $y_n \underline{u^T x_n} \geq \gamma$.

Proof (1/3)

Claim 1: After t mistakes, $\mathbf{u}^T \mathbf{w}_t \geq t\gamma$.

$$\begin{aligned}\mathbf{u}^T \mathbf{w}_{t+1} &= \mathbf{u}^T (\mathbf{w}_t + y_n \mathbf{x}_n) \\ &\geq \mathbf{u}^T \mathbf{w}_t + \gamma\end{aligned}$$

Because $\mathbf{w}_0 = 0$, simple induction gives us: $\mathbf{u}^T \mathbf{w}_t \geq t\gamma$.

```
a ← wTx  
if ay ≤ 0 then  
    w ← w + yx
```

Proof (1/3)

*good but not good enough
inner prod. between current & true weight vector
new mistakes are made,*

Claim 1: After t mistakes, $u^T w_t \geq t\gamma$.

$$\begin{aligned} u^T w_{t+1} &= u^T (w_t + y_n x_n) \\ &\geq u^T w_t + \gamma \end{aligned}$$

Because $w_0 = 0$, simple induction gives us: $u^T w_t \geq t\gamma$.

```
a ← w^T x
if ay ≤ 0 then
    w ← w + yx
```

$$\begin{aligned} w_{t+1} &= w_t + y_n x_n && \text{after } t+1 \text{ updates,} \\ u^T w_{t+1} &? u^T w_t && \text{inner prod. is at least} \\ &\xleftarrow{\text{inner prod.}} && \gamma \\ u^T w_{t+1} &= u^T (w_t + y_n x_n) \\ &= u^T w_t + u^T y_n x_n && \xrightarrow{\frac{y_n u^T x_n}{\geq \gamma}} \end{aligned}$$

The inner product between the true underlying model and the current model is non-decreasing after each update. **This could be because the directions of w and u align or because the length of w increases.**

Proof (2/3)

$$\|w_{t+1}\|_2^2 = \|w_t\|_2^2 + 2y_n w_t^T x_n + y_n^2 x_n^T x_n$$

$\underbrace{y_n (w_t^T x_n)}_{\text{if prediction}} \leq 0$

$\leq \|w_t\|_2^2 + 2.0$

$$\|x\|_2^2 = x^T x$$

Claim 2: After t mistakes, $\|w_t\|^2 \leq tR^2$

relative values
of weight vector
to best.
update
 $w_{t+1} = w_t + y_n x_n$

square of error — at least this much
true prod.

$$w_t^T (y_n x_n) = \overbrace{y_n (w_t^T x_n)}$$

$$\begin{aligned} \|w_{t+1}\|^2 &= \|w_t + y_n x_n\|^2 \\ &= \|w_t\|^2 + 2w_t^T (y_n x_n) + \|y_n x_n\|^2 \end{aligned}$$

↑
inner prod.
 $y_n \rightarrow \pm 1$
 $x_n \rightarrow$

$$\begin{aligned} \|w_{t+1}\|_2^2 &= \|w_t + y_n x_n\|_2^2 && \text{— same as euclidean distance} \\ &= (w_t + y_n x_n)^T (w_t + y_n x_n) \\ &= w_t^T w_t + w_t^T (y_n x_n) + y_n x_n^T w_t + (y_n x_n)^T y_n x_n \\ &= \|w_t\|_2^2 + y_n (w_t^T x_n) + y_n (x_n^T x_n) \end{aligned}$$

Because $w_0 = 0$, simple induction gives us: $\|w_t\|^2 \leq tR^2$.

Proof (3/3)

What we know

- ① After t mistakes, $\underline{u^T w_t \geq t\gamma}$.
- ② After t mistakes, $\cancel{\underline{\|w_t\|^2 \leq tR^2}}$

The inner product between the true underlying model and the current model is non-decreasing after each update. This could be because the directions of w and u align or because the length of w increases.

But the length of w does not increase too much!

$$u^T w_t = \|w_t\| \cos \theta$$



$$\|w_t\| \leq \sqrt{t} R$$

arcs as a v never goes quicker than lin. fcn

Proof (3/3)

What we know

- ① After t mistakes, $\mathbf{u}^T \mathbf{w}_t \geq t\gamma$.
- ② After t mistakes, $\|\mathbf{w}_t\|^2 \leq tR^2$

$$\underbrace{R\sqrt{t}}_{t \text{ is a constant}} \geq \|\mathbf{w}_t\| \geq \mathbf{u}^T \mathbf{w}_t \geq t\gamma$$

$$\text{Number of mistakes } t \leq \frac{R^2}{\gamma^2}.$$

Bounds the total number of mistakes!

$$\begin{aligned}\mathbf{u}^T \mathbf{w}_t &= \|\mathbf{w}_t\| \cos \theta \\ &\leq \|\mathbf{w}_t\| \\ \cos(\theta) &\leq 1\end{aligned}$$

$$\begin{aligned}R\sqrt{t} &\geq t\gamma && - \text{always hold}\\ \frac{R}{\gamma} &\geq \sqrt{t} && / \\ \frac{R^2}{\gamma^2} &\geq t && \# \text{mistakes} \\ t &\leq \frac{R^2}{\gamma^2} && t \text{ is bounded by}\end{aligned}$$

Beyond the separable case

linearly separable:

perceptron needs to know & find separable order

real world data:

not linearly separable

- Good news

- ▶ Perceptron makes no assumptions about the data, could be even adversarial.
- ▶ After a fixed number of mistakes, you are done. Do not need to see any more data.

- Bad news

- ▶ Real world data is often not linearly separable.

Voting and averaging

- Vanilla perceptron returns final weight vector.
- Might lose good weight vectors that were learned during training.
- Aggregating the models (or weight vectors) seen during training may give better results (especially when data is not separable).
- Remember every weight vector in your sequence of updates.
- At final prediction time, each weight vector gets to vote on the label.
- The number of votes it gets is the number of iterations it survived before being updated
- Comes with strong theoretical guarantees about generalization, impractical because of storage issues

given $(w^{(t)})_{t=1}^T$ (good weight vectors)
predict a good weight vector

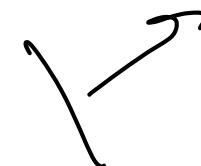
Averaged Perceptron training/learning

data = N samples/instances: = $\{(x_1, y_1), \dots, (x_N, y_N)\}$

Algorithm 1 AveragedPerceptronTrain (*data, maxIter*)

```
1:  $w \leftarrow 0.$   $\mu \leftarrow 0.$ 
2: for iter = 1 ... MaxIter do
3:   for  $(x, y) \in data$  do
4:      $a \leftarrow w^T x$ 
5:     if  $ay \leq 0$  then
6:        $w \leftarrow w + yx$ 
7:     end if
8:      $\mu \leftarrow \mu + w$ 
9:   end for
10: end for
11: return  $\mu$ 
```

- Instead of using all weight vectors, use the average weight vector (i.e longer surviving weight vectors get more say)
- More practical alternative and widely used

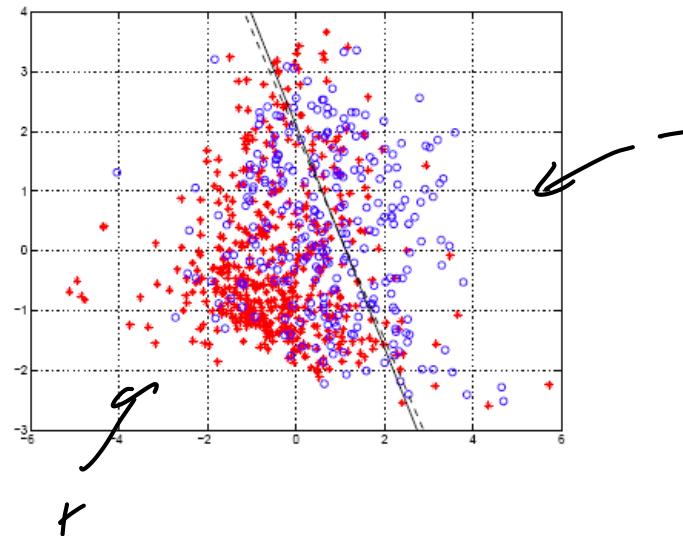


Prediction: $sign(\mu^T x).$

Summary

- Perceptron: linear classification
- Perceptron learning: learning from mistakes
- Perceptron convergence: linear separability and margin
- Variants: voting and averaging.
- See chapter 4 of CIML for reference.

Dealing with lack of separability



- Instead of predicting the class label (-1 vs $+1$), predict the probability of instance being in a class ($P(y = 1|x)$).
- Perceptron does not produce probability estimates

Predict $P(y = 1|x)$

Output a number; i.e. w^Tx

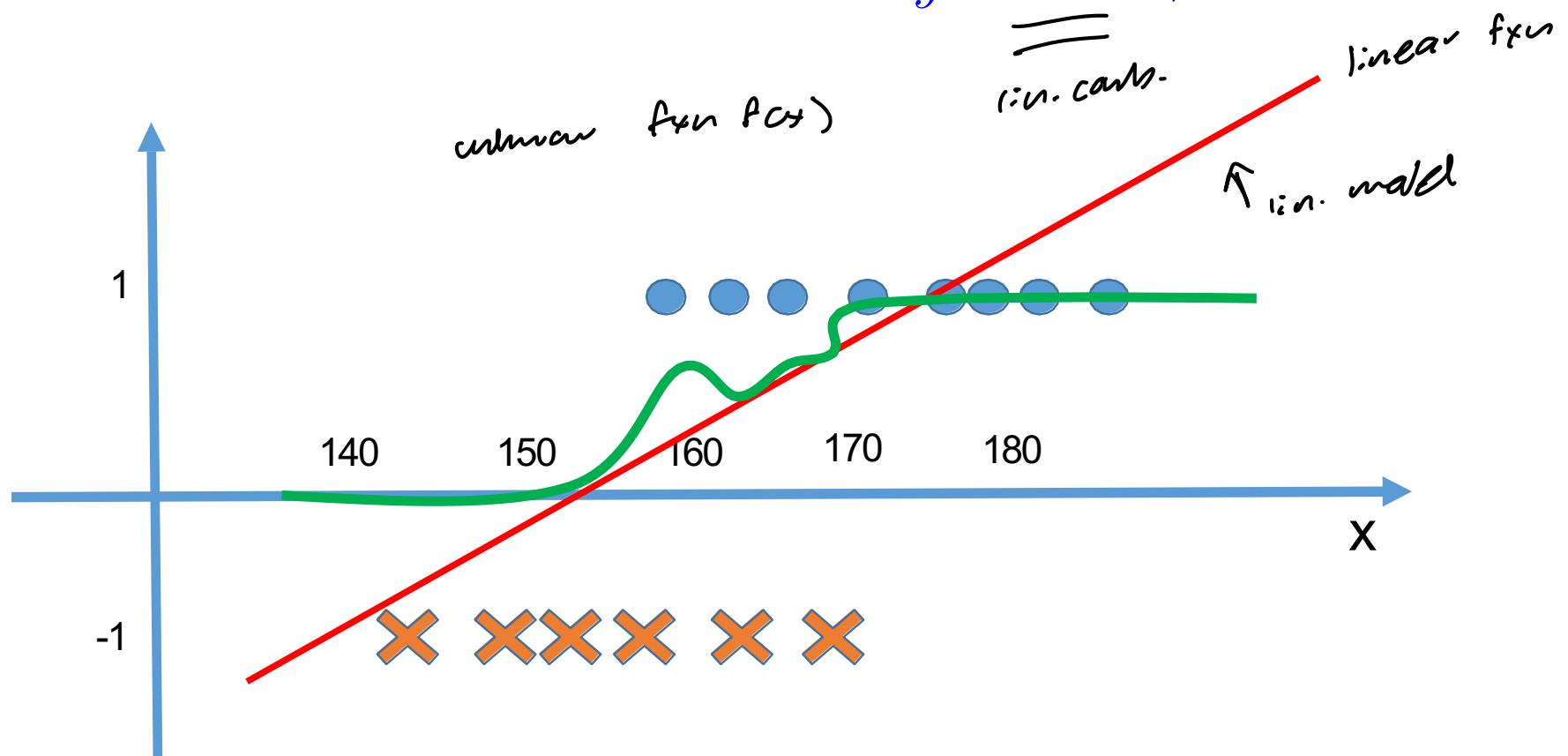
o $\in P(y=1|x) \in [0, 1]$

- Previously, hypothesis space consists of functions that output $\{-1, +1\}$. Now change the hypothesis space to functions that output a value in $[0, 1]$.
- Build a model $h(x) = \sigma(w^T x + b)$ such that $\approx \underbrace{P(y = 1|x)}$.
- Effectively make the problem a regression problem.

How to model $P(y = 1|x)$

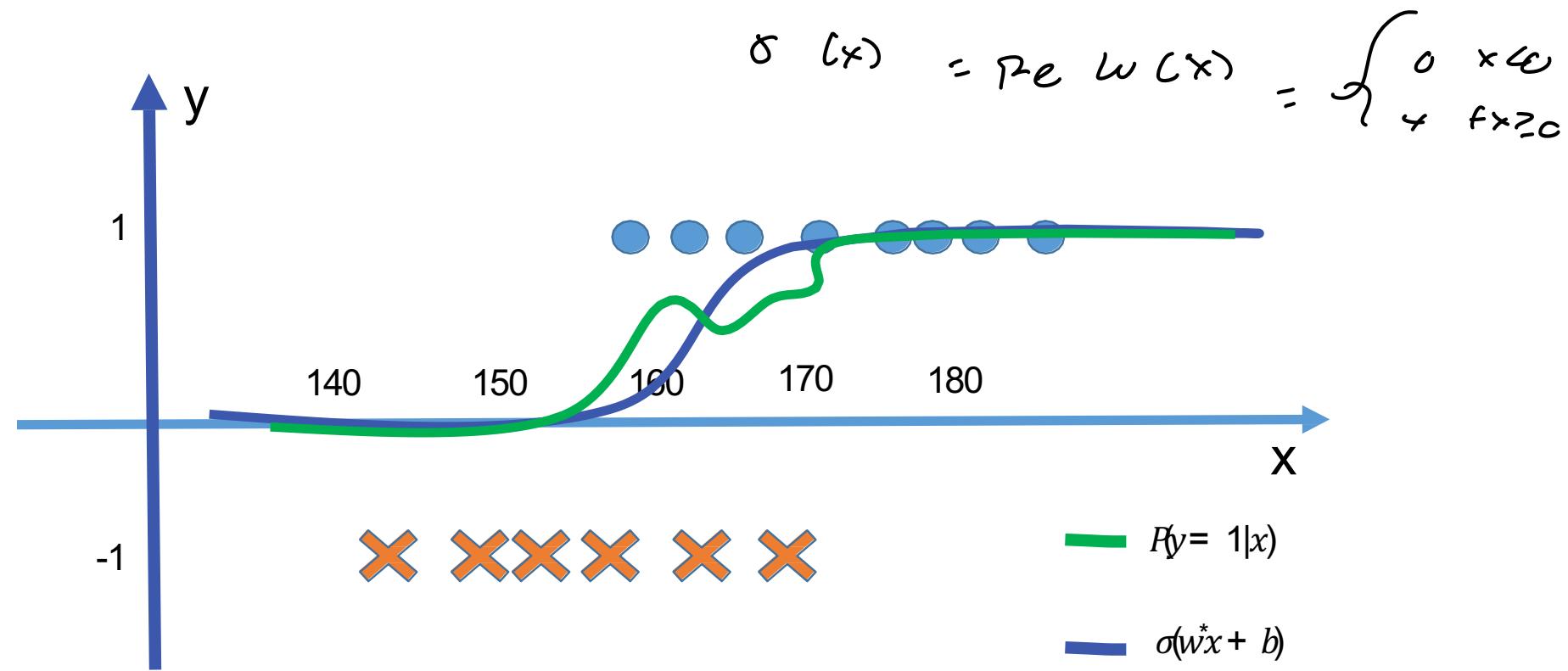
how to model cond. P

Can we use a linear function $y = w^T x + b$?



How to model $P(y = 1|x)$

Define a transformation function $\sigma(w^T x + b) \approx P(y = 1|x)$.



21

Logistic regression

Setup for binary classification

- Input: $\mathbf{x} \in \mathbb{R}^D$
- Output: $y \in \{0, 1\}$ *bin. classif. of input*
- Training data: $\mathcal{D} = \{(\mathbf{x}_n, y_n), n = 1, 2, \dots, N\}$
- Hypotheses/Model:

$$h_{\mathbf{w}, b}(\mathbf{x}) = p(y = 1 | \mathbf{x}; b, \mathbf{w}) = \sigma(\underline{a(\mathbf{x})})$$

weight vector.

where

$$a(\mathbf{x}) = b + \sum_d w_d x_d = b + \mathbf{w}^T \mathbf{x}$$

and $\sigma(\cdot)$ stands for the sigmoid function — *transformation*

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Why the sigmoid function?

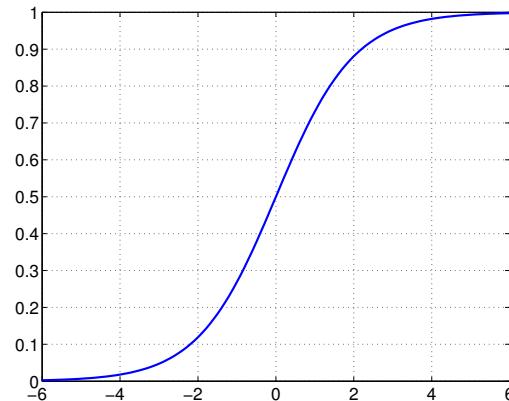
$$\frac{1}{1+e^{-1000}}$$

What does it look like?

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

where

$$a = b + \mathbf{w}^T \mathbf{x}$$



Properties

- Bounded between 0 and 1 ← thus, interpretable as probability
- Monotonically increasing thus, usable to derive classification rules
 - ▶ $\sigma(a) > 0.5$, positive (classify as '1')
 - ▶ $\sigma(a) < 0.5$, negative (classify as '0')
 - ▶ $\sigma(a) = 0.5$, undecidable
- Nice computational properties As we will see soon

The hypothesis space for logistic regression

All functions of the form

$$h_{\mathbf{w},b}(x) = \sigma(\mathbf{w}^T \mathbf{x} + b)$$

A linear function composed with a sigmoid function.

We want to find a model $h_{\mathbf{w},b}(x)$ such that

$$h_{\mathbf{w},b}(x) \approx p(y = 1 | \mathbf{x})$$

Mini-summary

$$P(y=0|x) = 1 - P(y=1|x)$$

- What is the goal of logistic regression ?
 - ▶ Model $P(y=1|x)$
- What is the hypothesis space ?
 - ▶ $H = \{h|h : \mathbb{X} \rightarrow [0, 1], h(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + b)\}$

Prediction in logistic regression

$$P(y=1|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + b) = \frac{1}{1 + \exp^{-(\mathbf{w}^T \mathbf{x} + b)}}$$

Compute $\sigma(\mathbf{w}^T \mathbf{x} + b)$. If this is greater than 0.5, predict 1 else 0.

Decision boundary: Linear or nonlinear?

$\sigma(a)$ is **nonlinear**, however, the decision boundary is determined by

$$\sigma(a) = 0.5 \Rightarrow a(\mathbf{x}) = 0 \Rightarrow a(\mathbf{x}) = b + \mathbf{w}^T \mathbf{x} = 0$$

which is a *linear* function in \mathbf{x}

As in the case of perceptron, b the **bias or offset or intercept term**.

\mathbf{w} the **weights** .

Logistic regression

Setup for binary classification

- Input: $\mathbf{x} \in \mathbb{R}^D$
- Output: $y \in \{0, 1\}$
- Training data: $\mathcal{D} = \{(\mathbf{x}_n, y_n), n = 1, 2, \dots, N\}$
- Hypotheses/Model:

$$h_{\mathbf{w}, b}(\mathbf{x}) = p(y = 1 | \mathbf{x}; b, \mathbf{w}) = \sigma(a(\mathbf{x}))$$

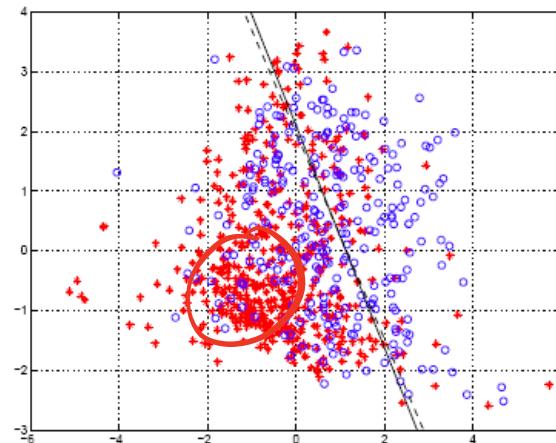
where

$$a(\mathbf{x}) = b + \sum_d w_d x_d = b + \mathbf{w}^T \mathbf{x}$$

- Given training data N samples/instances:
 $\mathcal{D}^{\text{TRAIN}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$, train/learn/induce $h_{\mathbf{w}, b}$.
Find values for (\mathbf{w}, b) .

Review

- Instead of predicting the class, predict the probability of instance being in a class
- Perceptron does not produce probability estimates



Prediction in logistic regression

input x - feature vector

$$x \in \mathbb{R}^D \quad w \in \mathbb{R}^D$$

y in one of 2 classes

linear comb.

$$P(y = 1|x) = \sigma(w^T x + b) = \frac{1}{1 + \exp^{-(w^T x + b)}}$$

Compute $\sigma(w^T x + b)$. If this is greater than 0.5, predict 1 else 0.

binary decision

< 0.5 - class 0

> 0.5

Decision boundary: Linear or nonlinear?

① look at decision boundary
linear or nonlinear?

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

$a=0$

sigmoid

$$\sigma(w^T x + b)$$

lies between 0 & 1

?

hyperplane
 \rightarrow 2/s decision

$$\begin{aligned} > 0.5 &\Rightarrow 1 \\ = 0.5 & \\ < 0.5 &\Rightarrow 0 \end{aligned}$$

Perception

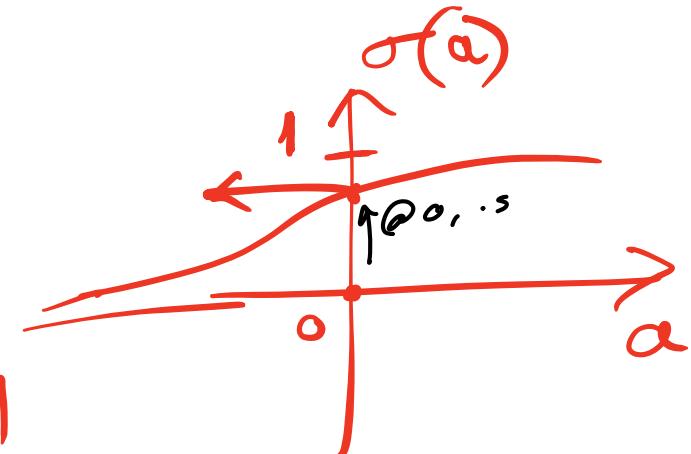
use sign for pred.
lin. comb.

$$\text{sign}(w^T x + b) = +1$$

$w^T x + b = 0$
hyperplane

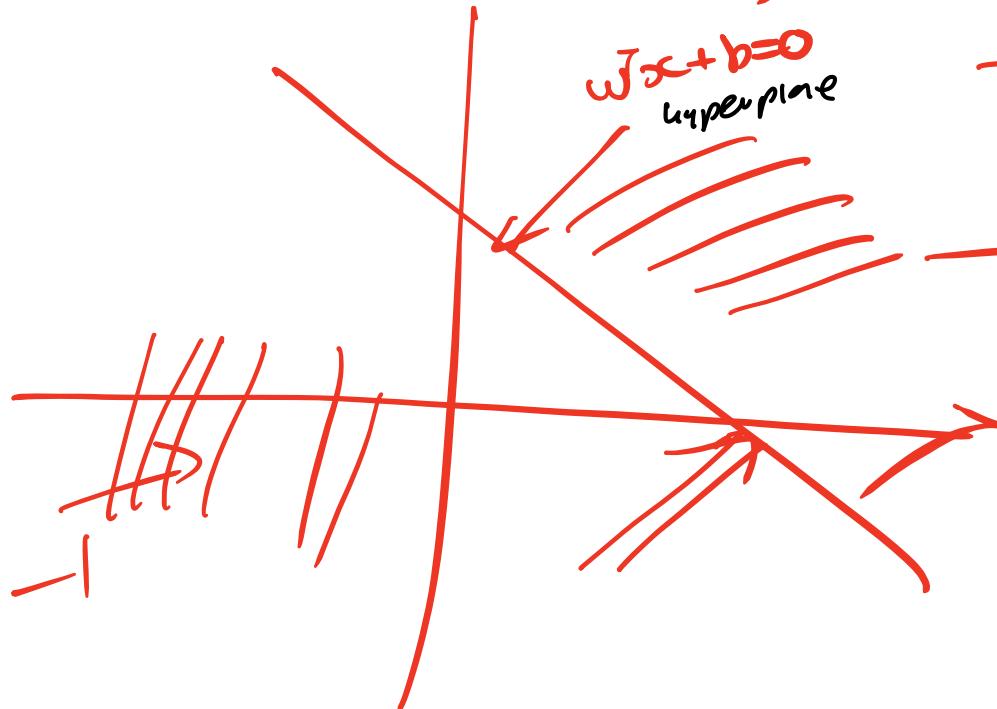
-1

+1



what values of input does this yield?

$$\sigma(w^T x + b) = 0.5$$



$$w^T x + b = 0$$

hyperplane line
derivative col

Decision boundary: Linear or nonlinear?

diff. parameters

$\sigma(a)$ is **nonlinear**, however, the decision boundary is determined by

$$\sigma(a) = 0.5 \Rightarrow a(\mathbf{x}) = 0 \Rightarrow a(\mathbf{x}) = b + \mathbf{w}^T \mathbf{x} = 0$$

which is a *linear* function in \mathbf{x}

As in the case of perceptron, b the **bias or offset or intercept term**.

\mathbf{w} the **weights** .

Logistic regression

Setup for binary classification

- Input: $\underline{x} \in \mathbb{R}^D$ D dim. feature vector
- Output: $\underline{y} \in \{0, 1\}$ binary
- Training data: $\mathcal{D} = \{(x_n, y_n), n = 1, 2, \dots, N\}$
- Hypotheses/Model:

$$h_{w,b}(x) = p(y = 1 | x; b, w) = \sigma(a(x))$$

where a follows val. given x

fin. - eq. model

sigmoid fun.

where

$$a(x) = b + \sum_d w_d x_d = b + \underline{w^T x}$$

Activation fun.

- Given training data N samples/instances:

$\mathcal{D}^{\text{TRAIN}} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, train/learn/induce $\underline{h_{w,b}}$.
Find values for (w, b) .

Example: bag of words

$$P(aDaaa \mid \text{Bag 1}) \gg P(aDaaa \mid \text{Bag 2})$$

words or seq. if bag 1

same P w/
bag 2

Which bag of words is more likely to generate : aDaaa ?



1



2 ? .

Example: bag of words

Which bag of words is more likely to generate : aDaaa ?

$$\begin{aligned} & \cancel{0.7} \times \cancel{0.1} \times \cancel{0.7} \times \cancel{0.7} \times \cancel{0.7} \\ & = 2.401 \times 10^{-2} \end{aligned}$$



$$\begin{aligned} & \cancel{0.2} \times \cancel{0.1} \times \cancel{0.2} \times \cancel{0.2} \times \cancel{0.2} \\ & = 1.6 \times 10^{-4} \end{aligned}$$



Example: drawing color cards from an envelope

2 possibilities

$Y: \theta$

$P: 1 - \theta$

- An envelope with two colors of cards: yellow and purple.
- Assume if you draw a card at random, probability of drawing a yellow card is θ and probability of drawing a purple card is $1 - \theta$.
- Sample with replacement n times.
- k times we get yellow, $n - k$ times we get purple.
- The joint probability (likelihood) : $\theta^k (1 - \theta)^{n-k}$.
- What is the value of θ that maximizes the joint probability?

$$P(k \text{ yellow} \& n-k \text{ purple}) = \theta^k (1 - \theta)^{n-k}$$

joint probability

$$\times P(\text{yellow in 1st draw}) \times P(\text{yellow in 2nd draw}) \times \dots \times P(\text{purple in } n\text{th draw})$$
$$\times P(\text{purple in } n+1\text{th draw}) \times \dots \times P(\text{purple in } m\text{th draw})$$

Example: drawing color cards from an envelope

Solve $\operatorname{argmax}_{\theta} \theta^k (1 - \theta)^{n-k}$

Equivalently, we can solve:

$$\begin{aligned} & \operatorname{argmax}_{\theta} \log(\theta^k (1 - \theta)^{n-k}) = \\ &= \operatorname{argmax}_{\theta} k \log \theta + (n - k) \log(1 - \theta) \end{aligned}$$

log of this func

Log(θ^k) + Log($(1-\theta)^{n-k}$)

multiplying log likelihood

for

$$\frac{d}{d\theta} = 0$$

At the optimum: $\frac{(k \log \theta + (n - k) \log(1 - \theta))}{d\theta} = 0$ *derivative vanishes*

Example: drawing color cards from an envelope

Maximum likelihood estimate (MLE): $\hat{\theta} = \frac{k}{n}$

These are easy examples. We don't always have a closed-form solution for the MLE typically !

$\hat{\theta}$, active value, ques!

Likelihood Function

likelihood
+ maximizing
the likelihood

statistical model

$$p(x|\theta)$$

$$p(x; \theta)$$

Let X_1, \dots, X_N be IID (independent and identically distributed) random variables with PDF $p(x|\theta)$ (also written as $p(x; \theta)$). The *likelihood function* is defined by $\underline{\underline{L(\theta)}}$,

$$\underline{\underline{L(\theta)}} = p(\underline{\underline{X_1, \dots, X_N}}; \underline{\underline{\theta}}).$$

every value of θ

$$= \prod_{i=1}^N p(X_i; \theta).$$

Notes The likelihood function is just the joint density of the data, except that we treat it as a function of the parameter θ .

Maximum Likelihood Estimator

$$L(\theta) = \prod_{i=1}^n P(x_i; \theta)$$

log o C (independent)

$$\begin{aligned} l(\theta) &= \log(L(\theta)) = \log\left(\prod_{i=1}^n P(x_i; \theta)\right) \\ &= \sum_{i=1}^n \log P(x_i; \theta) \end{aligned}$$

Definition: The maximum likelihood estimator (MLE) $\hat{\theta}$, is the value of θ that maximizes $L(\theta)$.

The log-likelihood function is defined by $l(\theta) = \log L(\theta)$. Its maximum occurs at the same place as that of the likelihood function.

- Using logs simplifies mathematical expressions (converts exponents to products and products to sums)
- Using logs helps with numerical stability

The same is true of the likelihood function times any constant. Thus we shall often drop constants in the likelihood function.

$$\underline{c L(\theta)}$$

$$L(\theta)$$

Likelihood function for logistic regression

$$P(Y=1|x; b, w) = \sigma(w^T x + b) \quad - \text{single training sample}$$

$$P(Y=0|x; b, w) \Rightarrow P(Y=0|x; b, w) \\ = 1 - \sigma(w^T x + b)$$

Probability of a single training sample (x_n, y_n)

$$p(y_n|x_n; b, w) = \begin{cases} h_{w,b}(x_n) = \sigma(b + w^T x_n) & \text{if } y_n = 1 \\ = 1 - h_{w,b}(x_n) = 1 - \sigma(b + w^T x_n) & \text{otherwise} \end{cases}$$

Compact expression, exploring that y_n is either 1 or 0

$$p(y_n|x_n; b; w) = h_{w,b}(x_n)^{y_n} [1 - h_{w,b}(x_n)]^{1-y_n}$$

$y_n = 1$

$$(h_{w,b}(x_n))^1 (1 - h_{w,b}(x_n))^0$$

\Downarrow

$$h_{w,b}(x_n) \cdot 1$$

Log Likelihood

$$\log(L(w, b)) = \log \left(\prod_{n=1}^N P(y_n | x_n; w, b) \right)$$

Log-likelihood of the whole training data \mathcal{D}

$$l(w, b) = \sum_n \{y_n \log h_{w,b}(x_n) + (1 - y_n) \log[1 - h_{w,b}(x_n)]\}$$

$$l(\underline{w}, \underline{b}) = \sum_{n=1}^N \log \underline{P(y_n | x_n; w, b)}$$

$$\begin{aligned} &= - \sum_{n=1}^N \log \left(h_{w,b}(x_n)^{y_n} (1 - h_{w,b}(x_n))^{1-y_n} \right) \\ &= \sum_{n=1}^N \log \left(h_{w,b}(x_n)^{y_n} \right) + \log \left((1 - h_{w,b}(x_n))^{1-y_n} \right) \end{aligned}$$

$$\log(a^b) = b \log(a)$$

Log Likelihood

Log-likelihood of the whole training data \mathcal{D}

$$l(\mathbf{w}, b) = \sum_n \{y_n \log h_{\mathbf{w}, b}(\mathbf{x}_n) + (1 - y_n) \log[1 - h_{\mathbf{w}, b}(\mathbf{x}_n)]\}$$

It is convenient to work with its negation termed negative log likelihood

$$\underline{\underline{J(b, \mathbf{w})}} = - \sum_n \{y_n \log h_{\mathbf{w}, b}(\mathbf{x}_n) + (1 - y_n) \log[1 - h_{\mathbf{w}, b}(\mathbf{x}_n)]\}$$

We can ignore the distinction between bias and weights

(b, w)

bases & neurons
hyperplane

as features
append 1
new set of parameters
new for

This is for convenience

- Append 1 to x

$$x \leftarrow [1 \quad x_1 \quad x_2 \quad \cdots \quad x_D]$$

\equiv $\overbrace{\hspace{10em}}$

- Append b to w

$$\theta \leftarrow [b \quad w_1 \quad w_2 \quad \cdots \quad w_D]$$

\equiv $\overbrace{\hspace{10em}}$

-

$$J(\theta) = - \sum_n \{y_n \log h_{\theta}(x_n) + (1 - y_n) \log [1 - h_{\theta}(x_n)]\}$$

\equiv

- Same trick as in the case of perceptrons

- we are rewriting a hyperplane in D dimensions as one in $D + 1$ dimensions that passes through the origin.

How to find the optimal parameters for logistic regression?

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \ell(\theta) \quad \text{minimum error value}$$

MLE

$$= \underset{\theta}{\operatorname{argmin}} -\ell(\theta)$$

We will minimize the negative log likelihood

$$J(\theta) = - \sum_n \{y_n \log h_{\theta}(x_n) + (1 - y_n) \log[1 - h_{\theta}(x_n)]\}$$

How do we find its minimum?

$$\theta = (\theta_0, \theta_1, \dots, \theta_D)$$

$$\nabla J(\theta) = \begin{pmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_D} \end{pmatrix} \Rightarrow$$

How to find the optimal parameters for logistic regression?

We will minimize the negative log likelihood

$$J(\theta) = - \sum_n \{y_n \log h_{\theta}(x_n) + (1 - y_n) \log[1 - h_{\theta}(x_n)]\}$$

How do we find its minimum?

$$\theta = (\theta_0, \theta_1, \dots, \theta_D)$$

$$\nabla J(\theta) = \left(\begin{array}{c} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_D} \end{array} \right) \Rightarrow$$

Outline

what is now likelihood

1 Logistic regression

2 Optimization

3 Stochastic gradient descent

$\hat{\theta}$ over max
a: in
each gen

$$\hat{\theta} = \arg \max_{\theta} L(\theta) + P(X_1 \dots X_n | \theta)$$

w. w.r.t:
the param.

Maximum Likelihood

$$x_1, \dots, x_n \sim P(x | \theta)$$

$$P(X_i = \text{Yellow}) = 0.2$$

$$P(X_i = \text{Purple}) = 0.8$$

$$P(X_1, \dots, X_n | \theta = 0.1)$$

$$P(X_1, \dots, X_n | \theta = 0.8)$$

Optimization

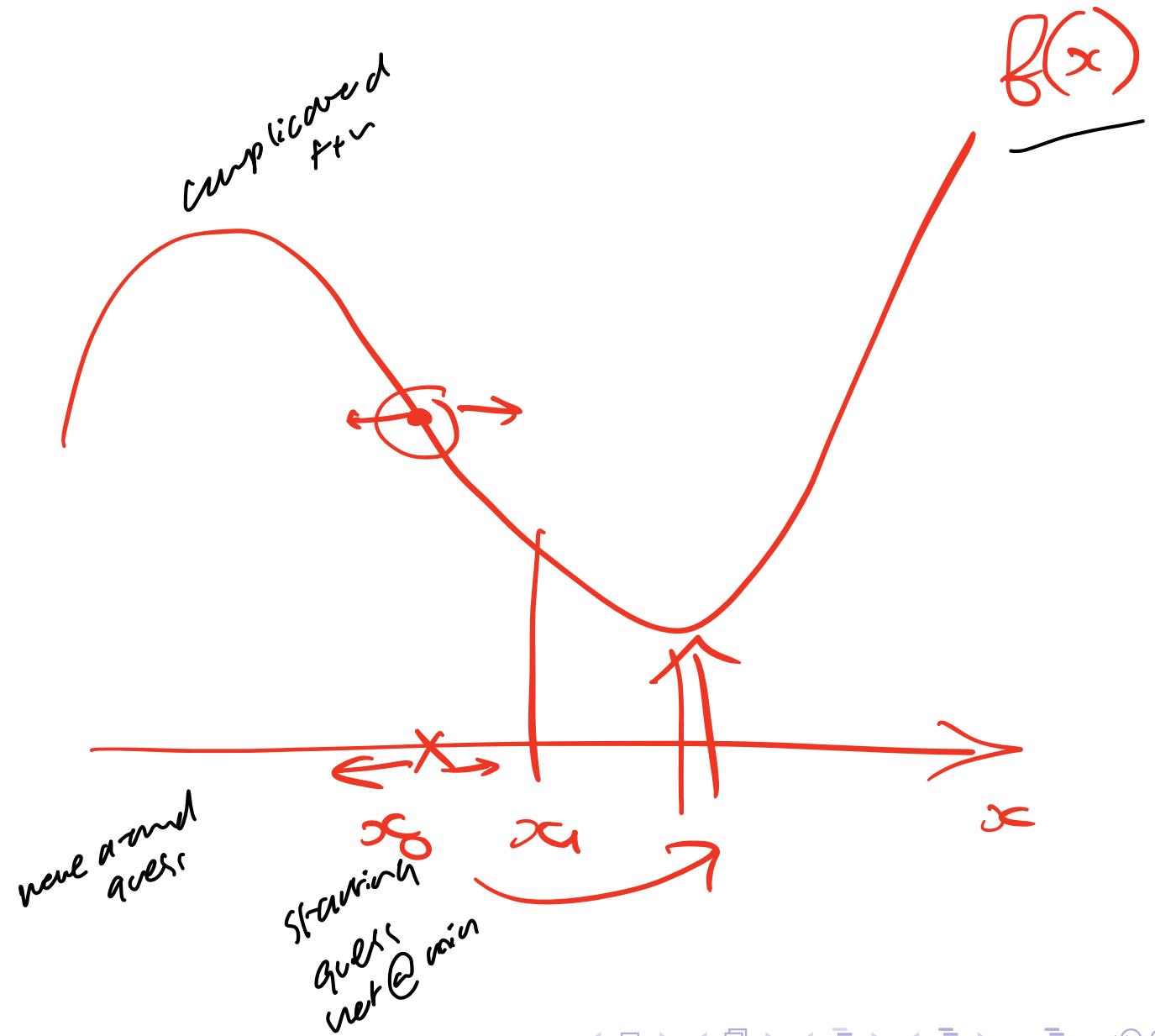
Given a function $f(x)$, find its minimum (or maximum).

- f is called the **objective function**.
- Maximizing f is equivalent to minimizing $\underline{-f}$

So we only need to consider minimization problems.

One way to minimize a function f

Gradient descent



Gradient Descent

Start at a random point

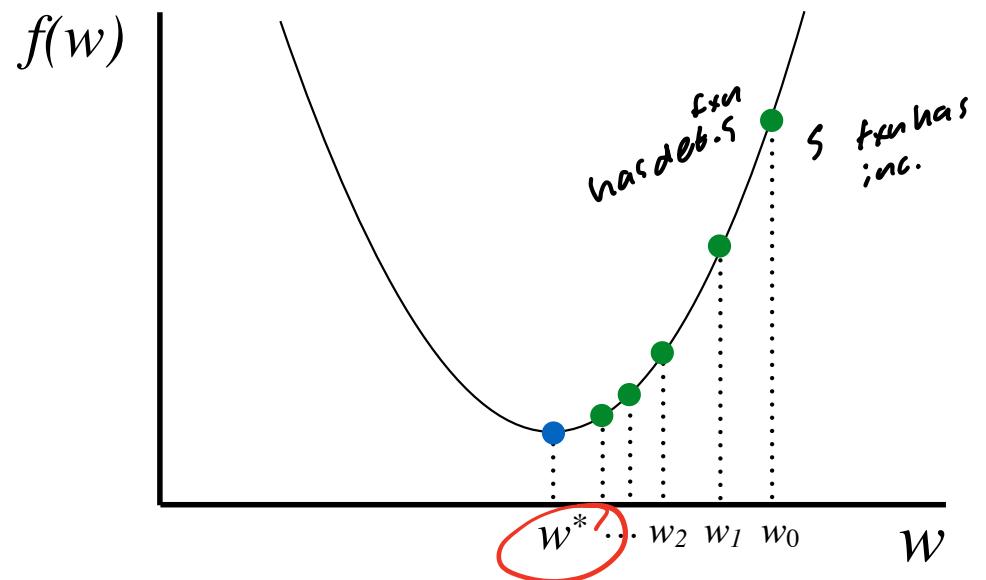
Repeat

Determine a descent direction

Choose a step size

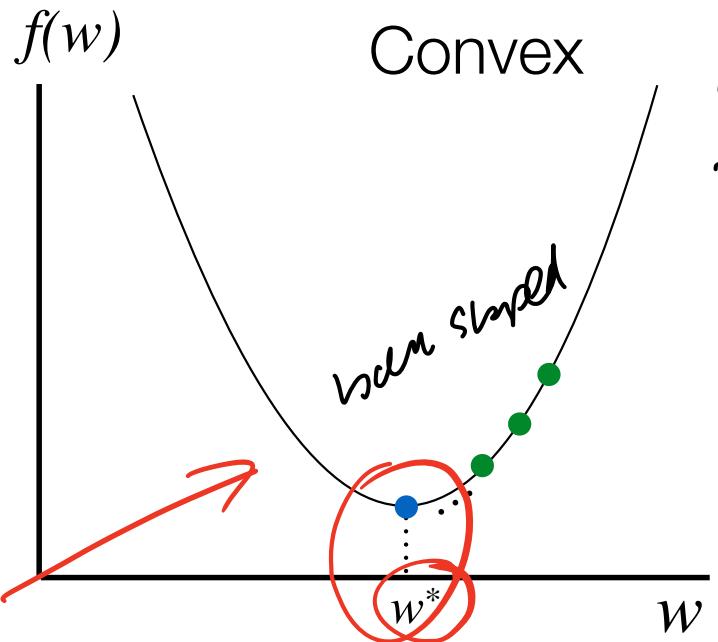
Update

Until stopping criterion is satisfied



choosing step size
is very important

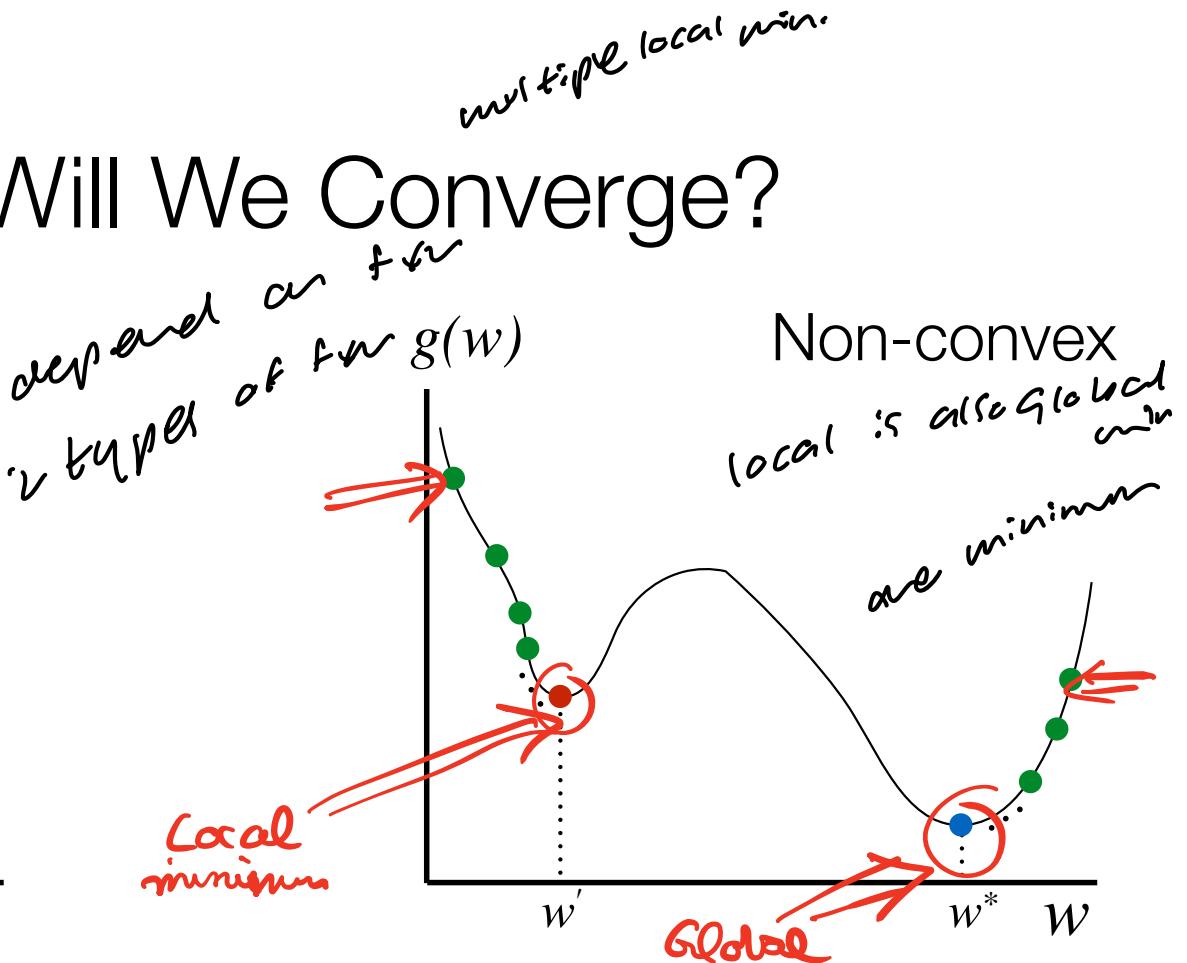
Where Will We Converge?



Convex

U-shape

w^*



Non-convex

local is also global
min

are minimum

Global

w^*

Multiple local minima may exist



**Least Squares, Ridge Regression and
Logistic Regression are all convex!**

Convex functions

A function $f(x)$ is convex if

$$f(\lambda a + (1 - \lambda)b) \leq$$

$$f(\lambda a + (1 - \lambda)b) \leq \lambda f(a) + (1 - \lambda)f(b)$$

for

$$x_\lambda = \lambda a + (1 - \lambda)b$$

$$0 \leq \lambda \leq 1$$

$$0 \leq \lambda \leq 1$$

$$\lambda = 1$$

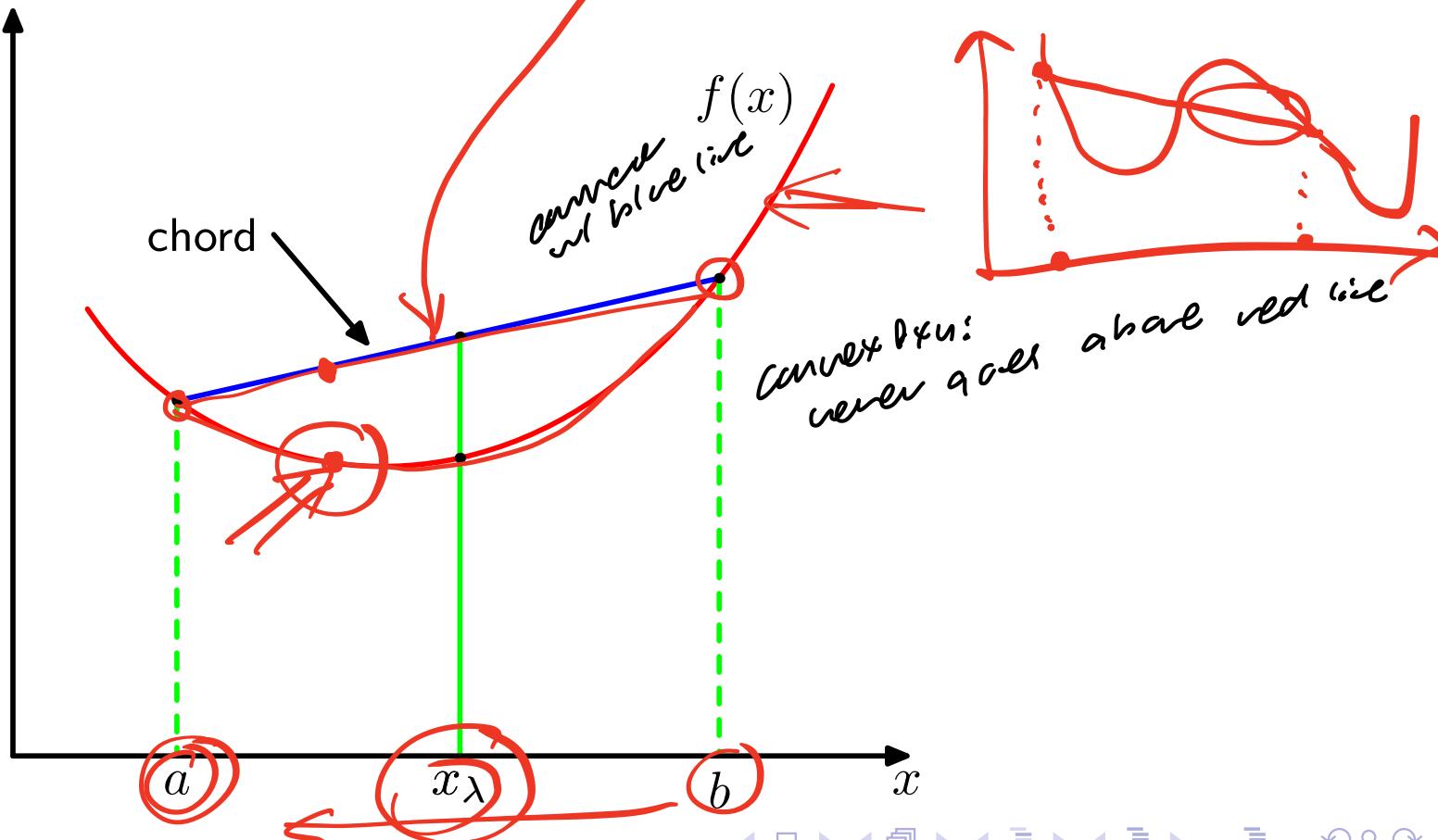
$$a = a$$

$$\lambda = 0$$

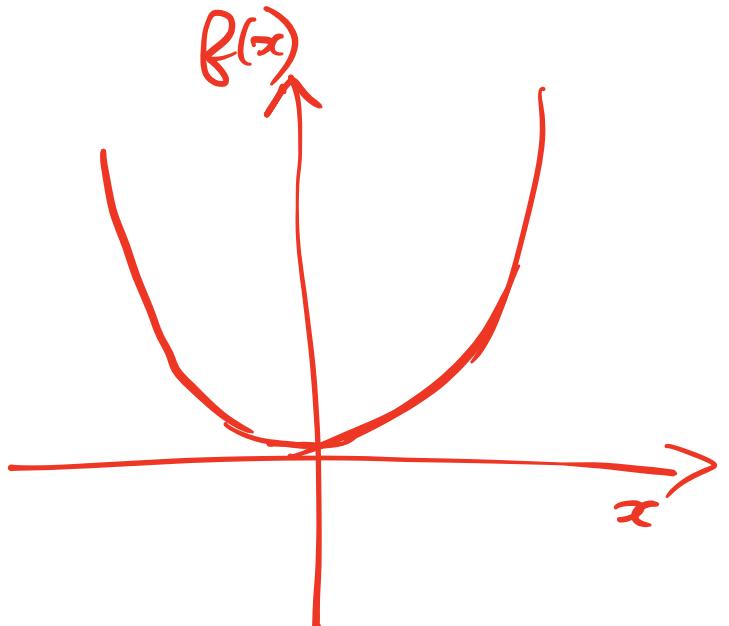
$$x = b$$

$$\lambda = \frac{1}{2}$$

$$x = \frac{1}{2}a + \frac{1}{2}b$$



How to determine convexity?



$f(x)$ is convex if

$$\underline{\underline{f''(x) \geq 0}}$$

Examples:

$$f(x) = x^2, f''(x) = 2 > 0$$

$$\underline{\underline{}}$$

$$f'(x) = 2$$

$$f'(x) = 2x$$

Examples

Convex functions

$$f''(x) = 0 \geq 0$$

$$f(x) = ax + b$$

$$f(x) = x^2$$

$$\underline{f(x) = e^x}$$

$$f''(x) = e^x \geq 0$$

$$f(x) = \frac{1}{x}, x \geq 0$$

Nonconvex functions

$$f(x) = \cos(x)$$

$$f(x) = e^x - x^2$$

$$f(x) = \log(x)$$

Multi-variate functions

Definition

$f(x)$ is convex

multi-dim.

$$f(\lambda \mathbf{a} + (1 - \lambda) \mathbf{b}) \leq \lambda f(\mathbf{a}) + (1 - \lambda) f(\mathbf{b})$$

for all \mathbf{a}, \mathbf{b} , $0 \leq \lambda \leq 1$

Multi-variate functions

$$\frac{\partial^2 f}{\partial x_i \partial x_j}$$

How to determine convexity in this case?

Matrix of second-order derivatives (Hessian)

$$H = \begin{pmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \ddots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_D} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_D} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_D} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_D^2} \end{pmatrix}$$

(PXP)

coord. direction derivatives

prod. of $m \times m$ second deriv.

Marwd
dev.v. kn
even
par

$$f''(\mathbf{x}) = \frac{d^2 Q(\mathbf{x})}{d\mathbf{x}^2} \geq 0$$

Multi-variate functions

How to determine convexity in this case?

If the Hessian is positive semi-definite $\underline{\underline{H \geq 0}}$ then f is convex.

A matrix H is positive semi-definite if and only if

$$\underbrace{z^T H z}_{\sum_{j,k} H_{j,k} z_j z_k} \geq 0$$

for all z .

H
 $D \times D$

z
 D

$$z^T H z \geq 0$$

scal. w/ unit vector dir.
convex
pos. definite

$\underbrace{z^T}_{1 \times D} \underbrace{H}_{D \times D} \underbrace{z}_{D \times 1}$

Multi-variate functions

$$\underline{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Example

$$f(\underline{x}) = x_1^2 + 2x_2^2$$

$$H = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}$$

derivative
evaluation pos.

symmetric

$$\underline{z}^T H \underline{z} = \underline{\underline{2z_1^2}} + \underline{\underline{4z_2^2}} \geq 0$$

Multi-variate functions

Example

$$f(\mathbf{x}) = x_1^2 + 2x_2^2$$

$$\mathbf{H} = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}$$

$$\mathbf{z}^T \mathbf{H} \mathbf{z} = 2z_1^2 + 4z_2^2 \geq 0$$

Example: $\min f(\theta) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2$

- We compute the gradients

$$\left\{ \begin{array}{l} \frac{\partial f}{\partial \theta_1} = 2(\theta_1^2 - \theta_2)\theta_1 + \theta_1 - 1 \\ \frac{\partial f}{\partial \theta_2} = -(\theta_1^2 - \theta_2) \end{array} \right. \quad (1)$$

$$= -(\theta_1^2 - \theta_2) \quad (2)$$

- Use the following *iterative* procedure for *gradient descent*

① Initialize $\theta_1^{(0)}$ and $\theta_2^{(0)}$, and $t = 0$

② do

the gradient

$$\underline{\theta_1^{(t+1)}} \leftarrow \underline{\theta_1^{(t)}} - \eta \underline{[2(\theta_1^{(t)} - \theta_2^{(t)})\theta_1^{(t)} + \theta_1^{(t)} - 1]} \quad (3)$$

$$\underline{\theta_2^{(t+1)}} \leftarrow \underline{\theta_2^{(t)}} - \eta \underline{[-(\theta_1^{(t)} - \theta_2^{(t)})]} \quad (4)$$

$$t \leftarrow t + 1 \quad \begin{matrix} \text{repeat} \\ \text{under your step} \end{matrix} \quad (5)$$

③ until $f(\theta^{(t)})$ does not change much

Gradient descent

General form for minimizing $f(\theta)$

$$\theta^{t+1} \leftarrow \theta^t - \eta \nabla f(\theta^t)$$

compute gradient

general form

Remarks

- η is often called *step size* – literally, how far our update will go along the direction of the negative gradient
- Note that this is for *minimizing* a function, hence the subtraction ($-\eta$)
- With a *suitable* choice of η , the iterative procedure converges to a stationary point where

$$\nabla f(\theta) = 0$$

converges

where gradient = 0

- A stationary point is only necessary for being the minimum.



apply to original model

Summary

Setup for binary classification

- Logistic Regression models conditional distribution as:
 $p(y = 1|\mathbf{x}; \boldsymbol{\theta}) = \sigma[a(\mathbf{x})]$ where $a(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$
- Linear decision boundary: $a(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} = 0$

Minimizing the negative log-likelihood

- $J(\boldsymbol{\theta}) = -\sum_n \{y_n \log \sigma(\boldsymbol{\theta}^T \mathbf{x}_n) + (1 - y_n) \log[1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}_n)]\}$
- No closed form solution; must rely on iterative solvers

Numerical optimization

- Gradient descent: simple, scalable to large-scale problems
 - ▶ move in direction opposite of gradient!
 - ▶ gradient of logistic function takes nice form

Logistic regression (continued); Linear regression

how do we learn these params

write down likelihood

fun of parameters
pick param. that maximizes

Sriram Sankararaman

The instructor gratefully acknowledges Fei Sha, Ameet Talwalkar, Eric Eaton, Kai-Wei Chang, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

maximum of likelihood

optimization

→ alg. that optimizes

Outline

1 Logistic regression

- Stochastic gradient descent

2 Linear regression

3 Multivariate solution

Gradient descent

$$\begin{aligned} \text{obj. fun} \\ \theta^t &= \left(\begin{array}{c} \theta_0^t \\ \vdots \\ \theta_n^t \end{array} \right) \\ \nabla f &= \left(\begin{array}{c} \frac{\partial f}{\partial \theta_0} \\ \vdots \\ \frac{\partial f}{\partial \theta_n} \end{array} \right) \end{aligned}$$

General form for minimizing $f(\theta)$

find θ that minimizes fun
partial deriv. w.r.t. many input variables
minimizes value

$$\theta^{t+1} \leftarrow \theta^t - \eta \nabla f(\theta^t)$$

decreases
→ minimum

compute gradient of fun
evaluate gradient @ data
changes w.r.t.

Remarks

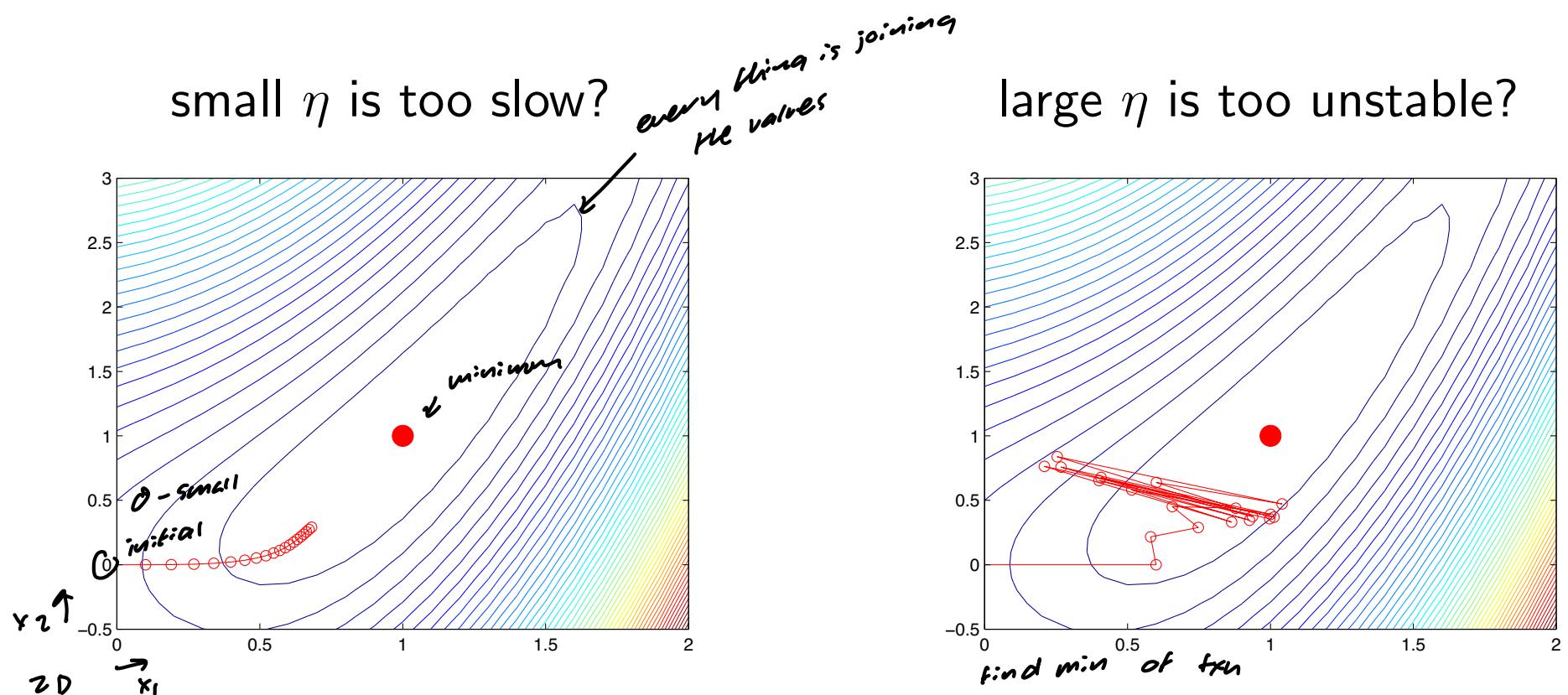
- η is often called *step size* – literally, how far our update will go along the direction of the negative gradient
- Note that this is for *minimizing* a function, hence the subtraction ($-\eta$)
- With a *suitable* choice of η , the iterative procedure converges to a stationary point where

$$\nabla f(\theta) = 0$$

- A stationary point is only necessary for being the minimum.

Seeing in action

Choosing the right η is important



Gradient descent

$$\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

algorithm

Algorithm 1 Gradient descent

- 1: $\theta \leftarrow 0$. starting val.
- 2: **for** $epoch = 1 \dots T$ **do** iterations
- 3: $\theta \leftarrow \theta - \eta \nabla J(\theta)$ *operations at moving individual*
- 4: **end for**
- 5: **return** θ

Gradient Descent Update for Logistic Regression

Derivatives of $\sigma(a)$

$$\begin{aligned}\frac{d\sigma(a)}{da} &= \frac{d}{da} (1 + e^{-a})^{-1} = \frac{-(1 + e^{-a})'}{(1 + e^{-a})^2} \\ &= \frac{e^{-a}}{(1 + e^{-a})^2} = \frac{1}{1 + e^{-a}} \frac{e^{-a}}{1 + e^{-a}} \\ &= \sigma(a)[1 - \sigma(a)]\end{aligned}$$

Gradients of the negative log likelihood

Negative log likelihood

$$\mathcal{J}(\theta) = -\sum_{n=1}^N \left[y_n \log h_\theta(x_n) + (1-y_n) \log [1 - h_\theta(x_n)] \right]$$

$\nabla \mathcal{J}(\theta)$ $h_\theta(x_n) = P(y_n=1 | x_n=\theta) = \sigma(\theta^T x_n)$

$$J(\theta) = - \sum_n \{y_n \log h_\theta(x_n) + (1 - y_n) \log[1 - h_\theta(x_n)]\}$$

Gradients

$$\frac{\partial J(\theta)}{\partial \theta} = - \sum_n \{y_n[1 - \sigma(\theta^T x_n)]x_n - (1 - y_n)\sigma(\theta^T x_n)]x_n\} \quad (1)$$

$$= \sum_n \{\sigma(\theta^T x_n) - y_n\} x_n \quad (2)$$

$$= \sum_n \{h_\theta(x_n) - y_n\} x_n \quad (3)$$

Remark

Gradients of the negative log likelihood

Negative log likelihood

$$J(\theta) = - \sum_n \{y_n \log h_{\theta}(x_n) + (1 - y_n) \log[1 - h_{\theta}(x_n)]\}$$

$$\stackrel{\text{def}}{=} - \sum_n \{y_n \log(\sigma(\theta^T x_n)) + (1 - y_n) \log(1 - \sigma(\theta^T x_n))\}$$

Gradients

$$\left[\frac{\partial J(\theta)}{\partial \theta} \right] = - \sum_n \{y_n [1 - \sigma(\theta^T x_n)] x_n - (1 - y_n) \sigma(\theta^T x_n) x_n\} \quad (1)$$

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

$$= \sum_n \{\sigma(\theta^T x_n) - y_n\} x_n$$

$$\frac{\partial \sigma(a)}{\partial a} = \frac{d\sigma}{dw} \times \frac{dw}{da} \quad (2)$$

$$\frac{d}{da} \sigma(a) = \frac{1}{(1 + e^{-a})^2}$$

$$= \sum_n \{h_{\theta}(x_n) - y_n\} x_n$$

$$= \frac{d\sigma}{dw} = \frac{1}{(1 + e^{-w})^2} \quad \frac{dw}{da} = -e^{-a} \quad (3)$$

Remark

$$\frac{\partial \sigma(w)}{\partial w} = \frac{1}{(1 + e^{-w})^2} \quad \frac{dw}{da} = e^{-a}$$

$$\begin{aligned} &= \frac{1}{(1 + e^{-w})^2} \times (1 + e^{-a}) \\ &= \frac{e^{-a}}{(1 + e^{-a})^2} = \frac{e^{-a}}{(1 + e^{-a})^2} = \sigma(a)/(1 - \sigma(a)) \end{aligned}$$

- $e_n = \{h_{\theta}(x_n) - y_n\}$ is called **error** for the n th training sample.

Numerical optimization

Gradient descent

- Choose a proper step size $\eta > 0$
- Iteratively update the parameters following the negative gradient to minimize the error function *can be adapted*

$$\underbrace{\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \sum_n \{\sigma(\theta^T x_n) - y_n\} x_n}_{\text{gradient}}$$

*step size
can change* *each comp. - save*
*problem
is large* *iteration
is too big*

Remarks

- The step size needs to be chosen carefully to ensure convergence.
- The step size can be adaptive (i.e. varying from iteration to iteration). For example, we can use techniques such as *line search*
- There is a variant called *stochastic* gradient descent, also popularly used.

Stochastic gradient descent

$$\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

$$J(\theta) = -\sum_n \{y_n \log h_{\theta}(x_n) + (1 - y_n) \log[1 - h_{\theta}(x_n)]\}$$

write of form as
summation over n

avg.

objective
fun

$$J(\theta) = \left(\frac{1}{N} \sum_n J_n(\theta) \right)$$

applying to each n data

$$\nabla J(\theta) = \frac{1}{N} \sum_n \nabla J_n(\theta) = \mathbb{E}_{n \sim \mathcal{D}} \nabla J_n(\theta)$$

expectation choosing data set
uniformly randomly
gradient gradient

Approximate the gradient by the gradient computed on a single example at a time.

- Repeat until convergence
 - ▶ Randomly pick one example (x_n, y_n) .
 - ▶ Update $\theta \leftarrow \theta - \eta \nabla J_n(\theta)$.

Stochastic Gradient descent (SGD)

$$\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

Algorithm 2 Stochastic Gradient descent

```
1:  $\theta \leftarrow 0.$ 
2: for  $epoch = 1 \dots T$  do
3:   for  $(x, y) \in \mathcal{D}$  do
4:      $\theta \leftarrow \theta - \eta \nabla J_{(x,y)}(\theta)$  update decrease
5:   end for data per random six first training example
6: end for
7: return  $\theta$ 
```

$J(\theta) = \frac{1}{n} \sum_n l_{\theta}(x_n)$
 $= \frac{1}{n} \sum_i l_{\theta}(x_i) + \lambda \theta^T \theta$

(x_1, y_1)
 (x_2, y_2)
 (x_3, y_3)
 (x_4, y_4)
 (x_5, y_5)
 (x_6, y_6)

Compare to perceptron learning

$$\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

*compare
y vs. linear*

Algorithm 3 PerceptronTrain

```
1:  $\theta \leftarrow 0$ 
2: for  $iter = 1 \dots MaxIter$  do
3:   for  $(x, y) \in \mathcal{D}$  do
4:      $a \leftarrow \theta^T x$ 
5:     if  $ay \leq 0$  then           mistake:
6:        $\theta \leftarrow \theta + yx$       update parameter
7:     end if
8:   end for
9: end for
10: return  $\theta$ 
```

What is the objective function?

Compare to perceptron learning

$$\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

Algorithm 4 PerceptronTrain

```
1:  $\theta \leftarrow 0$ 
2: for  $iter = 1 \dots MaxIter$  do
3:   for  $(x, y) \in \mathcal{D}$  do
4:      $a \leftarrow \theta^T x$ 
5:     if  $ay \leq 0$  then
6:        $\theta \leftarrow \theta + yx$ 
7:     end if
8:   end for
9: end for
10: return  $\theta$ 
```

*1st order
→ converges quickly*

*minimizing objective func.
→ leads to opt. weights*

it is an obs. func

Perceptron effectively minimizes:

$$\Rightarrow J(\theta) = \sum_n \max(0, 1 - y_n \theta^T x_n)$$

Summary

Setup for binary classification

- Logistic Regression models conditional distribution as:
 $p(y = 1|\mathbf{x}; \boldsymbol{\theta}) = \sigma[a(\mathbf{x})]$ where $a(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$
- Linear decision boundary: $a(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} = 0$

Minimizing the negative log-likelihood

- $J(\boldsymbol{\theta}) = -\sum_n \{y_n \log \sigma(\boldsymbol{\theta}^T \mathbf{x}_n) + (1 - y_n) \log[1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}_n)]\}$
- No closed form solution; must rely on iterative solvers

Numerical optimization

- Gradient descent: simple, scalable to large-scale problems
 - ▶ move in direction opposite of gradient!
 - ▶ gradient of logistic function takes nice form

Outline

1 Logistic regression

2 Linear regression

- Motivation
- Algorithm
- Learning linear regression
- Univariate solution
- Probabilistic interpretation

3 Multivariate solution

Regression

Predicting a continuous outcome variable

- Predicting shoe size from height, weight and gender
- Predicting a company's future stock price using its profit and other financial info
- Predicting annual rainfall based on local flora / fauna
- Predicting song year from audio features

common learning solv

Regression

Predicting a continuous outcome variable

- Predicting shoe size from height, weight and gender
- Predicting a company's future stock price using its profit and other financial info
- Predicting annual rainfall based on local flora / fauna
- Predicting song year from audio features

Key difference from classification

Regression

Predicting a continuous outcome variable

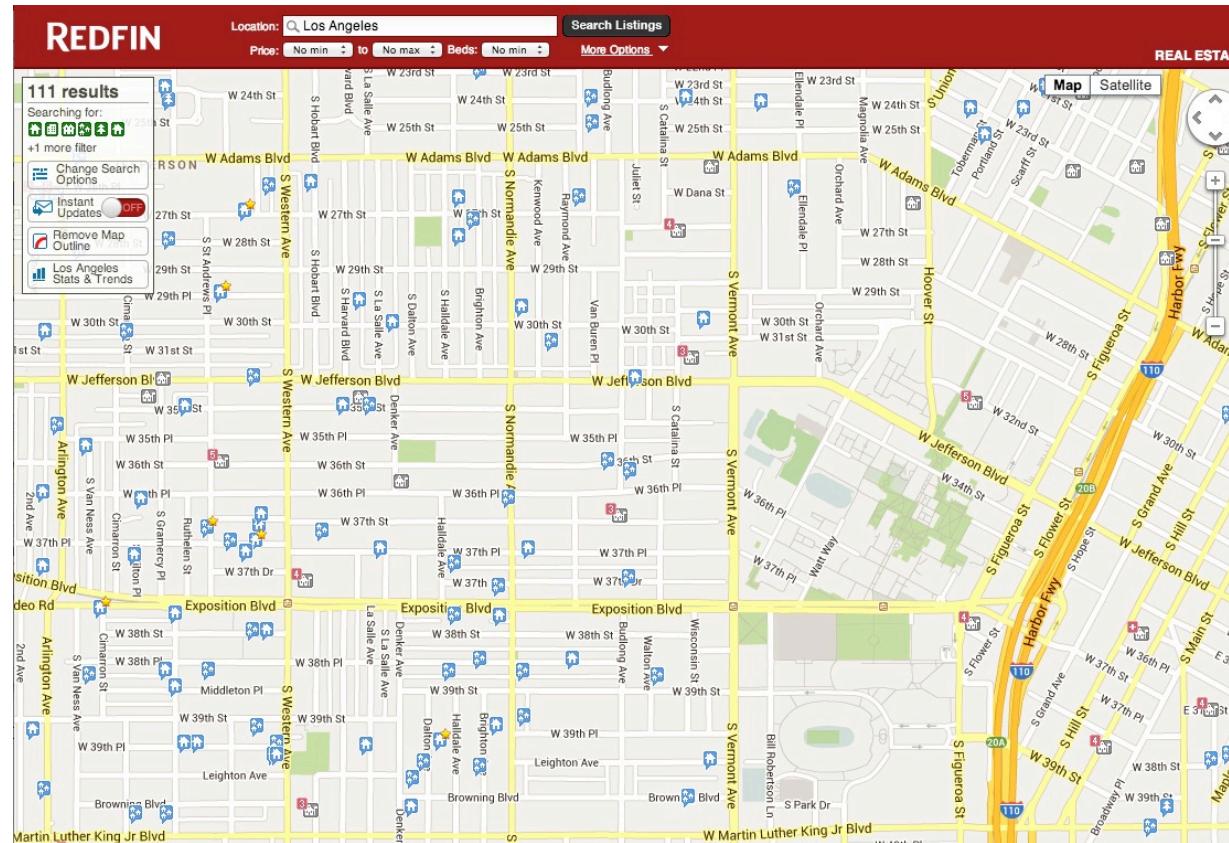
- Predicting shoe size from height, weight and gender
- Predicting a company's future stock price using its profit and other financial info
- Predicting annual rainfall based on local flora / fauna
- Predicting song year from audio features

Key difference from classification

- We can measure 'closeness' of prediction and labels, leading to different ways to evaluate prediction errors.
 - ▶ Predicting shoe size: better to be off by one size than by 5 sizes
 - ▶ Predicting song year: better to be off by one year than by 20 years
- This will lead to different learning models and algorithms

Example: predicting the sale price of a house

Retrieve historical sales records
(This will be our training data)



Features used to predict

3620 South BUDLONG
Los Angeles, CA 90007
Status: Closed

\$1,510,000 | **14** Beds | **6** Baths | **4,418 Sq. Ft.**
Last Sold Price | \$342 / Sq. Ft.
Built: 1956 | Lot Size: 9,649 Sq. Ft. | Sold On: Jul 26, 2013

Overview Property Details Tour Insights Property History Public Records Activity Schools



1 of 12 

Five unit apartment complex within 2 blocks of USC campus, Gate #6. Great for students (most student leases have parents as guarantors). Most USC students live off campus, so housing units like this are always fully leased. Situated on a gated, corner lot, and across from an elementary school, this complex was recently renovated, and has in-unit laundry hook ups, wall-unit AC, and 12 parking spaces. It is within a DPS (Department of Public Safety) and Campus Cruiser patrolled area. This is a great income generating property, not to be missed!

Property Type Multi-Family | Style Two Level, Low Rise
Community Downtown Los Angeles | County [Los Angeles](#)
MLS# 22176741

Property Details for 3620 South BUDLONG, Los Angeles, CA 90007

Details provided by i-Tech MLS and may not match the public record. [Learn More](#).

Interior Features

Kitchen Information

- Remodeled
- Oven, Range

Laundry Information

- Inside Laundry

Heating & Cooling

- Wall Cooling Unit(s)

Multi-Unit Information

Community Features

- Units in Complex (Total): 5

Multi-Family Information

- # Leased: 5
- # of Buildings: 1
- Owner Pays Water
- Tenant Pays Electricity, Tenant Pays Gas

Unit 1 Information

- # of Beds: 2
- # of Baths: 1
- Unfurnished
- Monthly Rent: \$1,700

Unit 2 Information

- # of Beds: 3
- # of Baths: 1
- Unfurnished
- Monthly Rent: \$2,250

Unit 3 Information

- Unfurnished

Unit 4 Information

- # of Beds: 3
- # of Baths: 1
- Unfurnished

Unit 5 Information

- # of Beds: 3
- # of Baths: 2
- Unfurnished
- Monthly Rent: \$2,350

Unit 6 Information

- # of Beds: 3
- # of Baths: 1
- Monthly Rent: \$2,325

Property / Lot Details

Property Features

- Automatic Gate, Card/Code Access

- Tax Parcel Number: 5040017019

Lot Information

- Lot Size (Sq. Ft.): 9,649
- Lot Size (Acres): 0.2215
- Lot Size Source: Public Records

Property Information

- Updated/Remodeled
- Square Footage Source: Public Records

Parking / Garage, Exterior Features, Utilities & Financing

Parking Information

- # of Parking Spaces (Total): 12
- Parking Space
- Gated

Utility Information

- Green Certification Rating: 0.00
- Green Location: Transportation, Walkability
- Green Walk Score: 0
- Green Year Certified: 0

Financial Information

- Capitalization Rate (%): 6.25
- Actual Annual Gross Rent: \$128,331
- Gross Rent Multiplier: 11.29

Location Details, Misc. Information & Listing Information

Location Information

- Cross Streets: W 36th Pl

Expense Information

- Operating: \$37,664

Listing Information

- Listing Terms: Cash, Cash To Existing Loan
- Buyer Financing: Cash

How to learn the unknown parameters?

training data (past sales record)

what is outcome

sqft	sale price
2000	800K
2100	907K
1100	312K
5500	2,600K
...	...

Our model

Sale price = $\text{price_per_sqft} \times \text{square_footage} + \text{fixed_expense} + \text{unexplainable_stuff}$

Reduce prediction error

How to measure errors?

- The classification error (*hit or miss*) is not appropriate for continuous outcomes.
- How should we evaluate quality of a prediction?

penalized the same if ya are/under predict
 ϵ (true prediction)²

Reduce prediction error

How to measure errors?

- The classification error (*hit or miss*) is not appropriate for continuous outcomes.
- How should we evaluate quality of a prediction?
 - ▶ *absolute* difference: $| \text{prediction} - \text{sale price}|$
 - ▶ *squared* difference: $(\text{prediction} - \text{sale price})^2$

sqft	sale price	prediction	error	squared error
2000	810K	720K	90K	90^2
2100	907K	800K	107K	107^2
1100	312K	350K	-38K	38^2
5500	2,600K	2,600K	0	0
...	...			

Minimize squared errors

Our model

Sale price = price_per_sqft \times square_footage + fixed_expense + unexplainable_stuff

Training data

sqft	sale price	prediction	error	squared error
2000	810K	720K	90K	90^2
2100	907K	800K	107K	107^2
1100	312K	350K	38K	38^2
5500	2,600K	2,600K	0	0
...	...			
Total				$90^2 + 107^2 + 38^2 + 0 + \dots$

here the model prediction

Aim

Adjust model such that the sum of the squared error is minimized — i.e., the residual/remaining unexplainable_stuff is minimized.

Linear regression (ordinary least squares)

leads to lin. reg.

Setup

- Input: $x \in \mathbb{R}^D$ (covariates, predictors, features, etc)
- Output: $y \in \mathbb{R}$ (responses, targets, outcomes, outputs, etc)

Linear regression (ordinary least squares)

Setup

- Input: $\mathbf{x} \in \mathbb{R}^D$ (covariates, predictors, features, etc)
- Output: $y \in \mathbb{R}$ (responses, targets, outcomes, outputs, etc)
- Hypotheses/Model: $h_{\mathbf{w}, b}$, with $h_{\mathbf{w}, b}(\mathbf{x}) = b + \sum_d w_d x_d = b + \mathbf{w}^T \mathbf{x}$
 $\mathbf{w} = [w_1 \ w_2 \ \cdots \ w_D]^T$: *weights*

b is called the **bias or offset or intercept term**.

$$\boldsymbol{\theta} = [b \ w_1 \ w_2 \ \cdots \ w_D]^T$$

Linear regression (ordinary least squares)

Setup

- Input: $\mathbf{x} \in \mathbb{R}^D$ (covariates, predictors, features, etc)
- Output: $y \in \mathbb{R}$ (responses, targets, outcomes, outputs, etc)
- Hypotheses/Model: $h_{\mathbf{w}, b}$, with $h_{\mathbf{w}, b}(\mathbf{x}) = b + \sum_d w_d x_d = b + \mathbf{w}^T \mathbf{x}$
 $\mathbf{w} = [w_1 \ w_2 \ \cdots \ w_D]^T$: *weights*
 b is called the **bias or offset or intercept term**.
 $\boldsymbol{\theta} = [b \ w_1 \ w_2 \ \cdots \ w_D]^T$
- Training data: $\mathcal{D} = \{(\mathbf{x}_n, y_n), n = 1, 2, \dots, N\}$

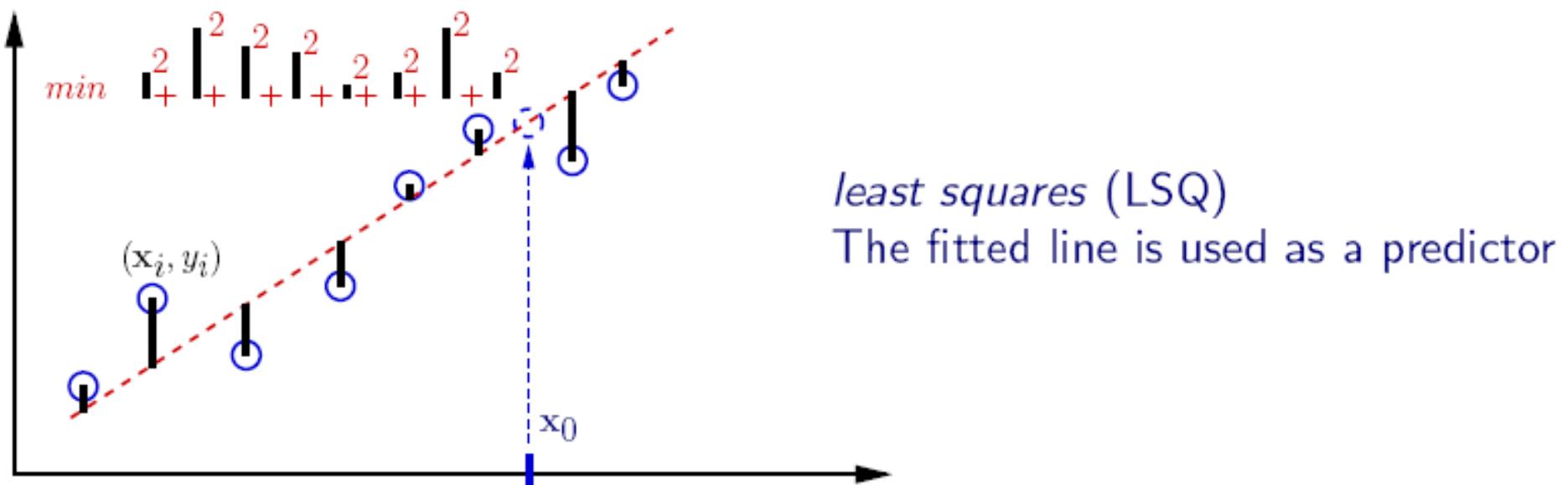
How do we learn parameters?

Minimize prediction error on training data

- Hypothesis:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

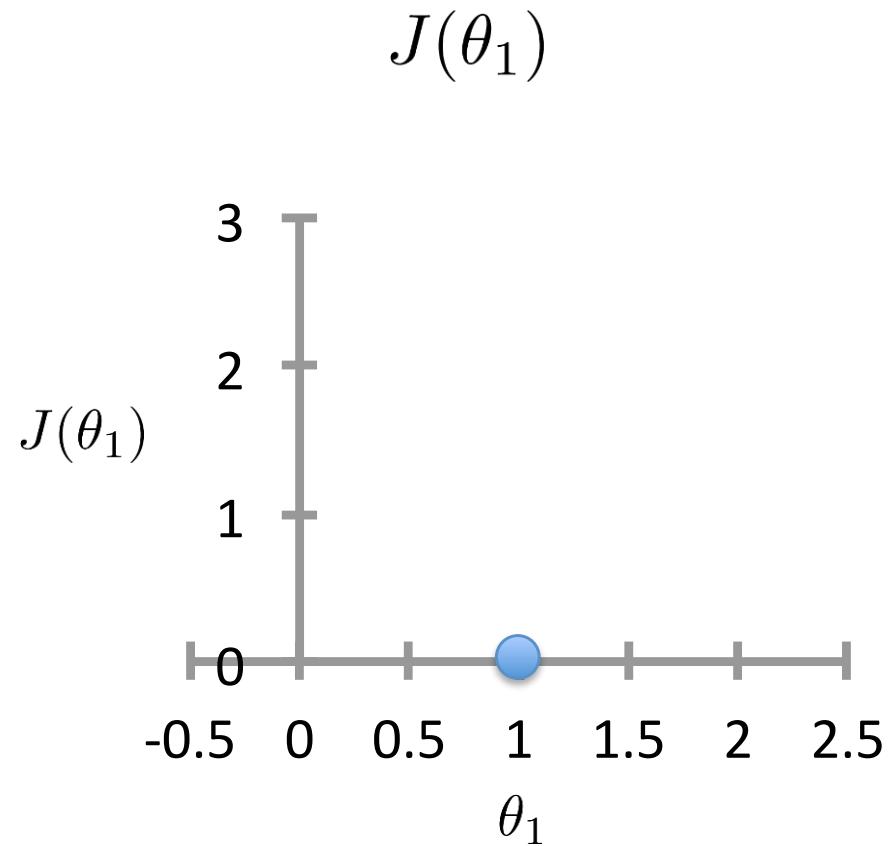
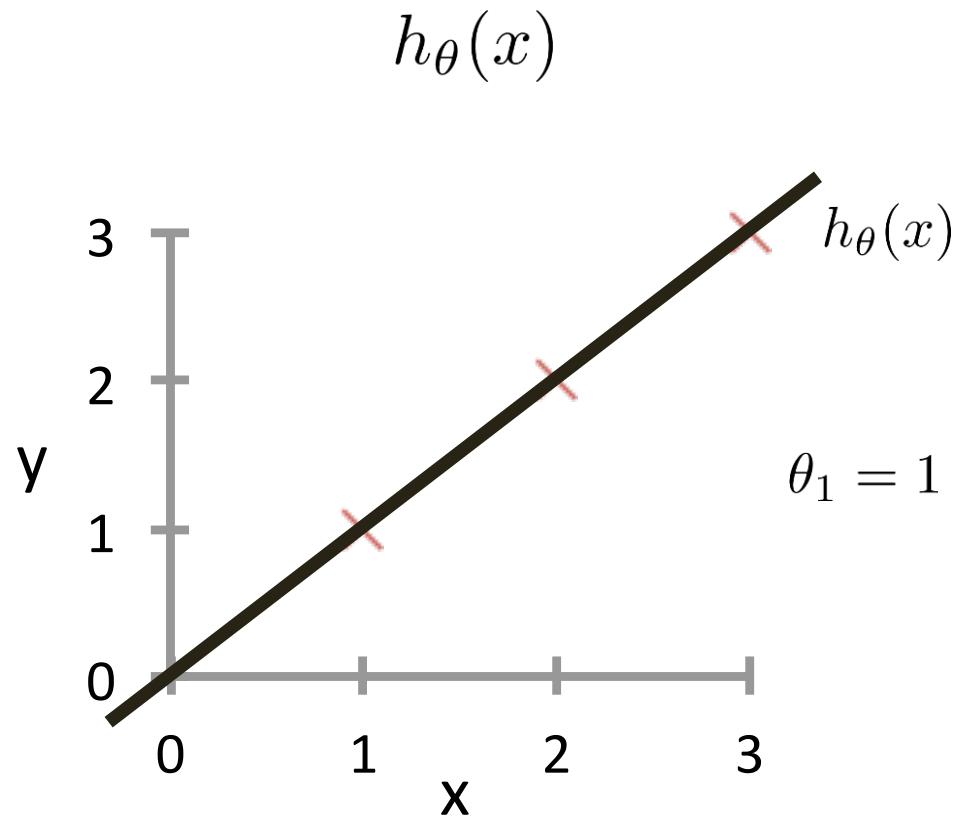
- Minimize the sum of squared errors (also called residual sum of squares RSS): **cost function** for linear regression.
- **Cost function** for logistic regression is the negative log likelihood.



Intuition behind cost function (residual sum of squares RSS)

Assume $x \in \mathbb{R}$, $\theta_0 = 0$.

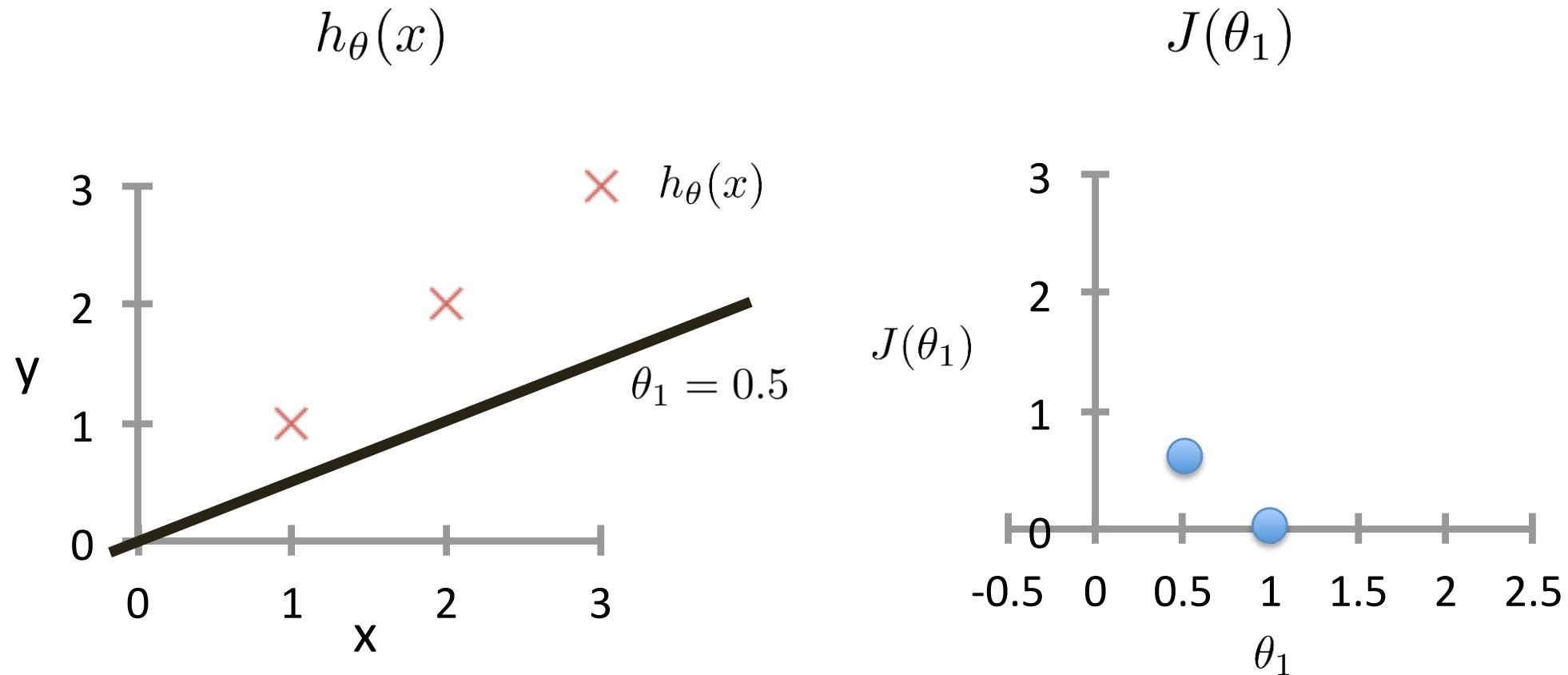
$$h_{\theta}(x) = \theta_0 + \theta_1 x = \theta_1 x$$



Intuition behind cost function (residual sum of squares RSS)

Assume $x \in \mathbb{R}$, $\theta_0 = 0$.

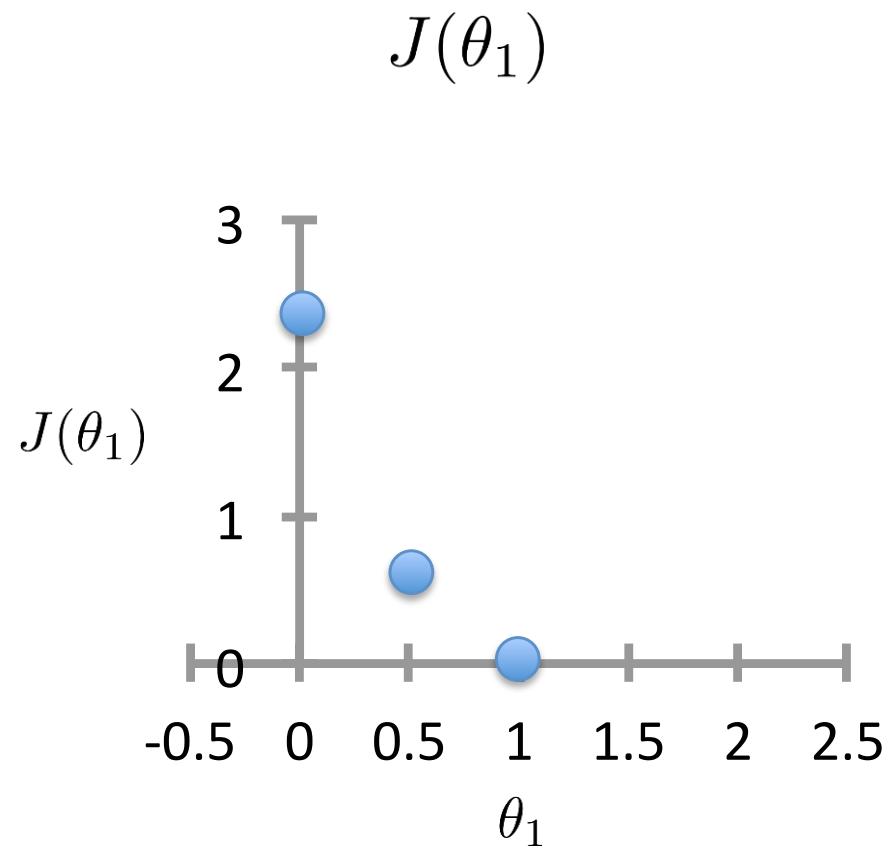
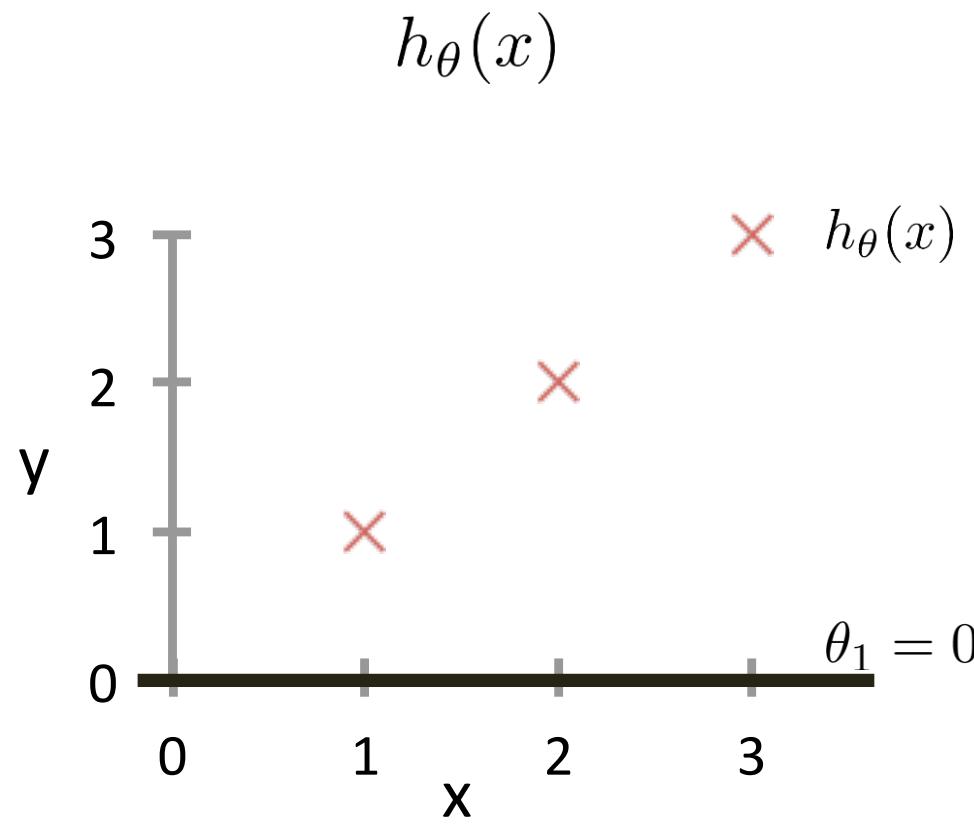
$$h_{\theta}(x) = \theta_0 + \theta_1 x = \theta_1 x$$



Intuition behind cost function (residual sum of squares RSS)

Assume $x \in \mathbb{R}$, $\theta_0 = 0$.

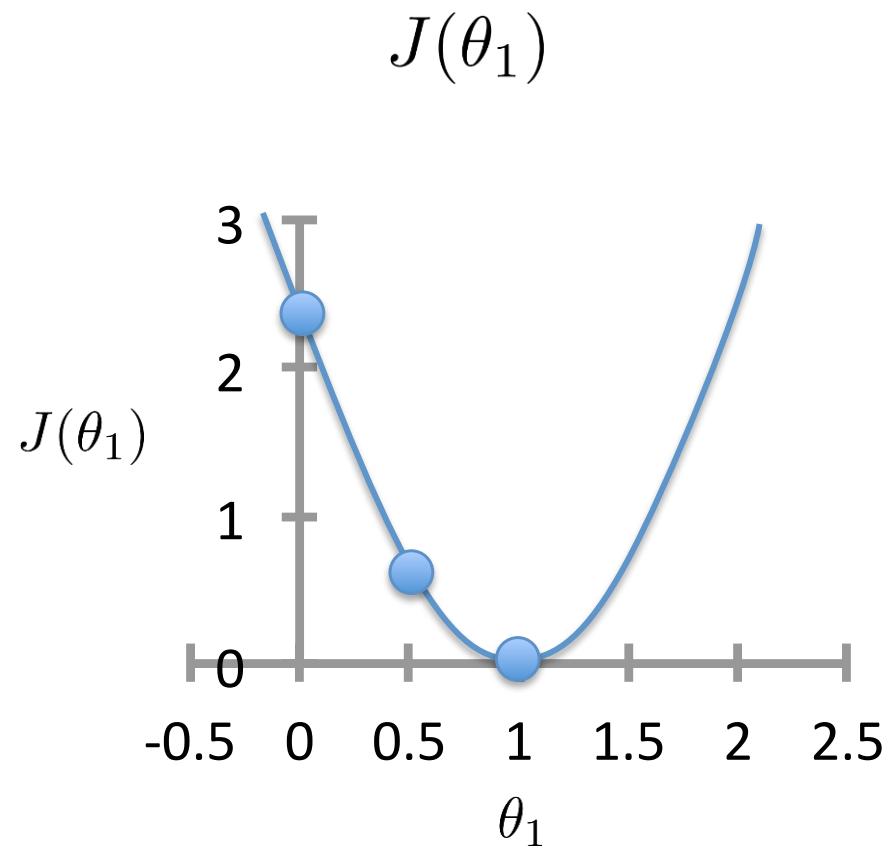
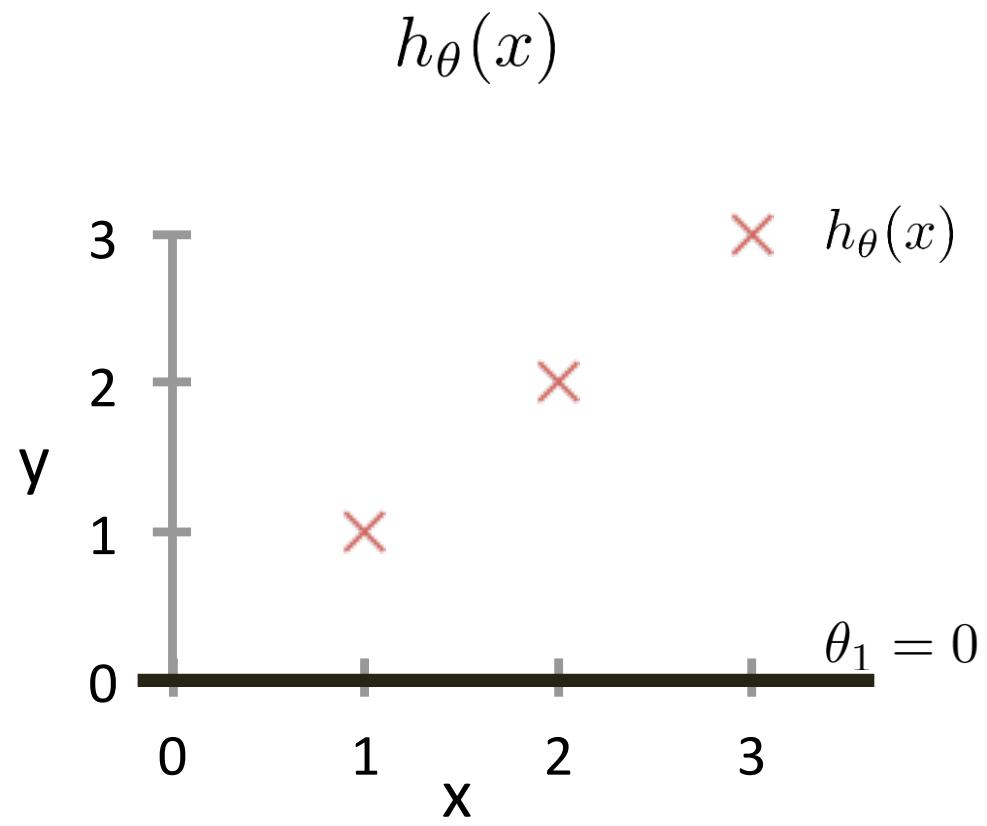
$$h_{\theta}(x) = \theta_0 + \theta_1 x = \theta_1 x$$



Intuition behind cost function (residual sum of squares RSS)

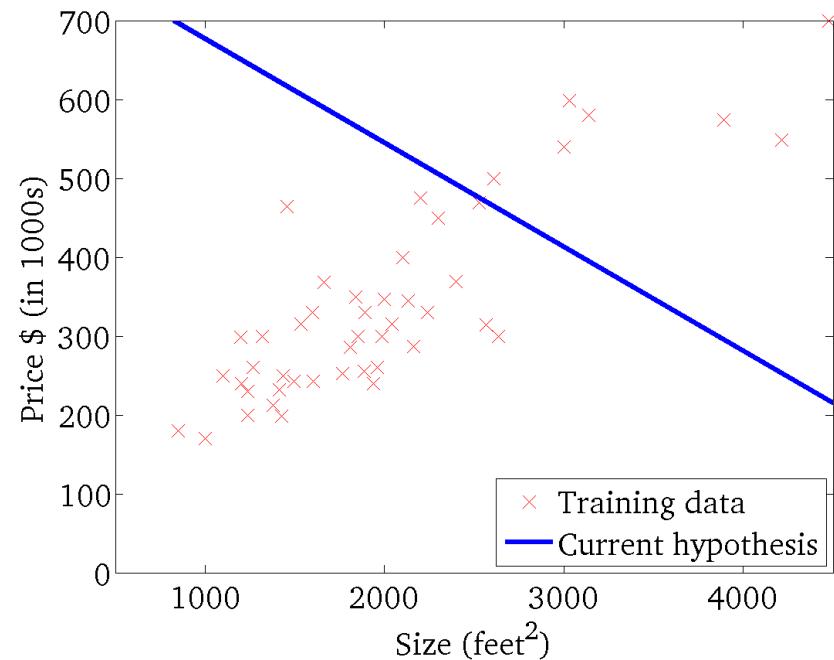
Assume $x \in \mathbb{R}$, $\theta_0 = 0$.

$$h_{\theta}(x) = \theta_0 + \theta_1 x = \theta_1 x$$

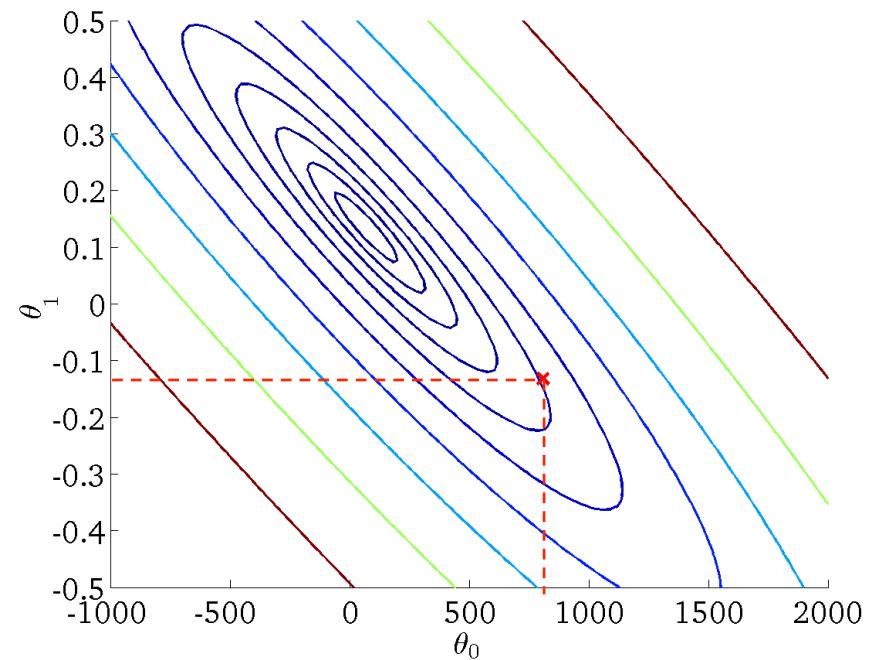


Intuition behind cost function (residual sum of squares)

$$h_{\theta}(x)$$

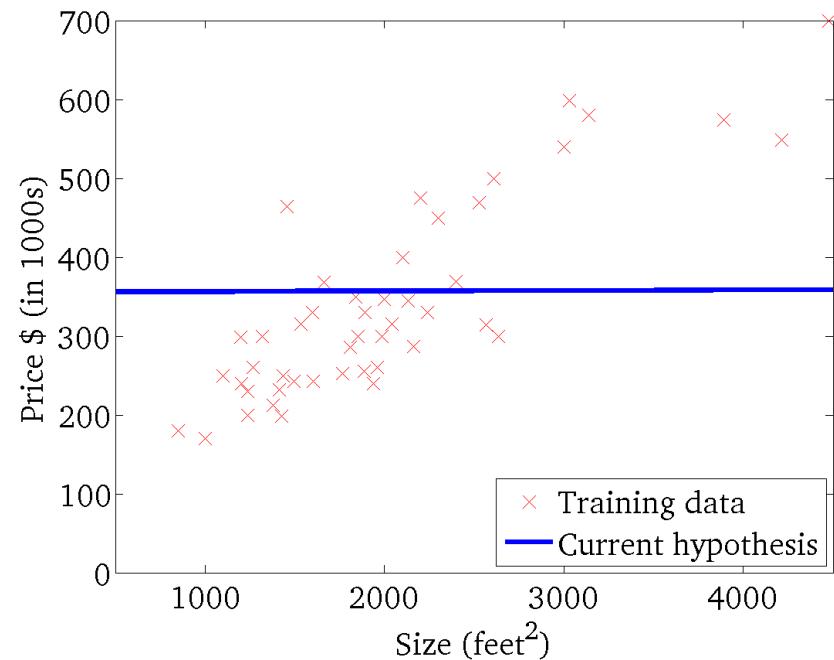


$$J(\theta_0, \theta_1)$$

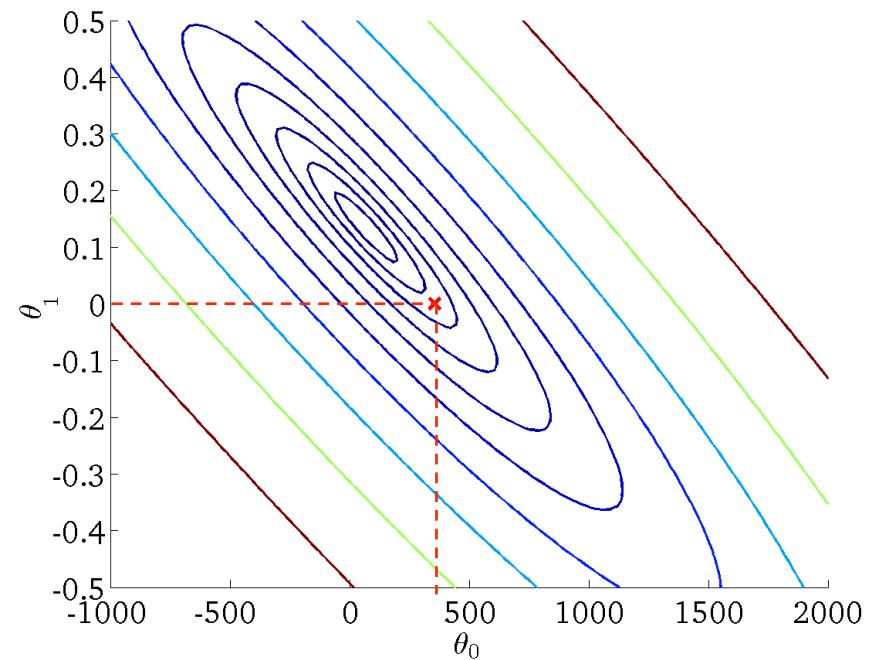


Intuition behind cost function (residual sum of squares)

$$h_{\theta}(x)$$

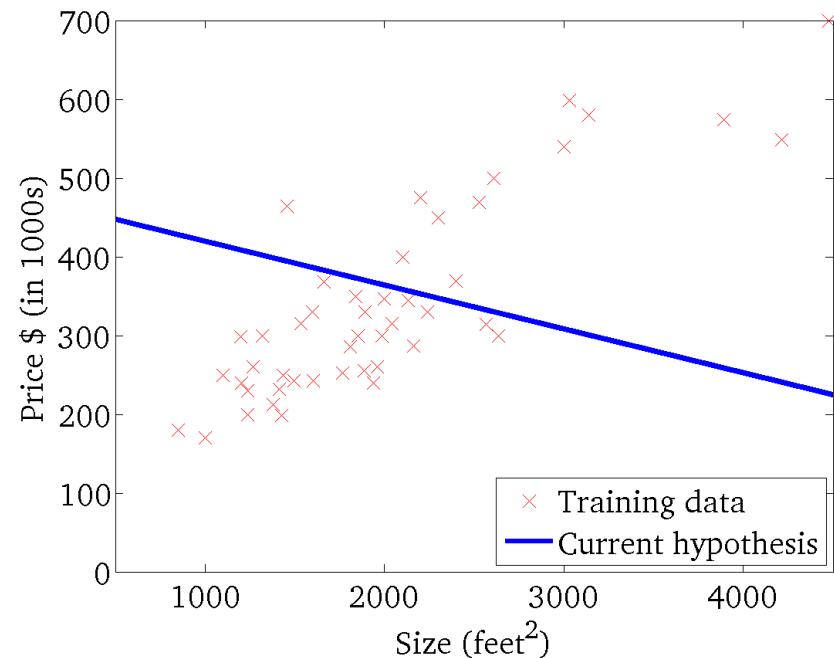


$$J(\theta_0, \theta_1)$$

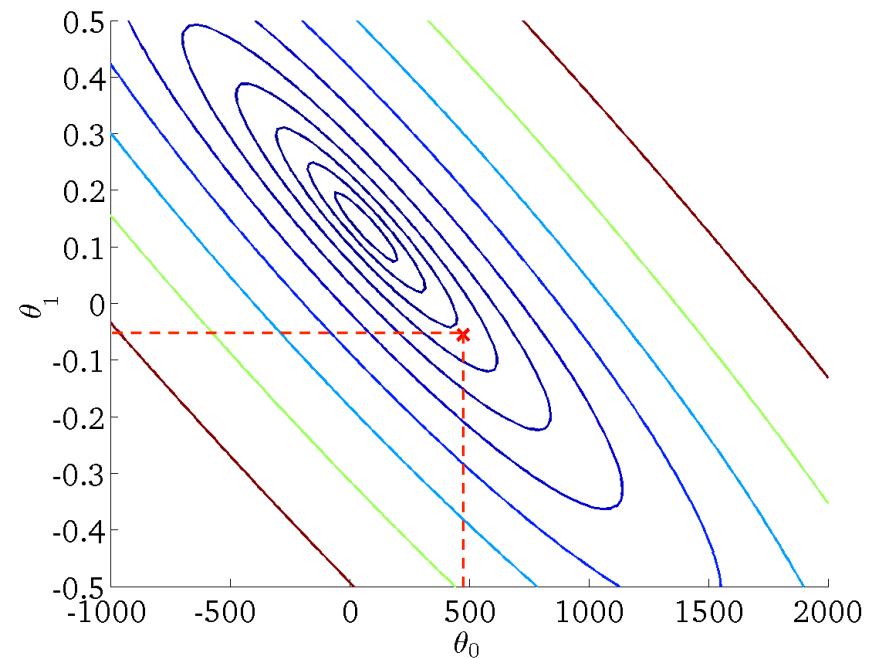


Intuition behind cost function (residual sum of squares)

$$h_{\theta}(x)$$

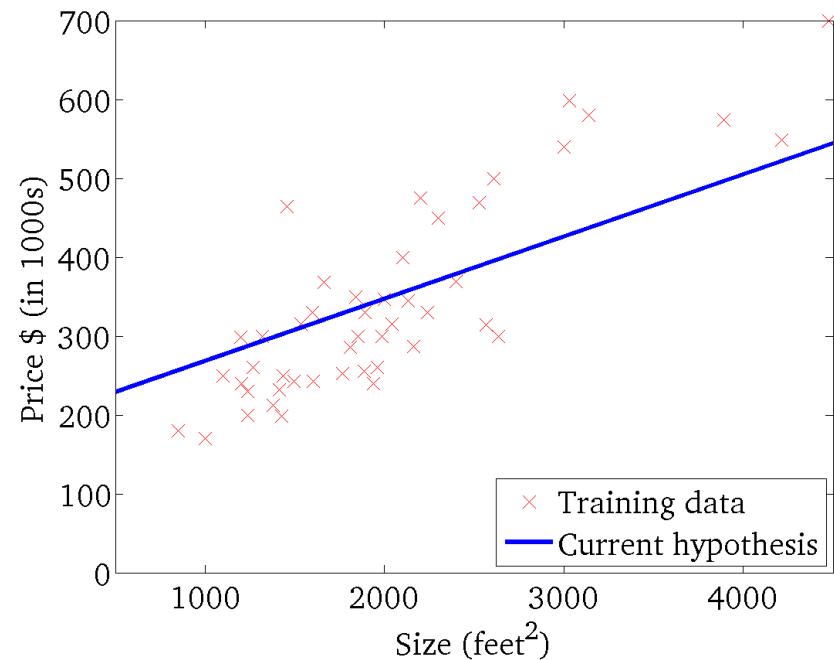


$$J(\theta_0, \theta_1)$$

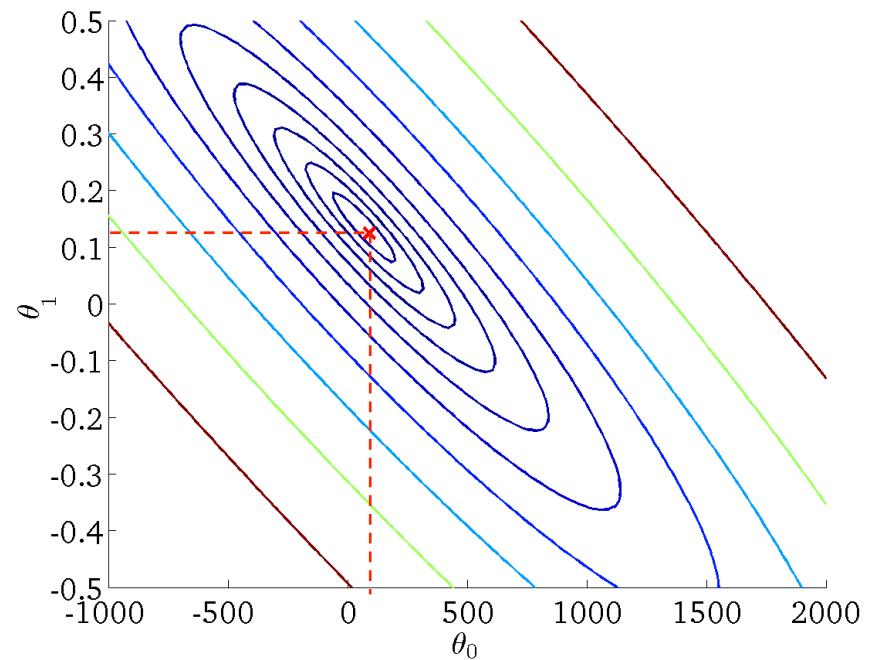


Intuition behind cost function (residual sum of squares)

$$h_{\theta}(x)$$



$$J(\theta_0, \theta_1)$$



How do we minimize the *RSS*?

Numerical optimization

Algorithm 5 Gradient Descent (J)

- 1: $t \leftarrow 0$
- 2: Initialize $\theta^{(0)}$
- 3: **repeat**
- 4: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla J(\theta^{(t)})$
- 5: $t \leftarrow t + 1$
- 6: **until** convergence
- 7: Return final value of θ

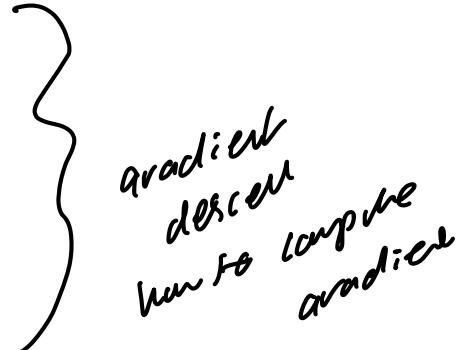
Need to compute the gradient for the linear regression cost function (residual sum of squares *RSS*)

How do we minimize the RSS ?

Numerical optimization

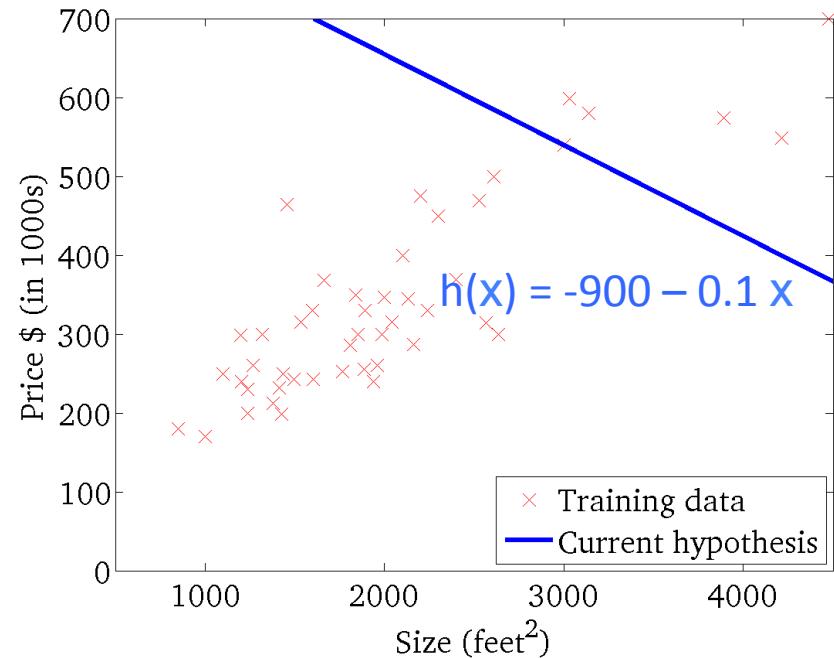
Algorithm 5 Gradient Descent (J)

```
1:  $t \leftarrow 0$ 
2: Initialize  $\theta^{(0)}$ 
3: repeat
4:    $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \sum_n (h_{\theta^{(t)}}(\mathbf{x}_n) - y_n) \mathbf{x}_n$ 
5:    $t \leftarrow t + 1$ 
6: until convergence
7: Return final value of  $\theta$ 
```

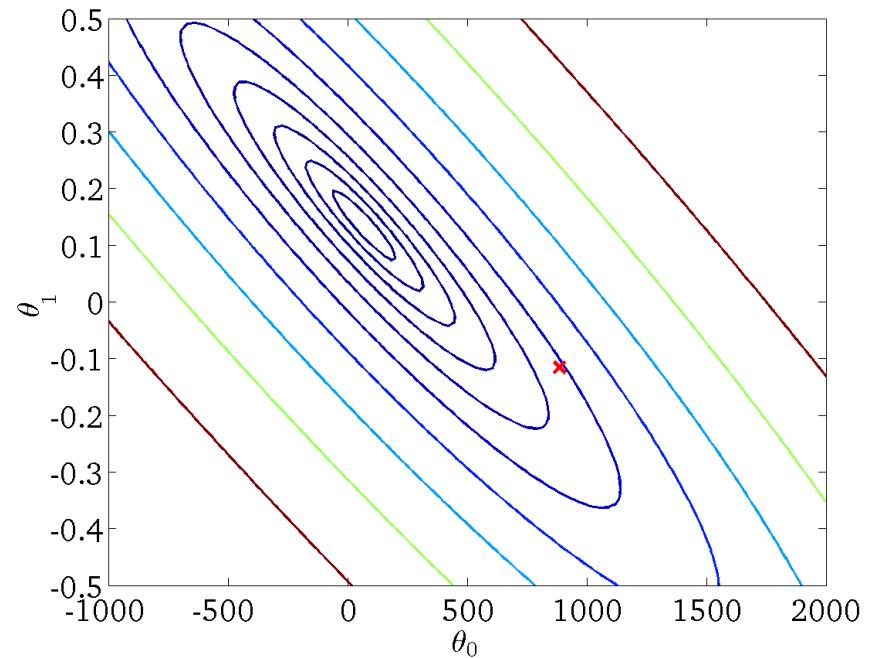


Gradient descent

$$h_{\theta}(x)$$

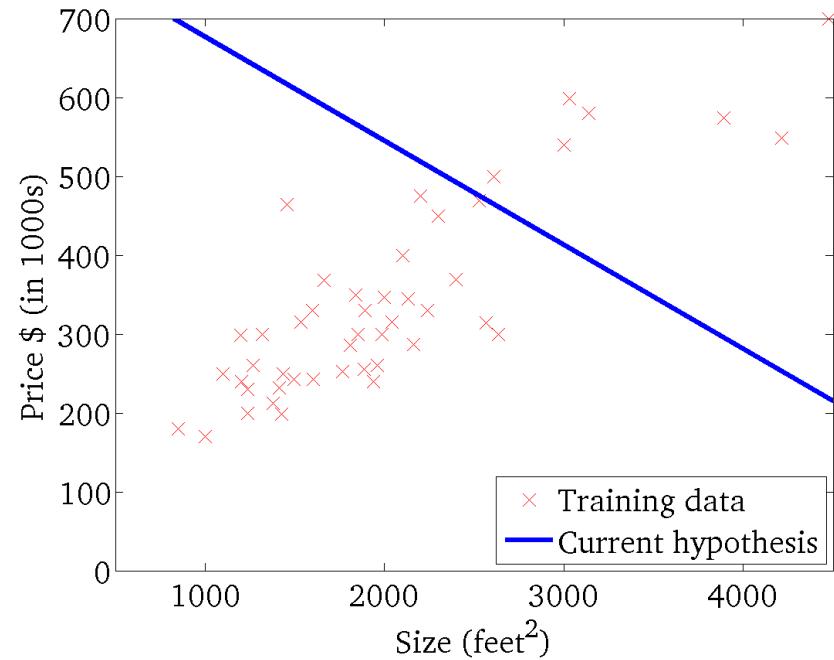


$$J(\theta_0, \theta_1)$$

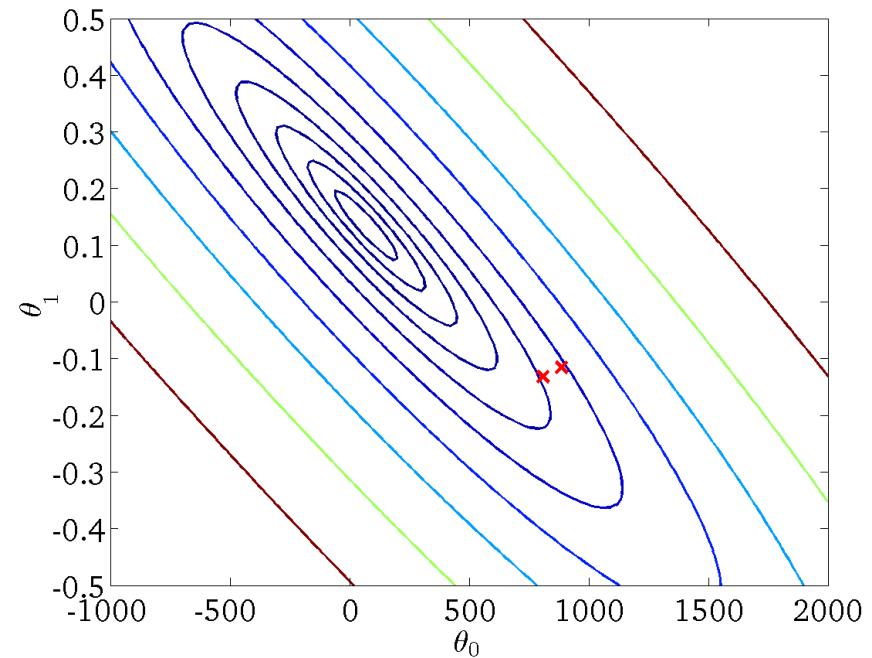


Gradient descent

$$h_{\theta}(x)$$

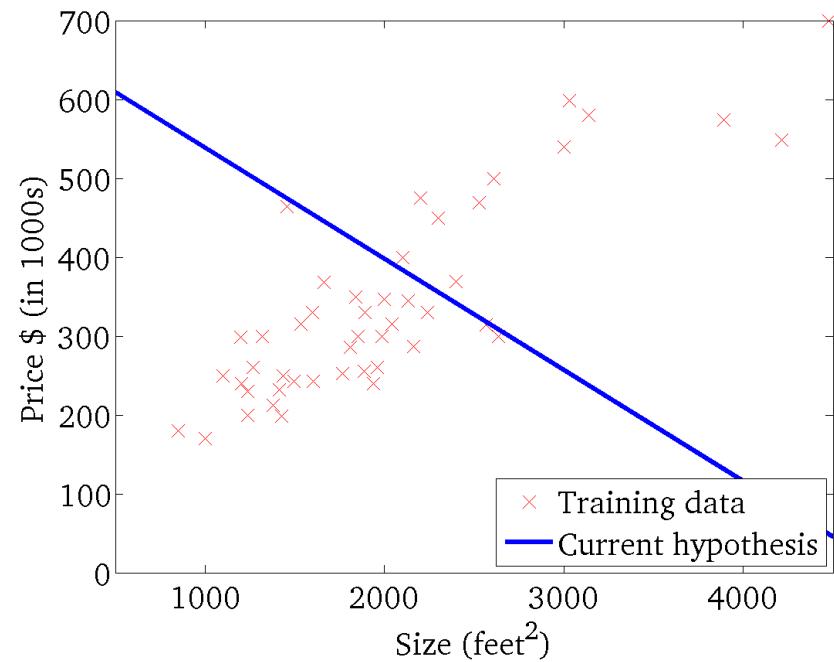


$$J(\theta_0, \theta_1)$$

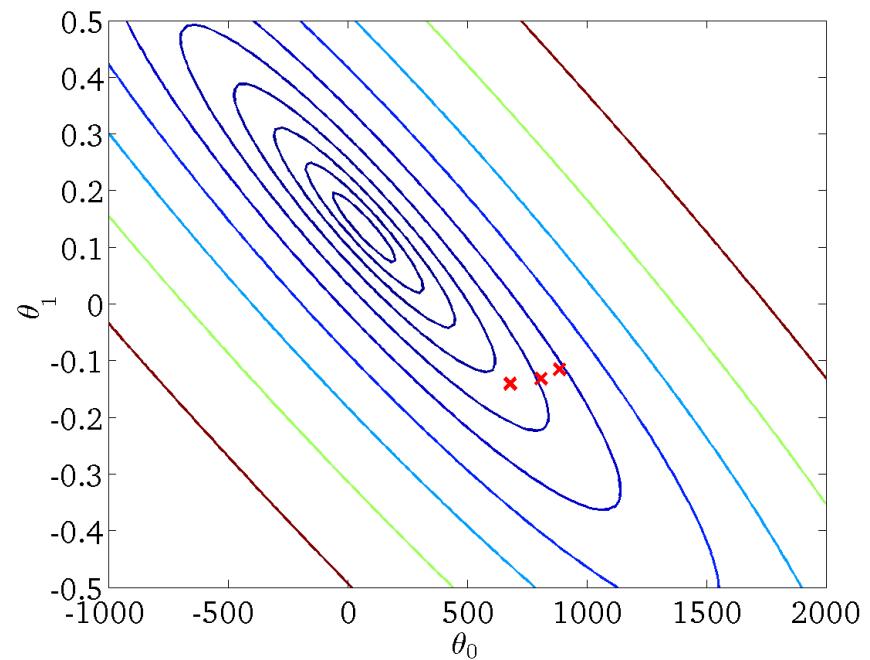


Gradient descent

$$h_{\theta}(x)$$

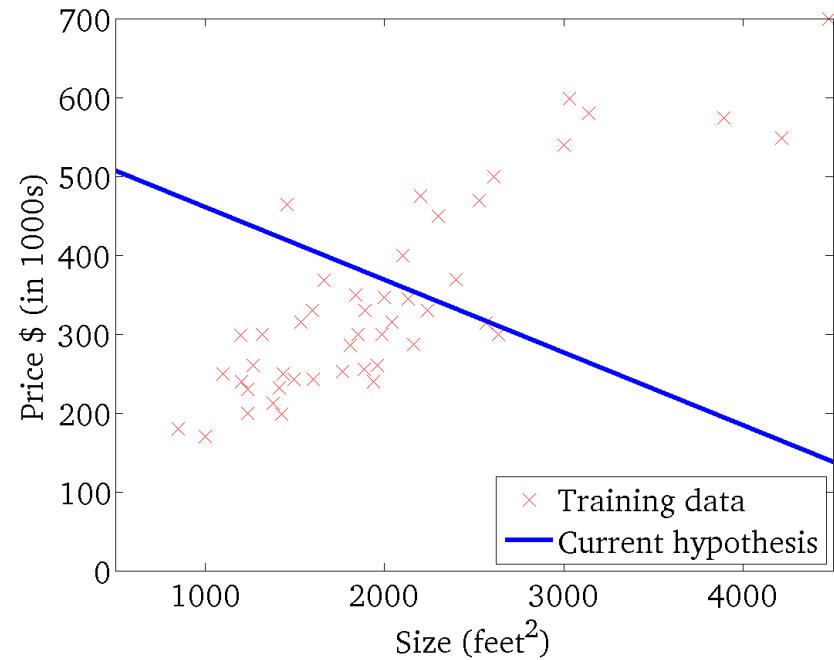


$$J(\theta_0, \theta_1)$$

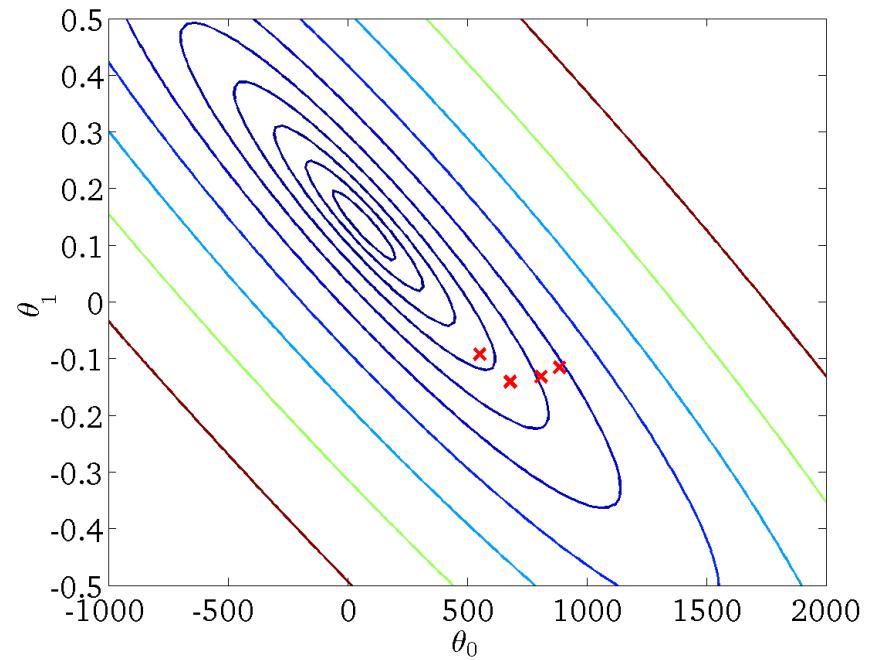


Gradient descent

$$h_{\theta}(x)$$

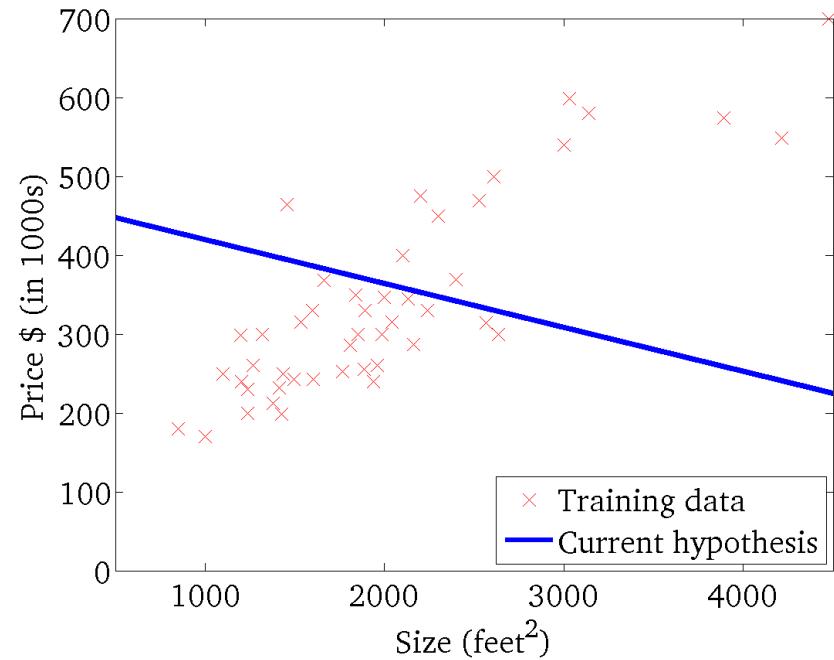


$$J(\theta_0, \theta_1)$$

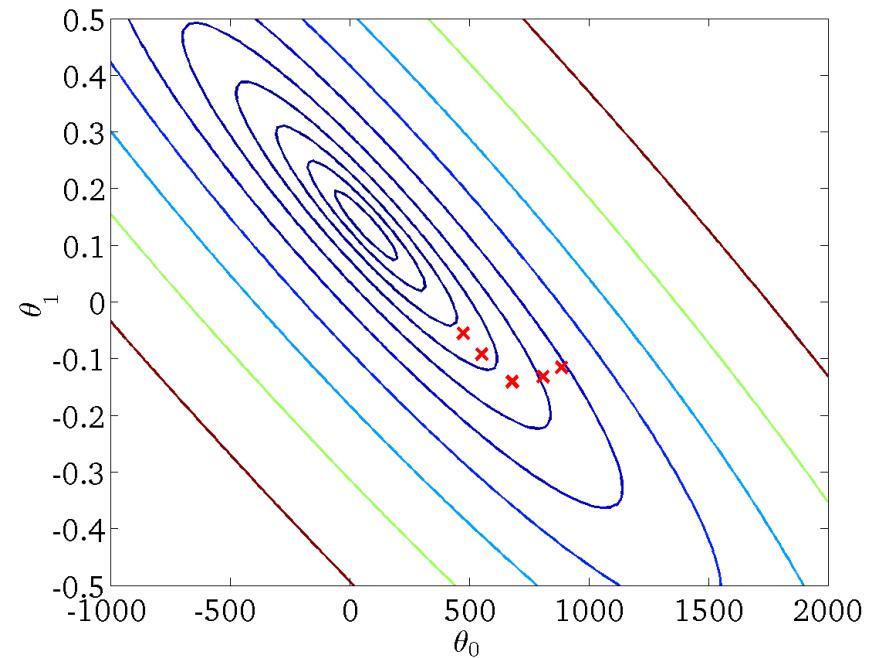


Gradient descent

$$h_{\theta}(x)$$

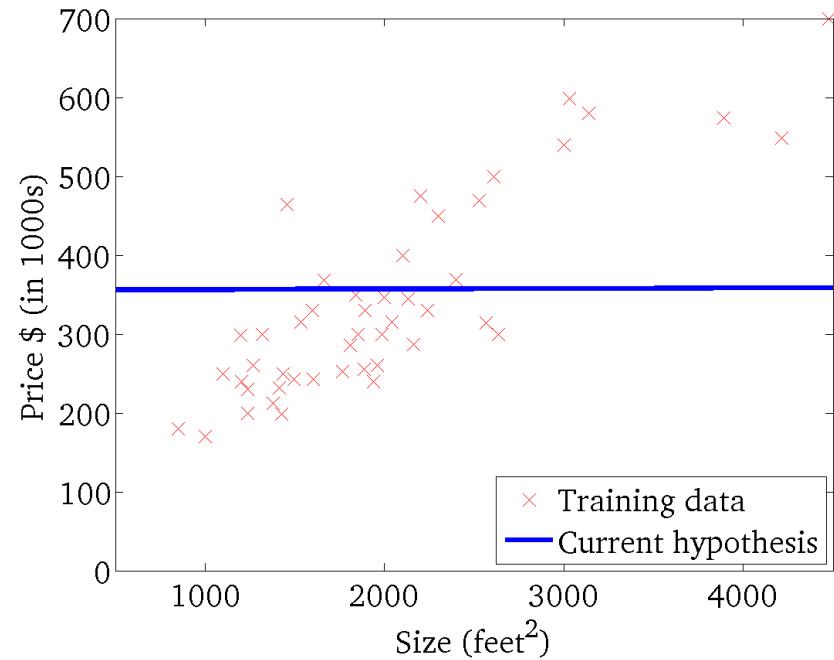


$$J(\theta_0, \theta_1)$$

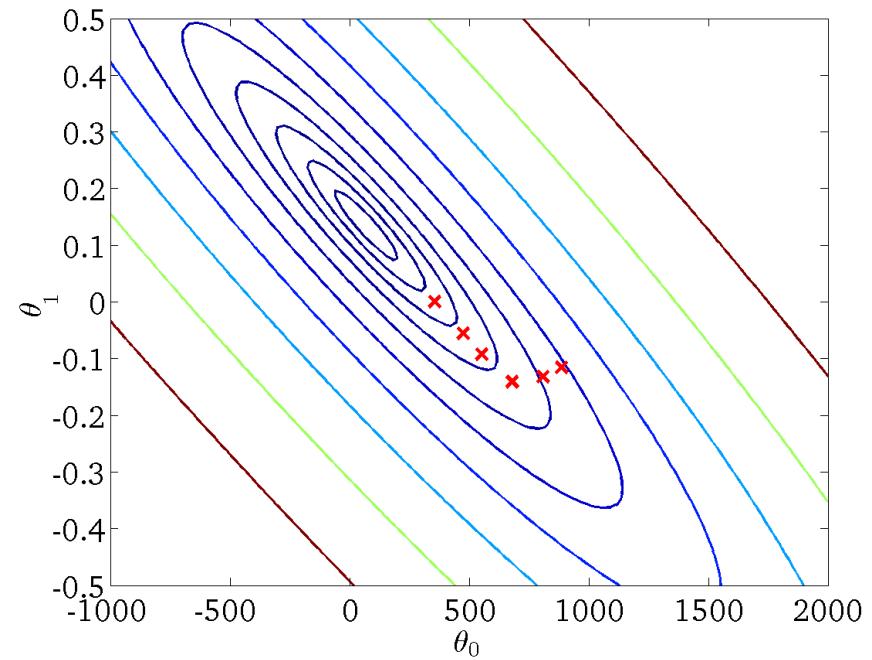


Gradient descent

$$h_{\theta}(x)$$

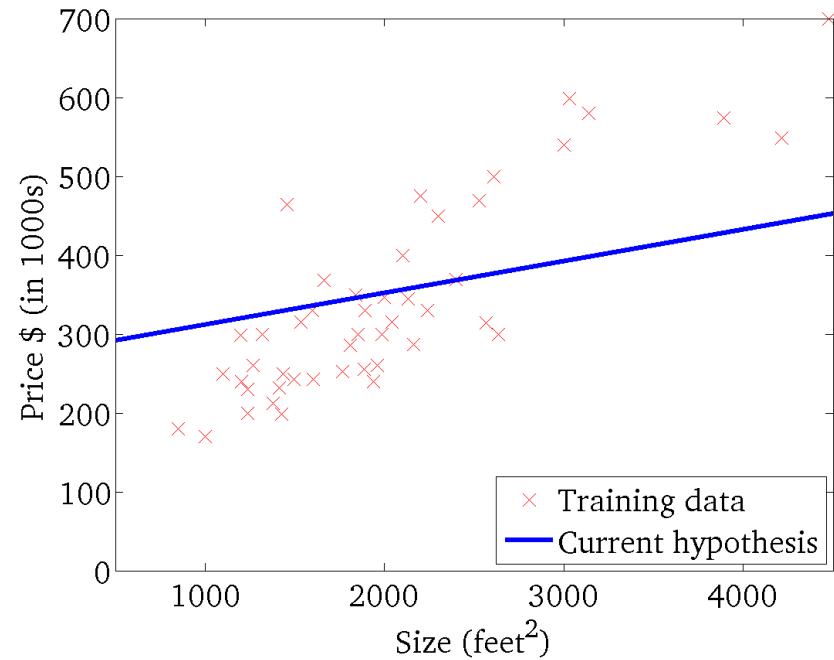


$$J(\theta_0, \theta_1)$$

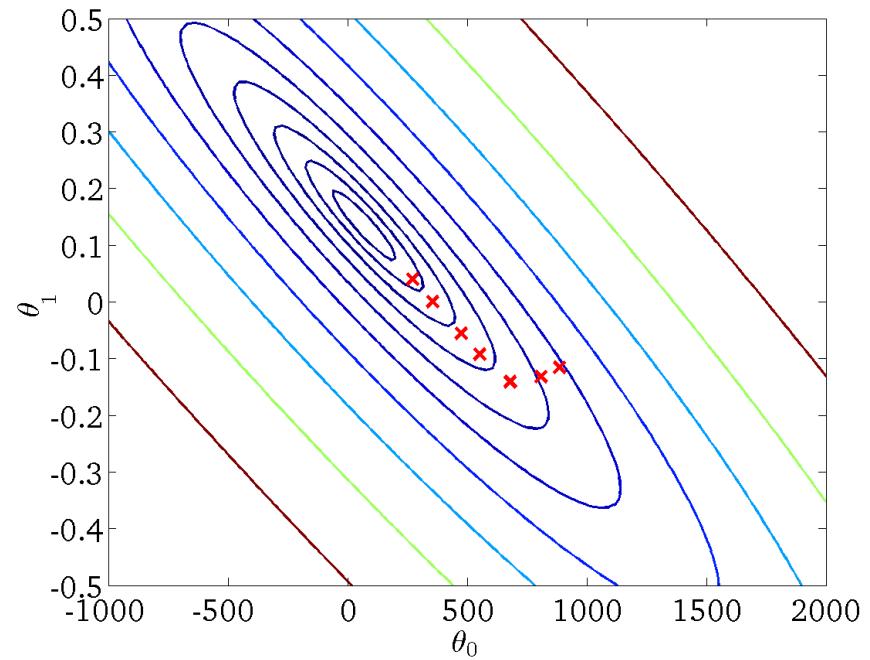


Gradient descent

$$h_{\theta}(x)$$

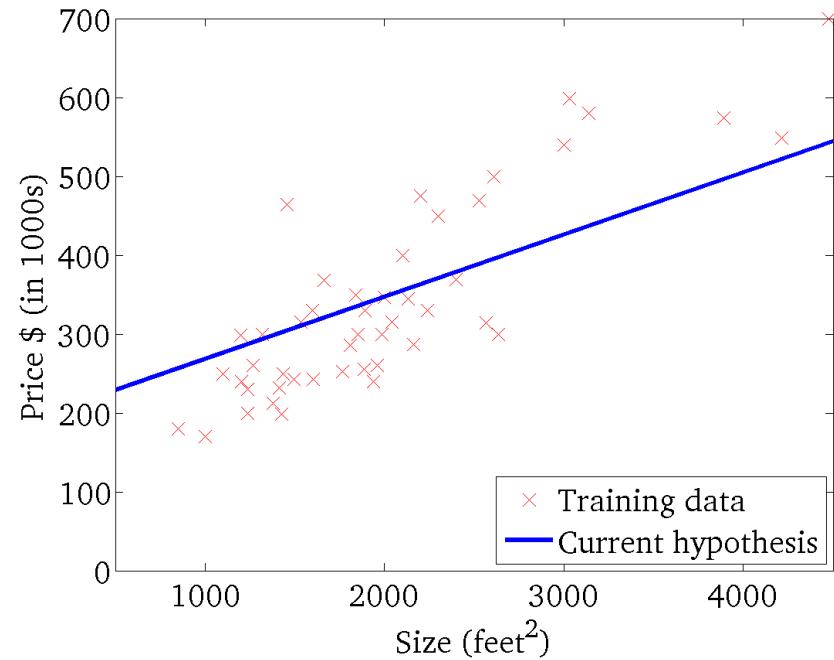


$$J(\theta_0, \theta_1)$$

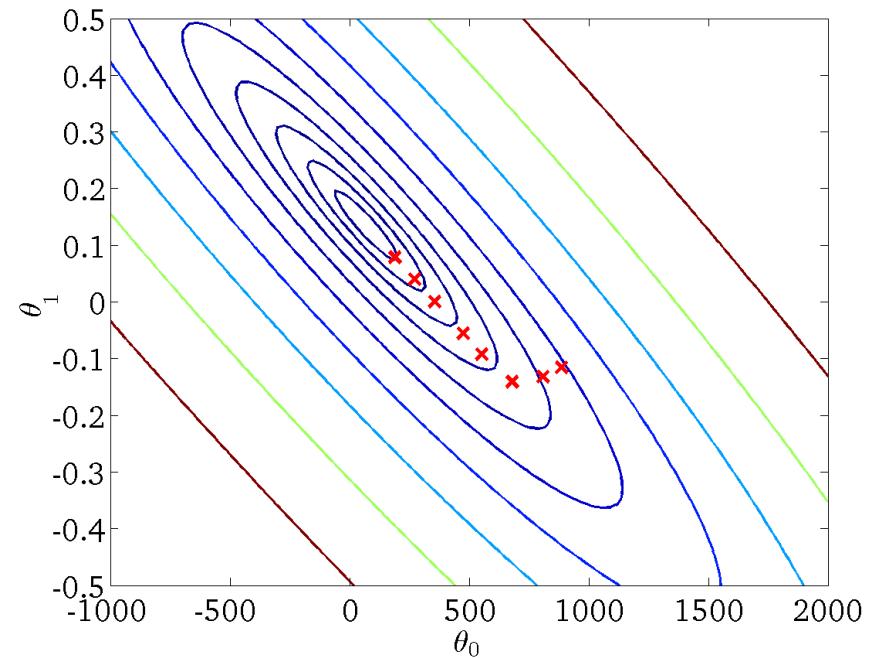


Gradient descent

$$h_{\theta}(x)$$

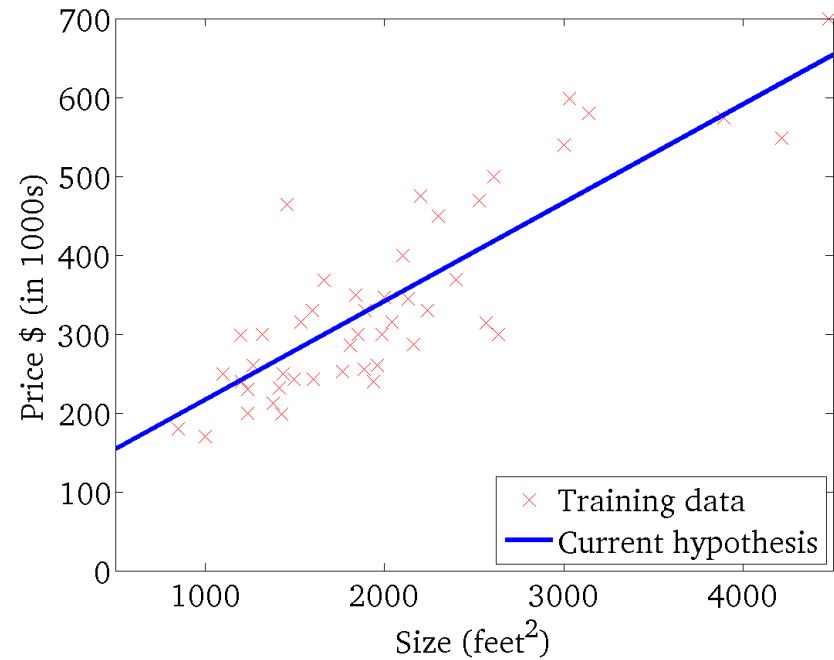


$$J(\theta_0, \theta_1)$$

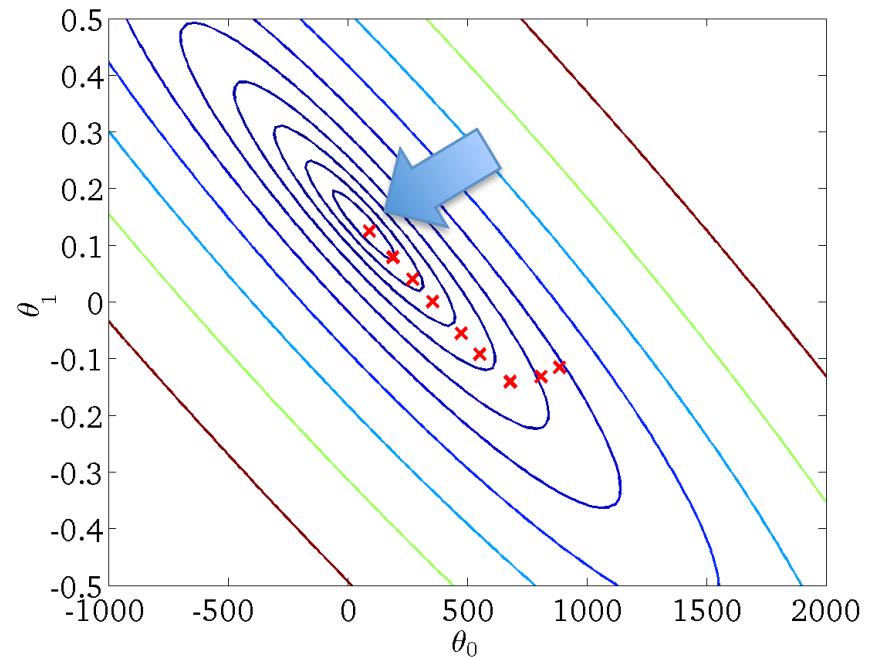


Gradient descent

$$h_{\theta}(x)$$



$$J(\theta_0, \theta_1)$$



How do we minimize the cost function (residual sum of squares)?

Numerical optimization

Gradient descent

Analytical solution

Can compute minimum in closed form for linear regression!

A simple case: x is just one-dimensional ($D=1$)

Residual sum of squares (RSS)

$$J(\boldsymbol{\theta}) = \sum_n [y_n - h_{\boldsymbol{\theta}}(x_n)]^2 = \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2$$

A simple case: x is just one-dimensional ($D=1$)

Residual sum of squares (RSS)

$$J(\boldsymbol{\theta}) = \sum_n [y_n - h_{\boldsymbol{\theta}}(x_n)]^2 = \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2$$

Identify stationary points by taking derivative with respect to parameters and setting to zero

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_0} = 0 \Rightarrow -2 \sum_n [y_n - (\theta_0 + \theta_1 x_n)] = 0$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} = 0 \Rightarrow -2 \sum_n [y_n - (\theta_0 + \theta_1 x_n)] x_n = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_0} = 0 \Rightarrow -2 \sum_n [y_n - (\theta_0 + \theta_1 x_n)] = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_1} = 0 \Rightarrow -2 \sum_n [y_n - (\theta_0 + \theta_1 x_n)] x_n = 0$$

Simplify these expressions to get “Normal Equations”

$$\frac{\partial J(\theta)}{\partial \theta_0} = 0 \Rightarrow -2 \sum_n [y_n - (\theta_0 + \theta_1 x_n)] = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_1} = 0 \Rightarrow -2 \sum_n [y_n - (\theta_0 + \theta_1 x_n)] x_n = 0$$

Simplify these expressions to get “Normal Equations”

$$\sum y_n = N\theta_0 + \theta_1 \sum x_n$$

$$\sum x_n y_n = \theta_0 \sum x_n + \theta_1 \sum x_n^2$$

$$\frac{\partial J(\theta)}{\partial \theta_0} = 0 \Rightarrow -2 \sum_n [y_n - (\theta_0 + \theta_1 x_n)] = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_1} = 0 \Rightarrow -2 \sum_n [y_n - (\theta_0 + \theta_1 x_n)] x_n = 0$$

Simplify these expressions to get “Normal Equations”

$$\sum y_n = N\theta_0 + \theta_1 \sum x_n$$

$$\sum x_n y_n = \theta_0 \sum x_n + \theta_1 \sum x_n^2$$

We have two equations and two unknowns! Do some algebra to get:

$$\theta_1 = \frac{\sum (x_n - \bar{x})(y_n - \bar{y})}{\sum (x_i - \bar{x})^2} \quad \text{and} \quad \theta_0 = \bar{y} - \theta_1 \bar{x}$$

where $\bar{x} = \frac{1}{n} \sum_n x_n$ and $\bar{y} = \frac{1}{n} \sum_n y_n$.

Why is minimizing J sensible?

Probabilistic interpretation

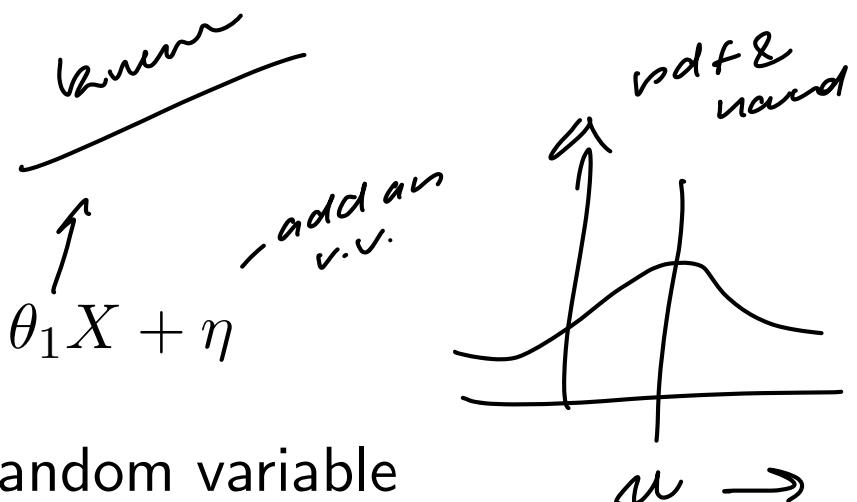
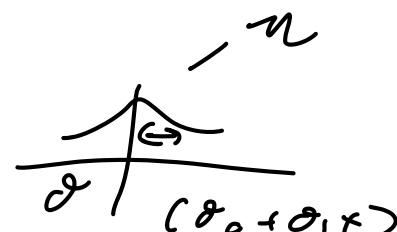
- Noisy observation model

where $\eta \sim \mathcal{N}(0, \sigma^2)$ is a Gaussian random variable

$$y = \theta_0 + \theta_1 x + \eta$$

mean of y :
 $\theta_0 + \theta_1 x$
variance:
 σ^2

$\theta_0 + \theta_1 x + \eta \sim \mathcal{N}(\theta_0 + \theta_1 x, \sigma^2)$



use of noise
by specifying mean & var
for diff. R.V.s

Why is minimizing J sensible?

Probabilistic interpretation

- Noisy observation model

$$Y = \theta_0 + \theta_1 X + \eta$$

where $\eta \sim \mathcal{N}(0, \sigma^2)$ is a Gaussian random variable

- Likelihood of one training sample (x_n, y_n)

$$p(y_n | x_n; \theta) = \mathcal{N}(\theta_0 + \theta_1 x_n, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{[y_n - (\theta_0 + \theta_1 x_n)]^2}{2\sigma^2}}$$

Probabilistic interpretation (cont'd)

Log-likelihood of the training data \mathcal{D} (assuming i.i.d)

$$\mathcal{LL}(\theta) = \log P(\mathcal{D})$$

$$= \log \prod_{n=1}^N p(y_n|x_n) = \sum_n \log p(y_n|x_n)$$

Probabilistic interpretation (cont'd)

Log-likelihood of the training data \mathcal{D} (assuming i.i.d)

$$\mathcal{LL}(\theta) = \log P(\mathcal{D})$$

$$\begin{aligned} &= \log \prod_{n=1}^N p(y_n|x_n) = \sum_n \log p(y_n|x_n) \\ &= \sum_n \left\{ -\frac{[y_n - (\theta_0 + \theta_1 x_n)]^2}{2\sigma^2} - \log \sqrt{2\pi}\sigma \right\} \end{aligned}$$

Probabilistic interpretation (cont'd)

Log-likelihood of the training data \mathcal{D} (assuming i.i.d)

$$\mathcal{LL}(\theta) = \log P(\mathcal{D})$$

$$\begin{aligned} &= \log \prod_{n=1}^N p(y_n | x_n) = \sum_n \log p(y_n | x_n) \\ &= \sum_n \left\{ -\frac{[y_n - (\theta_0 + \theta_1 x_n)]^2}{2\sigma^2} - \log \sqrt{2\pi}\sigma \right\} \\ &= -\frac{1}{2\sigma^2} \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 - \frac{N}{2} \log \sigma^2 - N \log \sqrt{2\pi} \end{aligned}$$

Probabilistic interpretation (cont'd)

Log-likelihood of the training data \mathcal{D} (assuming i.i.d)

$$\mathcal{LL}(\theta) = \log P(\mathcal{D})$$

$$\begin{aligned} &= \log \prod_{n=1}^N p(y_n | x_n) = \sum_n \log p(y_n | x_n) \\ &= \sum_n \left\{ -\frac{[y_n - (\theta_0 + \theta_1 x_n)]^2}{2\sigma^2} - \log \sqrt{2\pi}\sigma \right\} \\ &= -\frac{1}{2\sigma^2} \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 - \frac{N}{2} \log \sigma^2 - N \log \sqrt{2\pi} \\ &= -\frac{1}{2} \left\{ \frac{1}{\sigma^2} \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 + N \log \sigma^2 \right\} + \text{const} \end{aligned}$$

What is the relationship between minimizing J and maximizing the log-likelihood?

Maximum likelihood estimation

Estimating σ , θ_0 and θ_1 can be done in two steps

- Maximize over θ_0 and θ_1

$$\max \log P(\mathcal{D}) \Leftrightarrow \min \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 \leftarrow \text{That is } J(\theta)!$$

Maximum likelihood estimation

Estimating σ , θ_0 and θ_1 can be done in two steps

- Maximize over θ_0 and θ_1

$$\max \log P(\mathcal{D}) \Leftrightarrow \min \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 \leftarrow \text{That is } J(\theta)!$$

- Maximize over $s = \sigma^2$ (we could estimate σ directly)

$$\frac{\partial \log P(\mathcal{D})}{\partial s} = -\frac{1}{2} \left\{ -\frac{1}{s^2} \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 + N \frac{1}{s} \right\} = 0$$

Maximum likelihood estimation

Estimating σ , θ_0 and θ_1 can be done in two steps

- Maximize over θ_0 and θ_1

$$\max \log P(\mathcal{D}) \Leftrightarrow \min \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 \leftarrow \text{That is } J(\theta)!$$

- Maximize over $s = \sigma^2$ (we could estimate σ directly)

$$\frac{\partial \log P(\mathcal{D})}{\partial s} = -\frac{1}{2} \left\{ -\frac{1}{s^2} \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 + N \frac{1}{s} \right\} = 0$$

$$\rightarrow \sigma^{*2} = s^* = \frac{1}{N} \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2$$

Why does the probabilistic interpretation help us?

Gives us a template for modeling

- Probabilistic model $P(x; \theta)$
 - ▶ “Story” for generating the data
- Given data x and probabilistic model $P(x; \theta)$, we can find a “good” value for θ .
 - ▶ Maximize the likelihood
- We have seen this for logistic regression, linear regression (minimizing the RSS) but this principle is very general.
- Makes clear what the assumptions are.

Why does the probabilistic interpretation help us?

Assumptions underlying linear regression

y_n are

- Independent
- Normally distributed
- Mean is a linear function of x_n
- Constant variance σ^2

Outline

1 Logistic regression

2 Linear regression

3 Multivariate solution

- Input is D-dimensional
- Computational and numerical optimization

Linear regression when x is D-dimensional

$J(\theta)$) in matrix form

$$J(\theta) = \sum_n [y_n - (\theta_0 + \sum_d \theta_d x_{nd})]^2 = \sum_n [y_n - \theta^T x_n]^2$$

where we have redefined some variables (by augmenting)

$$x \leftarrow [1 \ x_1 \ x_2 \ \dots \ x_D]^T, \quad \theta \leftarrow [\theta_0 \ \theta_1 \ \theta_2 \ \dots \ \theta_D]^T$$

$J(\theta)$ in new notations

Design matrix and target vector

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{pmatrix} \in \mathbb{R}^{N \times (D+1)}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

$J(\theta)$ in new notations

Design matrix and target vector

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{pmatrix} \in \mathbb{R}^{N \times (D+1)}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

Vector of predictions for a given θ

$$\mathbf{X}\theta = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{pmatrix} \theta = \begin{pmatrix} \mathbf{x}_1^T \theta \\ \mathbf{x}_2^T \theta \\ \vdots \\ \mathbf{x}_N^T \theta \end{pmatrix}$$

$J(\theta)$ in new notations

Vector of errors for a given θ

$$\mathbf{y} - \mathbf{X}\theta = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} - \begin{pmatrix} \mathbf{x}_1^T \theta \\ \mathbf{x}_2^T \theta \\ \vdots \\ \mathbf{x}_N^T \theta \end{pmatrix} = \begin{pmatrix} y_1 - \mathbf{x}_1^T \theta \\ y_2 - \mathbf{x}_2^T \theta \\ \vdots \\ y_N - \mathbf{x}_N^T \theta \end{pmatrix}$$

$J(\theta)$ in new notations

Vector of errors for a given θ

$$\mathbf{y} - \mathbf{X}\theta = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} - \begin{pmatrix} \mathbf{x}_1^T \theta \\ \mathbf{x}_2^T \theta \\ \vdots \\ \mathbf{x}_N^T \theta \end{pmatrix} = \begin{pmatrix} y_1 - \mathbf{x}_1^T \theta \\ y_2 - \mathbf{x}_2^T \theta \\ \vdots \\ y_N - \mathbf{x}_N^T \theta \end{pmatrix}$$

Expression for $J(\theta)$

$$\begin{aligned} & \| \mathbf{y} - \mathbf{X}\theta \|_2^2 \\ &= (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) \\ &= (y_1 - \mathbf{x}_1^T \theta \quad \cdots \quad y_N - \mathbf{x}_N^T \theta) \begin{pmatrix} y_1 - \mathbf{x}_1^T \theta \\ \vdots \\ y_N - \mathbf{x}_N^T \theta \end{pmatrix} \\ &= \sum_n [y_n - \mathbf{x}_n^T \theta]^2 \leftarrow \text{That is } J(\theta) \end{aligned}$$

$J(\theta)$ in new notations

Compact expression

$$\begin{aligned} J(\theta) &= \|\mathbf{y} - \mathbf{X}\theta\|_2^2 \\ &= (\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - \mathbf{X}\theta) \\ &= (\mathbf{y}^T - \{\mathbf{X}\theta\}^T)(\mathbf{y} - \mathbf{X}\theta) \\ &= (\mathbf{y}^T - \theta^T \mathbf{X}^T)(\mathbf{y} - \mathbf{X}\theta) \\ &= (\mathbf{y}^T)(\mathbf{y} - \mathbf{X}\theta) - \theta^T \mathbf{X}^T(\mathbf{y} - \mathbf{X}\theta) \\ &= \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\theta - \theta^T \mathbf{X}^T \mathbf{y} + \theta^T \mathbf{X}^T \mathbf{X}\theta \\ &= \text{constant} - 2\mathbf{y}^T \mathbf{X}\theta + \theta^T \mathbf{X}^T \mathbf{X}\theta \end{aligned}$$

Solution in matrix form

Compact expression

$$J(\theta) = \|X\theta - y\|_2^2 = \left\{ \theta^T X^T X \theta - 2(X^T y)^T \theta \right\} + \text{constant}$$

Gradients of Linear and Quadratic Functions

- $\nabla b^T x = b$
- $\nabla x^T A x = 2Ax$ (symmetric A)

Solution in matrix form

Compact expression

$$J(\boldsymbol{\theta}) = \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 = \left\{ \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X} \boldsymbol{\theta} - 2 (\mathbf{X}^\top \mathbf{y})^\top \boldsymbol{\theta} \right\} + \text{constant}$$

Gradients of Linear and Quadratic Functions

- $\nabla \mathbf{b}^\top \mathbf{x} = \mathbf{b}$
- $\nabla \mathbf{x}^\top \mathbf{A} \mathbf{x} = 2 \mathbf{A} \mathbf{x}$ (symmetric \mathbf{A})

Normal equations

$$\nabla J(\boldsymbol{\theta}) = 2 \mathbf{X}^\top \mathbf{X} \boldsymbol{\theta} - 2 \mathbf{X}^\top \mathbf{y} = 0$$

Solution in matrix form

Compact expression

$$J(\boldsymbol{\theta}) = \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 = \left\{ \boldsymbol{\theta}^T \mathbf{X}^T \mathbf{X} \boldsymbol{\theta} - 2 (\mathbf{X}^T \mathbf{y})^T \boldsymbol{\theta} \right\} + \text{constant}$$

Gradients of Linear and Quadratic Functions

- $\nabla \mathbf{b}^T \mathbf{x} = \mathbf{b}$
- $\nabla \mathbf{x}^T \mathbf{A} \mathbf{x} = 2 \mathbf{A} \mathbf{x}$ (symmetric \mathbf{A})

Normal equations

$$\nabla J(\boldsymbol{\theta}) = 2 \mathbf{X}^T \mathbf{X} \boldsymbol{\theta} - 2 \mathbf{X}^T \mathbf{y} = 0$$

This leads to the linear regression solution

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Mini-Summary

- Linear regression is the linear combination of features $h : \mathbf{x} \rightarrow y$, with $h(\mathbf{x}) = \theta_0 + \sum_d \theta_d x_d = \boldsymbol{\theta}^T \mathbf{x}$
- If we minimize residual sum of squares as our learning objective, we get a closed-form solution of parameters
- Probabilistic interpretation: maximum likelihood if assuming the output is Gaussian distributed

Computational complexity

Bottleneck of computing the solution?

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

Computational complexity

Bottleneck of computing the solution?

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

Matrix multiply of $X^T X \in \mathbb{R}^{(D+1) \times (D+1)}$

Inverting the matrix $X^T X$

How many operations do we need?

Computational complexity

Bottleneck of computing the solution?

$$\hat{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Matrix multiply of $\mathbf{X}^T \mathbf{X} \in \mathbb{R}^{(D+1) \times (D+1)}$

Inverting the matrix $\mathbf{X}^T \mathbf{X}$

How many operations do we need?

- $O(ND^2)$ for matrix multiplication
- $O(D^3)$ for matrix inversion
- Impractical for very large D or N

Alternative method: numerical optimization

(Batch) Gradient descent

Algorithm 6 Gradient Descent (J)

- 1: $t \leftarrow 0$
- 2: Initialize $\theta^{(0)}$
- 3: **repeat**
- 4: $\nabla J(\theta^{(t)}) = \mathbf{X}^T \mathbf{X} \theta^{(t)} - \mathbf{X}^T \mathbf{y} = \sum_n (\mathbf{x}_n^T \theta^{(t)} - y_n) \mathbf{x}_n$
- 5: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla J(\theta^{(t)})$
- 6: $t \leftarrow t + 1$
- 7: **until** convergence
- 8: Return final value of θ

What is the complexity of each iteration?

Why would this work?

If gradient descent converges, it will converge to the same solution as using matrix inversion.

This is because $J(\theta)$ is a convex function in its parameters θ

Stochastic gradient descent

Update parameters using one example at a time

Algorithm 7 Stochastic Gradient Descent (J)

- 1: $t \leftarrow 0$
 - 2: Initialize $\theta^{(0)}$
 - 3: **repeat**
 - 4: Randomly choose a training a sample x_t
 - 5: Compute its contribution to the gradient $g_t = (x_t^T \theta^{(t)} - y_t)x_t$
 - 6: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta g_t$
 - 7: $t \leftarrow t + 1$
 - 8: **until** convergence
 - 9: Return final value of θ
-

How does the complexity per iteration of stochastic gradient descent (SGD) compare with gradient descent (GD)?

How does the complexity per iteration of stochastic gradient descent (SGD) compare with gradient descent (GD)?

- $O(ND)$ for GD versus $O(D)$ for SGD

Mini-summary

- Batch gradient descent computes the exact gradient.
- Stochastic gradient descent approximates the gradient with a single data point;
- Mini-batch variant: trade-off between accuracy of estimating gradient and computational cost
- Similar ideas extend to other ML optimization problems.
 - ▶ For large-scale problems, stochastic gradient descent often works well.

Summary

- Use of linear models for classification and regression.
- Learning is a problem of optimization.
 - ▶ The objective function is convex.
 - ▶ Numerical methods and sometimes analytical solutions.

Linear regression cost function (residual sum of squares)

- Input: $x \in \mathbb{R}$
- Output: $y \in \mathbb{R}$
- Hypotheses/Model: h_{θ} , with $h_{\theta}(x) = \theta_0 + \theta_1 x_1 = \theta^T x$

Residual sum of squares (RSS)

and min: θ

$$J(\theta) = \sum_n [y_n - \hat{h}_{\theta}(x_n)]^2 = \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2$$

Why is minimizing J sensible?

Probabilistic interpretation

- Noisy observation model

$$Y = \theta_0 + \theta_1 X + \eta$$

where $\eta \sim \mathcal{N}(0, \sigma^2)$ is a Gaussian random variable

Why is minimizing J sensible?

$$y_1 = \theta_0 + \theta_1 x_1 + n_1 \quad (\text{true})$$
$$y_2 = \theta_0 + \theta_1 x_2 + n_2 \quad (\text{observed})$$

assumptions

Probabilistic interpretation

- Noisy observation model

$$Y = \underbrace{\theta_0 + \theta_1 X}_{\text{true } \theta} + \eta \quad : \mathcal{N}(\theta_0, \theta_1, \sigma^2)$$

?

problem:

someone gave you training

data w/o σ , you

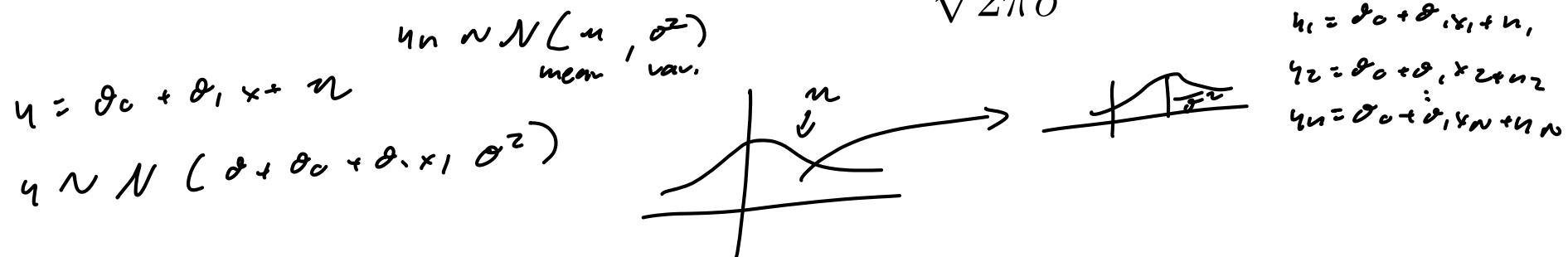
need to figure out

σ

where $\eta \sim \mathcal{N}(0, \sigma^2)$ is a Gaussian random variable

- Likelihood of one training sample (x_n, y_n)

$$p(y_n | x_n; \theta) = \mathcal{N}(\theta_0 + \theta_1 x_n, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{[y_n - (\theta_0 + \theta_1 x_n)]^2}{2\sigma^2}}$$



Probabilistic interpretation (cont'd)

Log-likelihood of the training data \mathcal{D} (assuming i.i.d)

$$\text{argmax}^{\theta} \mathcal{LL}(\theta) = \log P(\mathcal{D})$$

$$= \log \prod_{n=1}^N p(y_n | x_n) = \sum_n \log p(y_n | x_n)$$

$\text{argmin } J^{(\theta)}$
 $\in \{y_n - (\theta_0 + \theta_1 x_n)\}$
 maximize log likelihood
 minimize J

$$= \sum_n \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(\frac{-(y_n - (\theta_0 + \theta_1 x_n))^2}{2\sigma^2} \right) \right)$$

$$= \sum_n \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(\frac{-(y_n - (\theta_0 + \theta_1 x_n))^2}{2\sigma^2} \right) \right) + \text{const}$$

$$= \sum_n \left\{ -\frac{[y_n - (\theta_0 + \theta_1 x_n)]^2}{2\sigma^2} - \log \sqrt{2\pi\sigma^2} \right\} = -\frac{1}{2} \log 2\pi + \frac{1}{2} \log \sigma^2$$

$$\log \sqrt{2\pi\sigma^2} \approx \log (2\pi\sigma^2)$$

$$= \left\{ \frac{1}{2\sigma^2} \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 - \frac{N}{2} \log \sigma^2 - N \log \sqrt{2\pi} \right\}$$

$$= -\frac{1}{2} \left\{ \frac{1}{\sigma^2} \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 + N \log \sigma^2 \right\} + \text{const}$$

likelihood?

relationship between minimizing J & maximizing \log
 find θ to max. \log likelihood

Maximum likelihood estimation

Estimating σ , θ_0 and θ_1 can be done in two steps

- Maximize over θ_0 and θ_1

$$\max \log P(\mathcal{D}) \Leftrightarrow \min \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 \leftarrow \text{That is } J(\theta)!$$

- Maximize over $s = \sigma^2$ (we could estimate σ directly)

$$\frac{\partial \log P(\mathcal{D})}{\partial s} = -\frac{1}{2} \left\{ -\frac{1}{s^2} \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 + N \frac{1}{s} \right\} = 0$$

Maximum likelihood estimation

Estimating σ , θ_0 and θ_1 can be done in two steps

- Maximize over θ_0 and θ_1

$$\max \log P(\mathcal{D}) \Leftrightarrow \min \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 \leftarrow \text{That is } J(\theta)!$$

- Maximize over $s = \sigma^2$ (we could estimate σ directly)

$$\frac{\partial \log P(\mathcal{D})}{\partial s} = -\frac{1}{2} \left\{ -\frac{1}{s^2} \sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2 + N \frac{1}{s} \right\} = 0$$

$\rightarrow \sigma^{*2} = s^* = \frac{1}{N} \underbrace{\sum_n [y_n - (\theta_0 + \theta_1 x_n)]^2}_{\text{square each}} \underbrace{\text{and average}}_{\text{over } n \text{ training pairs}}$

optimal var?
& $\theta_0 + \theta_1 x + \epsilon$
lin. comb.

*averaging over trained
& compute MSE*

Why does the probabilistic interpretation help us?

Gives us a template for modeling

- Probabilistic model $P(x; \theta)$
 - ▶ “Story” for generating the data
- Given data x and probabilistic model $P(x; \theta)$, we can find a “good” value for θ .
 - ▶ Maximize the likelihood
- We have seen this for logistic regression, linear regression (minimizing the RSS) but this principle is very general.
- Makes clear what the assumptions are.

answer residual
same params
assumption

$$\sum_{n=1}^N \text{Loss} - (\theta_0 + \theta_1 x_n)^2$$
$$y_n = \theta_0 + \theta_1 x_n + \epsilon_n \sim N(\theta, \sigma^2)$$

Why does the probabilistic interpretation help us?

Assumptions underlying linear regression

y_n are

- Independent*
- Normally distributed*
- Mean is a linear function of $x_n \rightarrow \theta_0 + \theta_1 x_n$
- Constant variance $\sigma^2 \rightarrow$ variance is constant

* can be false
data analysis over time
validity: depends on assumptions
what if: we remove an assumption?
 \rightarrow new model

Linear regression when x is D-dimensional

$J(\theta)$) in matrix form

$$J(\theta) = \sum_n [y_n - (\underbrace{\theta_0 + \sum_d \theta_d x_{nd}}_{\text{label}})^2] = \sum_n [y_n - \overbrace{\theta^T x_n}^{\substack{\text{comb.} \\ \text{inner prod.} \\ \text{or feature vector}}}]^2$$

where we have redefined some variables (by augmenting)

$$\underbrace{x \leftarrow [1 \ x_1 \ x_2 \ \dots \ x_D]^T}_{D}, \quad \theta \leftarrow [\theta_0 \ \theta_1 \ \theta_2 \ \dots \ \theta_D]^T$$

, append w/ 1

$J(\theta)$ in new notations

$x_i = (x_{i0}, x_{i1}, \dots, x_{ik})$

Design matrix and target vector

$$X = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{pmatrix} \in \mathbb{R}^{N \times \underline{\underline{(D+1)}}}, \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

Vector of predictions for a given θ

$$X\theta = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{pmatrix} \theta = \begin{pmatrix} x_1^T \theta \\ x_2^T \theta \\ \vdots \\ x_N^T \theta \end{pmatrix}$$

$J(\theta)$ in new notations

Vector of errors for a given θ

$$\mathbf{y} - \mathbf{X}\theta = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} - \begin{pmatrix} \mathbf{x}_1^T \theta \\ \mathbf{x}_2^T \theta \\ \vdots \\ \mathbf{x}_N^T \theta \end{pmatrix} = \begin{pmatrix} y_1 - \mathbf{x}_1^T \theta \\ y_2 - \mathbf{x}_2^T \theta \\ \vdots \\ y_N - \mathbf{x}_N^T \theta \end{pmatrix}$$

Expression for $J(\theta)$

$$\begin{aligned} & \| \mathbf{y} - \mathbf{X}\theta \|_2^2 \\ &= (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) \\ &= (y_1 - \mathbf{x}_1^T \theta \quad \cdots \quad y_N - \mathbf{x}_N^T \theta) \begin{pmatrix} y_1 - \mathbf{x}_1^T \theta \\ \vdots \\ y_N - \mathbf{x}_N^T \theta \end{pmatrix} \\ &= \sum_n [y_n - \mathbf{x}_n^T \theta]^2 \leftarrow \text{That is } J(\theta) \end{aligned}$$

$J(\theta)$ in new notations

Compact expression

$$\begin{aligned} J(\theta) &= \|\mathbf{y} - \mathbf{X}\theta\|_2^2 \\ &= (\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - \mathbf{X}\theta) \quad f(x) = b^T x \propto \nabla f(x) = b \\ &= (\mathbf{y}^T - \{\mathbf{X}\theta\}^T)(\mathbf{y} - \mathbf{X}\theta) \\ &= (\mathbf{y}^T - \theta^T \mathbf{X}^T)(\mathbf{y} - \mathbf{X}\theta) \\ &= (\mathbf{y}^T)(\mathbf{y} - \mathbf{X}\theta) - \theta^T \mathbf{X}^T(\mathbf{y} - \mathbf{X}\theta) \\ &= \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\theta - \theta^T \mathbf{X}^T \mathbf{y} + \theta^T \mathbf{X}^T \mathbf{X}\theta \\ &= \text{constant} - 2\mathbf{y}^T \mathbf{X}\theta + \theta^T \mathbf{X}^T \mathbf{X}\theta \end{aligned}$$

$\nabla J(\theta)$

$$\mathbf{y}^T \mathbf{X}\theta = (\mathbf{X}\theta)^T \mathbf{y} \quad - 2\mathbf{y}^T \mathbf{X}\theta$$

Solution in matrix form

Compact expression

$$J(\theta) = \|X\theta - y\|_2^2 = \left\{ \theta^T X^T X \theta - 2(X^T y)^T \theta \right\} + \text{constant}$$

Gradients of Linear and Quadratic Functions

- $\nabla b^T x = b$
- $\nabla x^T A x = 2Ax$ (symmetric A)

$$f(x) = x^T A x$$

$x \in \mathbb{R}^D$ \downarrow $D \times D$ \downarrow $A x^2$

$\nabla f(x)$

Solution in matrix form

Compact expression

$$J(\theta) = \|X\theta - y\|_2^2 = \left\{ \theta^T X^T X \theta - 2(X^T y)^T \theta \right\} + \text{constant}$$

Gradients of Linear and Quadratic Functions

- $\nabla b^T x = b$
- $\nabla x^T A x = 2Ax$ (symmetric A)

Normal equations

$$\nabla J(\theta) = 2X^T X \theta - 2X^T y = 0$$

normal eqn $\rightarrow (X^T X) \theta = X^T y$

θ find minimizer
some & blur samples can

Solution in matrix form

Compact expression

$$J(\theta) = \|X\theta - y\|_2^2 = \left\{ \theta^T X^T X \theta - 2(X^T y)^T \theta \right\} + \text{constant}$$

Gradients of Linear and Quadratic Functions

- $\nabla b^T x = b$
- $\nabla x^T A x = 2Ax$ (symmetric A)

$$\begin{aligned} & (x^T x)^{-1} \quad \xrightarrow{\text{invertible}} \quad (x^T x)^{-1} \\ & (x^T x) \theta = x^T y \\ & \xrightarrow{\text{inverse of both sides}} \theta = (x^T x)^{-1} x^T y \end{aligned}$$

Normal equations

$$\nabla J(\theta) = 2X^T X \theta - 2X^T y = 0$$

This leads to the linear regression solution

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

Computational complexity

Bottleneck of computing the solution?

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

Matrix multiply of $X^T X \in \mathbb{R}^{(D+1) \times (D+1)}$

Inverting the matrix $X^T X$

How many operations do we need?

- $O(ND^2)$ for matrix multiplication
- $O(D^3)$ for matrix inversion
- Impractical for very large D or N

$$x_n = (x_1, x_n \theta)$$
$$\theta = (\theta_0, \dots, \theta_D)$$

$$\theta^T x_n$$

$$A \underset{D \times D}{\underset{\text{by } D}{\times}} B \underset{D \times D}{=} I$$
$$(\dots)$$

$$Ab_1 = z_1$$
$$Ab_2 = z_2$$
$$Ab_3 = I_2$$

Alternative method: an example of using numerical optimization

(Batch) Gradient descent

Algorithm 1 Gradient Descent (J)

1: $t \leftarrow 0$

2: Initialize $\theta^{(0)}$

3: **repeat**

4: $\nabla J(\theta^{(t)}) = \underline{\mathbf{X}^T \mathbf{X} \theta^{(t)}} - \underline{\mathbf{X}^T \mathbf{y}} = \sum_n (\mathbf{x}_n^T \theta^{(t)} - y_n) \mathbf{x}_n$

5: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla J(\theta^{(t)})$ *why would this work?*

6: $t \leftarrow t + 1$

7: **until** convergence

If gradient descent converges, it will converge to the same solution as using matrix inversion.

8: Return final value of θ

This is because $J(\theta)$ is a convex function in its parameters θ

What is the complexity of each iteration?

Stochastic gradient descent

Update parameters using one example at a time

Algorithm 2 Stochastic Gradient Descent (J)

- 1: $t \leftarrow 0$
 - 2: Initialize $\theta^{(0)}$
 - 3: **repeat**
 - 4: Randomly choose a training a sample x_t
 - 5: Compute its contribution to the gradient $g_t = (x_t^T \theta^{(t)} - y_t)x_t$
 - 6: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta g_t$
 - 7: $t \leftarrow t + 1$
 - 8: **until** convergence
 - 9: Return final value of θ
-

How does the complexity per iteration of stochastic gradient descent (SGD) compare with gradient descent (GD)?

- $O(ND)$ for GD versus $O(D)$ for SGD

Mini-summary

- Batch gradient descent computes the exact gradient.
- Stochastic gradient descent approximates the gradient with a single data point;
- Mini-batch variant: trade-off between accuracy of estimating gradient and computational cost
- Similar ideas extend to other ML optimization problems.
 - ▶ For large-scale problems, stochastic gradient descent often works well.

What if $X^T X$ is not invertible

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

$$(X^T X) \hat{\theta} = X^T y$$

Can you think of any reasons why that could happen?

Answer 1: N < D. Intuitively, not enough data to estimate all the parameters.

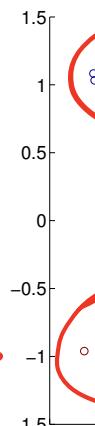
What if data is not linearly separable or fits to a line

Example of nonlinear classification

how do we deal w/ this problem

$x \in \mathbb{D}$

x_2



x_1

pos.
neg.

not linear
not separable

$x \in \mathbb{R}^2$

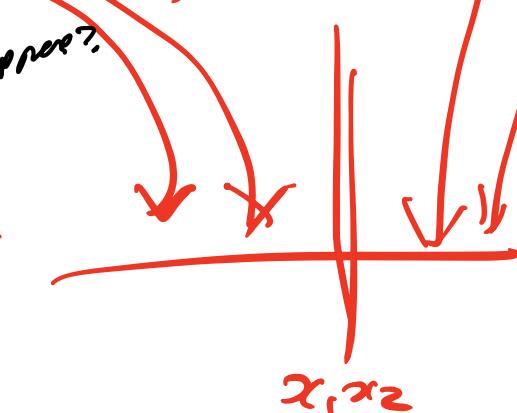
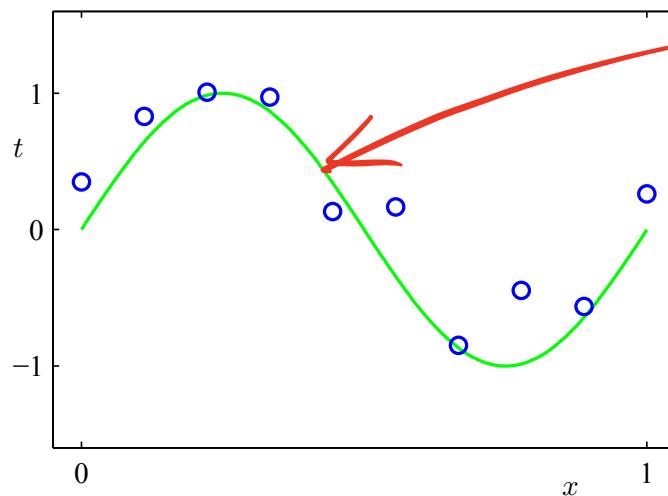
$$\phi(x) \leftarrow \text{transformation}$$

$$\phi(x) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ vector}$$

especially non-separable

what happens?

Example of nonlinear regression

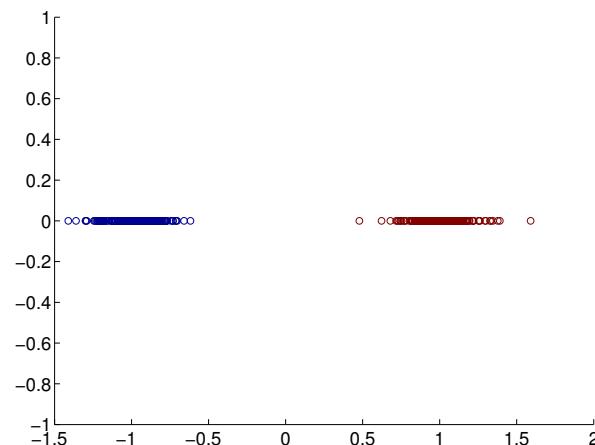
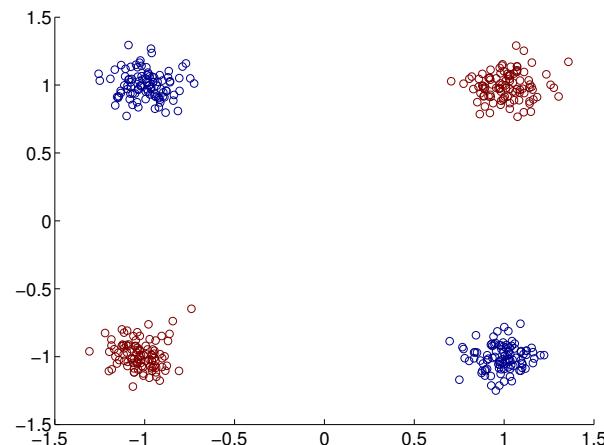


Nonlinear basis for classification

Transform the input/feature

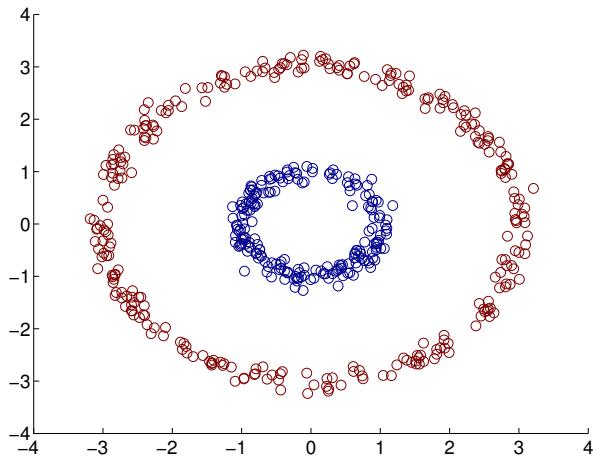
$$\phi(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^2 \xrightarrow{\text{transform pairs}} z = x_1 \cdot x_2$$

Transformed training data: linearly separable!



Another example

How to transform the input/feature?



*double - at
each point*

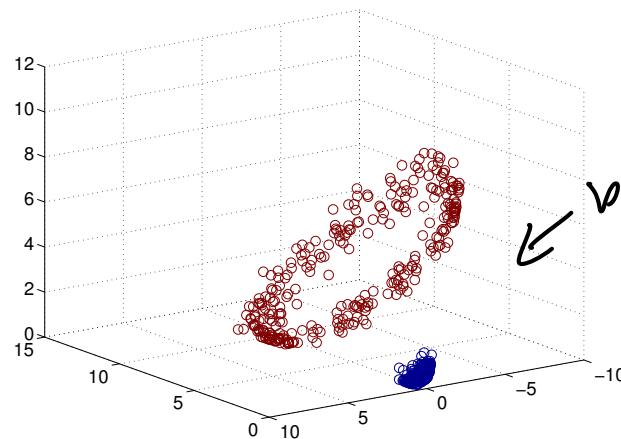
$$\phi(x) : x \in \mathbb{R}^2 \rightarrow z = \begin{bmatrix} x_1^2 \\ x_1 \cdot x_2 \\ x_2^2 \end{bmatrix} \in \mathbb{R}^3$$

2D \rightarrow 3D

$$\phi(x) = \frac{\|x\|_2}{2}$$

*vertical
axis for*

Transformed training data: linearly separable



becomes linearly separable

General nonlinear basis functions

We can use a **nonlinear mapping**

$$\phi(x) : x \in \mathbb{R}^D \rightarrow z \in \mathbb{R}^M$$

*feature vector
new set in
dimension*

where M is the dimensionality of the new feature/input z (or $\phi(x)$). Note that M could be either greater than D or less than or the same.

With the new features, we can apply our learning techniques to minimize our errors on the transformed training data

- linear methods: prediction is based on $\theta^T \phi(x)$
- other methods: nearest neighbors, decision trees, etc

Regression with nonlinear basis

Residual sum squares

$$\underset{\theta}{\operatorname{argmin}} \mathcal{J}(\theta) = \sum_n [\theta^T \phi(x_n) - y_n]^2$$

regression
dependent variables
parameters
label

regression

$$\theta^T x_n$$

$$\underset{\omega}{\operatorname{argmin}} \mathcal{J}(\omega) = \sum_n (\omega^T x_n - y_n)^2$$

where $\underline{\theta} \in \mathbb{R}^M$, the same dimensionality as the transformed features $\phi(x)$.

more difficult or less easier?

same price to pay?

how difficult is this prob?

convexity of cost func

Regression with nonlinear basis

analytically solve
data

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

Residual sum squares

$$\underset{\omega}{\operatorname{argmin}} J(\omega) = \sum_n (\underline{\omega^T x_n} - y_n)^2$$
$$\hat{\omega} = \underline{(x^T x)}^{-1} \underline{x^T y}$$

↑
design vector
design matrix

where $\theta \in \mathbb{R}^M$, the same dimensionality as the transformed features $\phi(x)$.

The linear regression solution can be formulated with the new design matrix

data pts \rightarrow design w/ sp
transformed

previous
design
matrix

$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$ $N \times D$

$\Phi = \begin{pmatrix} \Rightarrow \phi(x_1)^T \\ \Rightarrow \phi(x_2)^T \\ \vdots \\ \Rightarrow \phi(x_N)^T \end{pmatrix} \in \mathbb{R}^{N \times M}$

$M \ll D$

$\hat{\theta} = \underline{(\Phi^T \Phi)}^{-1} \underline{\Phi^T y}$

comple
solution

↑ more beyond linear models

Example with regression

Polynomial basis functions

example

$$\begin{array}{l} x \in \mathbb{R} \\ y \in \mathbb{R} \end{array}$$

$$\phi(x) = \begin{bmatrix} 1 & x^0 \\ x & x^1 \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \Rightarrow h(x) = \underline{\theta_0} + \sum_{m=1}^M \underline{\theta_m} x^m$$

1D input
one way to go beyond:
polynomial func
 $(M+1)$
how many degrees?

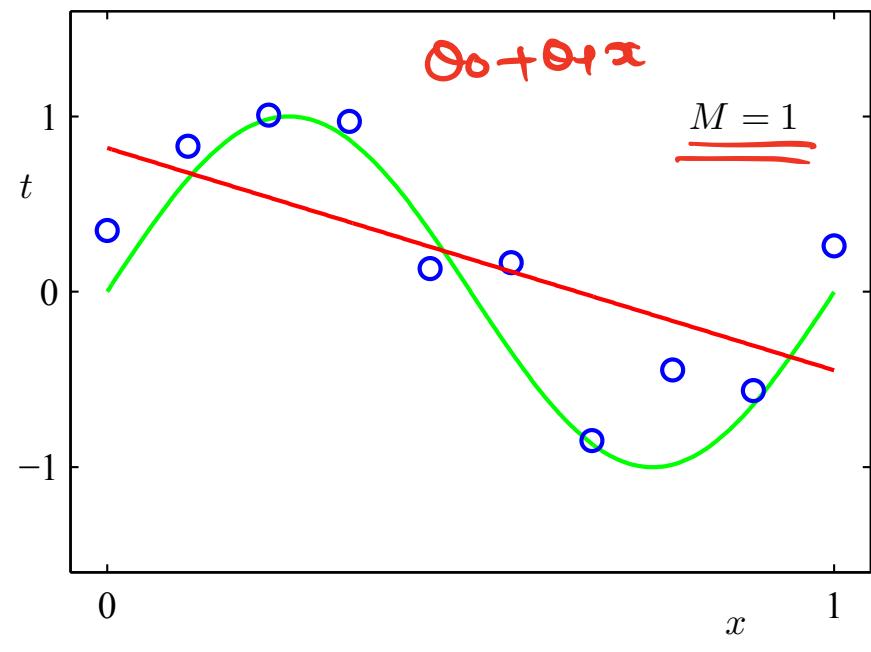
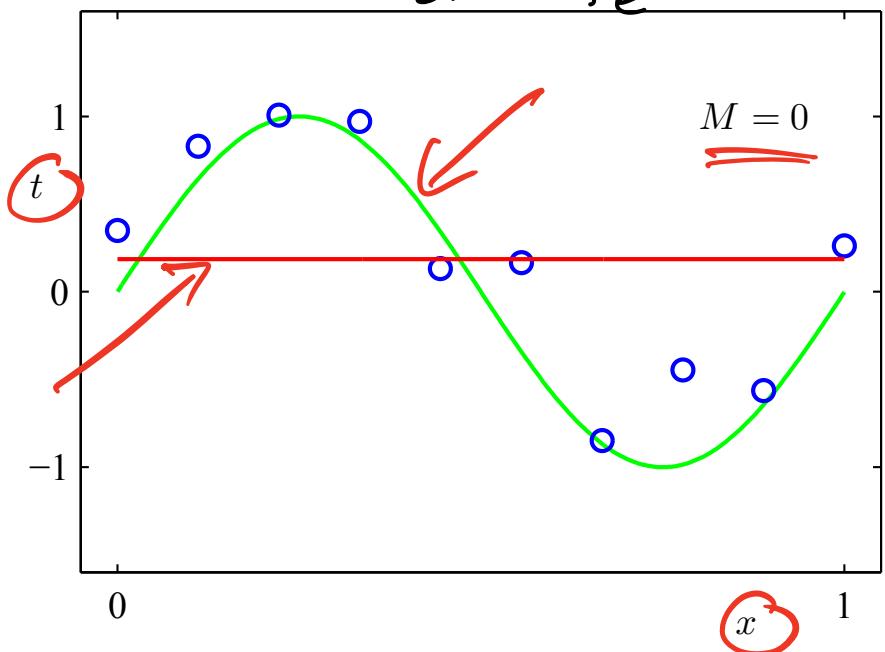
Example with regression

Polynomial basis functions *can do well*

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \Rightarrow h(x) = \theta_0 + \sum_{m=1}^M \theta_m x^m$$

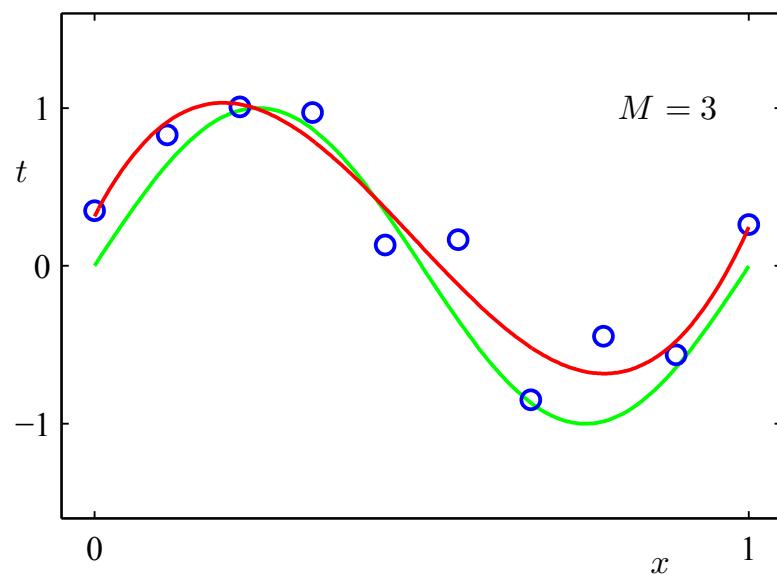
warning that can help

Fitting samples from a sine function: *underrfitting* as $h(x)$ is too simple
no fit stage

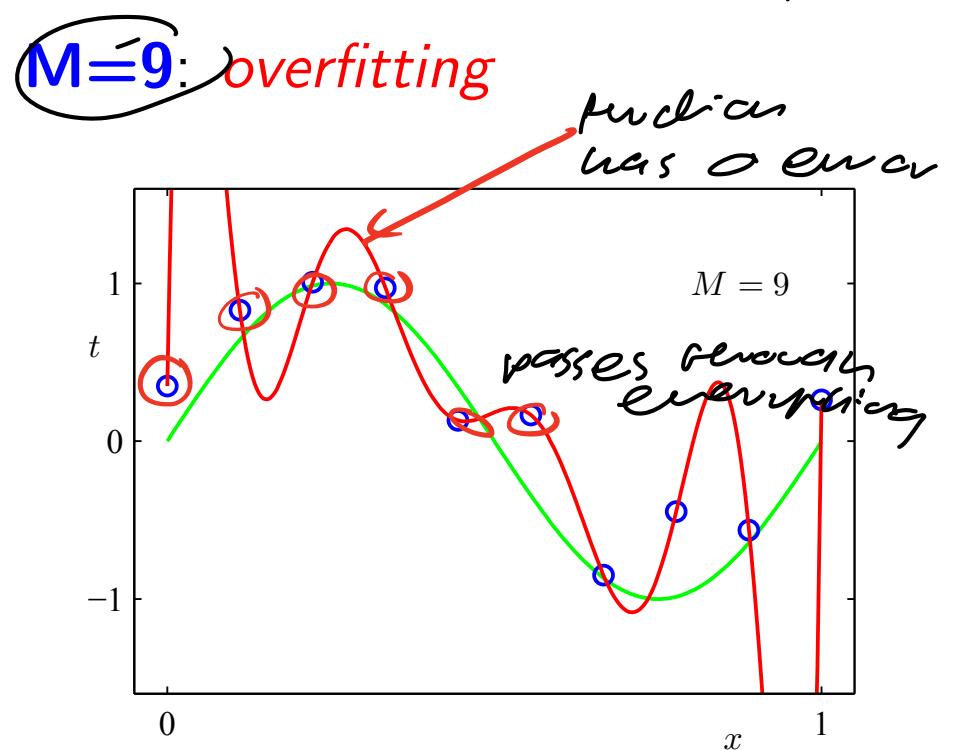


Adding high-order terms

$M=3$



more complex
→ worse on test data



function has a error
passes through every point

More complex features lead to better results on the training data, but potentially worse results on new data, e.g., test data!

Overfitting

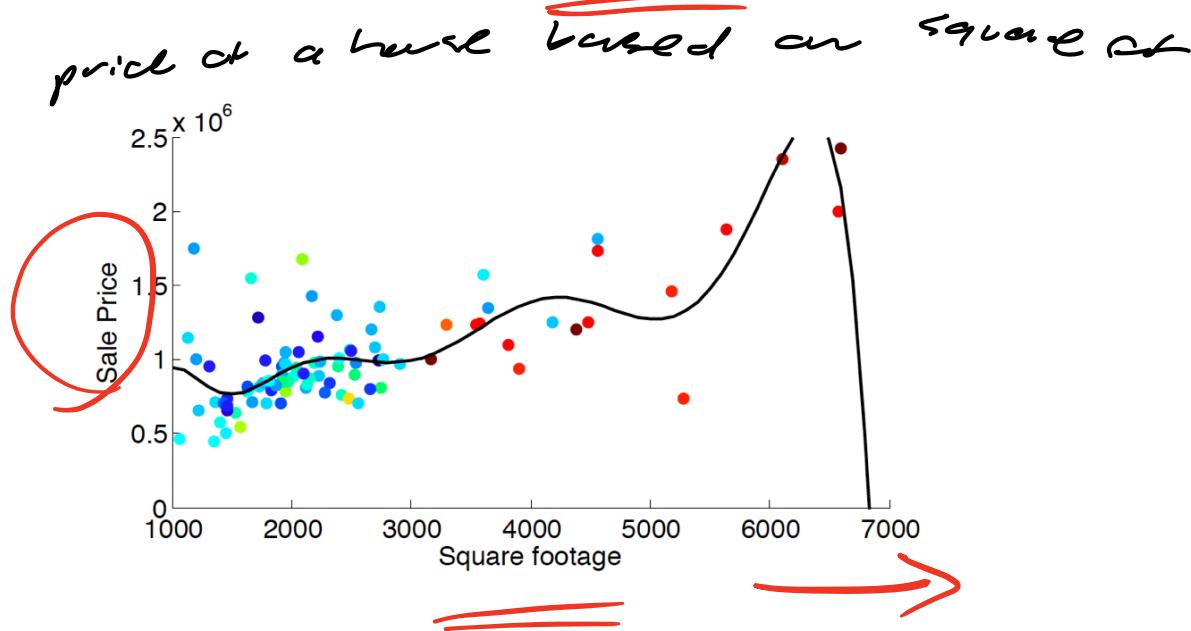
Parameters for higher-order polynomials are very large

*insred
parameters*

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
θ_0	0.19	0.82	0.31	0.35
θ_1		-1.27	7.99	232.37
θ_2			-25.43	-5321.83
θ_3			17.37	48568.31
θ_4				-231639.30
θ_5				640042.26
θ_6				-1061800.52
θ_7				1042400.18
θ_8				-557682.99
θ_9				125201.43

Overfitting can be quite disastrous

Fitting the housing price data with $M = 3$



Note that the price would goes to zero (or negative) if you buy bigger ones!
This is called poor generalization/overfitting.

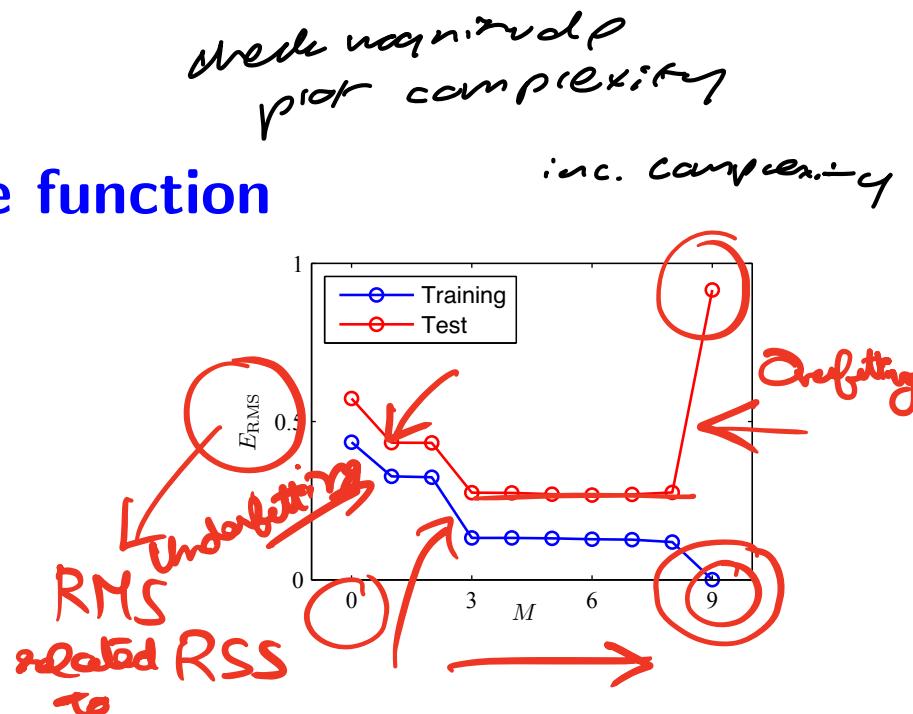
Detecting overfitting

Plot model complexity versus objective function

As a model increases in complexity, performance on training data keeps improving while performance on test data may first improve but eventually deteriorate.

- Horizontal axis: *measure of model complexity*; in this example complexity defined by order of the polynomial basis functions.

*test data
- dec.*

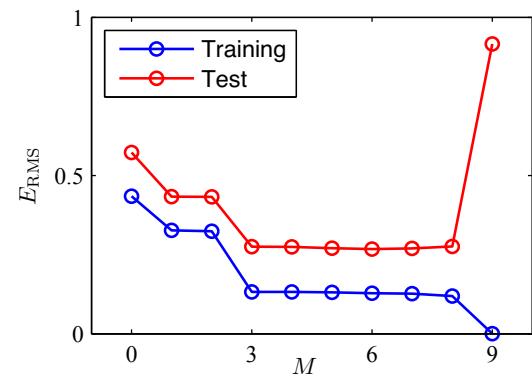


Detecting overfitting

Plot model complexity versus objective function

$RMS - \text{root mean } \sqrt{\cdot}$

As a model increases in complexity, performance on training data keeps improving while performance on test data may first improve but eventually deteriorate.



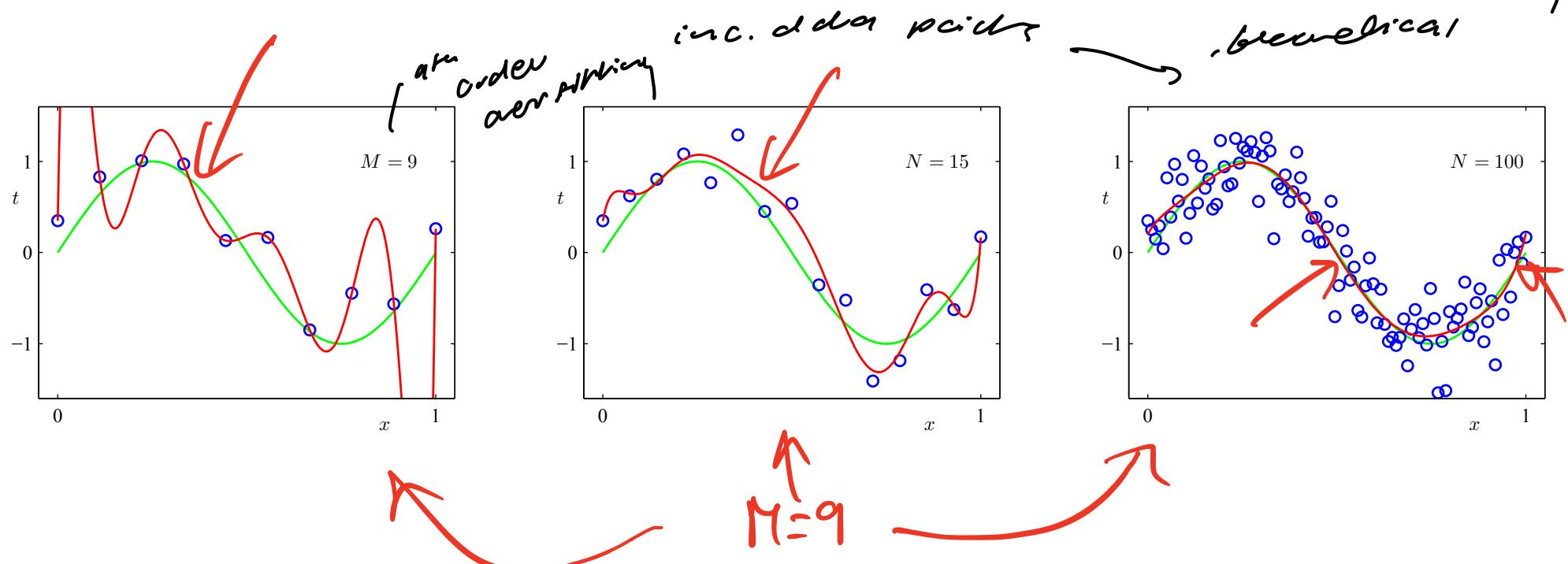
- Horizontal axis: *measure of model complexity*; in this example complexity defined by order of the polynomial basis functions.
- Vertical axis:
 - ① For regression, residual sum of squares or residual mean squared (squared root of RSS)
 - ② For classification, classification error rate.

overfitting

Use more training data to prevent over fitting

$$\hat{\theta}$$

The more, the merrier



Regularization methods

all set of methods

Intuition: For a linear model for regression

$$w = (w_1, 0, w_3, \dots, 0)$$

$$\underline{w^T x + b} \quad \text{model}$$

simple lin. reg?

we can try to identify 'simpler' models. But what does it mean for a model to be *simple*?

Simple:

How to define a simple model?

10

100

←
Simpler than a

Fewer parameters

Parameters take smaller values

$$|w| < 1$$

$$|w| \geq 10^6$$

4) Most features are ignored
←
= right entries
 $w=0$

3) Requires less computation

↓
fewer changes in diff airports

5)

Smoothness

6) Easier to interpret
model: simple w/ fewer params.

Regularization methods

Adding in extra - simple / complex

Intuition: For a linear model for regression

$$\underline{w^T x + b}$$

we can try to identify ‘simpler’ models. But what does it mean for a model to be *simple*?

Assumption (inductive bias)

A simpler model is one where most of the weights are zero.

A simpler model is one with smaller weights.

Smoothness
↑

Why are smaller weights associated with simpler models?



Simpler functions are smoother, i.e., nearby values of x have similar outputs \hat{y} .

Two values x and x' that differ in the first component by a small value ϵ .

Their predictions \hat{y} and \hat{y}' differ by ϵw_1 .

Smaller w_1 (closer to zero), more similar are the predictions.

in. comb.

$$\hat{y} = w_1 x_1 + w_2 \cancel{x_2} + \dots + \cancel{w_D x_D}$$

when is the prediction of the model

$$\hat{y}' = w_1 x'_1 + w_2 \cancel{x'_2} + \dots + \cancel{w_D x'_D}$$

differs a little bit

$$\hat{y} - \hat{y}' = w_1 (x'_1 - x_1) = w_1 \epsilon$$

weights - large

Regularized linear regression

A new cost function or error function to minimize

$$J(\underline{\underline{w}}, b) = \sum_n (y_n - \underline{\underline{w}}^T \underline{x}_n - b)^2 + \lambda \|\underline{\underline{w}}\|_2^2$$

residual sum of squares (RSS)

minimize
non-convex
 $\lambda > 0$
(new cost fun)

$$\lambda \|\underline{\underline{w}}\|_2^2 = (\underline{w}_1^2 + \underline{w}_2^2 + \dots + \underline{w}_D^2)$$

squared Euclidean norm

where $\lambda > 0$. This extra term $\|\underline{\underline{w}}\|_2^2$ is called regularization/regularizer and controls the model complexity.

$$\lambda \uparrow \infty$$

$$\hat{\underline{\underline{w}}} \rightarrow \underline{\underline{0}}$$

does not

$$\lambda \downarrow 0$$

$$\hat{\underline{\underline{w}}} = \hat{\underline{\underline{w}}}_{OLS}$$

ignoring some terms

exactly zero

can fit training to triviality of data

Regularized linear regression

A new cost function or error function to minimize

$$J(\mathbf{w}, b) = \sum_n (y_n - \mathbf{w}^T \mathbf{x}_n - b)^2 + \lambda \|\mathbf{w}\|_2^2$$

*minimizes a lot
compression*

where $\lambda > 0$. This extra term $\|\mathbf{w}\|_2^2$ is called regularization/regularizer and controls the model complexity.

Intuitions

- If $\lambda \rightarrow +\infty$, then

$$\hat{\mathbf{w}} \rightarrow \mathbf{0}$$

- If $\lambda \rightarrow 0$, then we trust our data more. Numerically,

$$\hat{\mathbf{w}} \rightarrow \arg \min \sum_n (\mathbf{w}^T \mathbf{x}_n + b - y_n)^2$$

Closed-form solution

For regularized linear regression: the solution changes very little (in form) from the OLS (Ordinary Least Squares) solution

$$\arg \min \sum_n (y_n - \mathbf{w}^T \mathbf{x}_n - b)^2 + \lambda \|\mathbf{w}\|_2^2 \Rightarrow \hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

and reduces to the OLS solution when $\lambda = 0$, as expected.

$$\nabla(\text{OLS}) + \nabla(\quad) = 0$$

* compare: inverse
design relax
haverspace

$$\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_{D \times D}$$
$$\Sigma = \begin{pmatrix} 1 & 0 & \dots \\ 0 & 1 & \dots \\ \vdots & \vdots & \ddots \end{pmatrix} \quad D \times D$$

Closed-form solution

For regularized linear regression: the solution changes very little (in form) from the OLS (Ordinary Least Squares) solution

$$\arg \min \sum_n (y_n - \underline{\underline{\mathbf{w}^T x_n}} - b)^2 + \lambda \|\mathbf{w}\|_2^2 \Rightarrow \hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

and reduces to the OLS solution when $\lambda = 0$, as expected.

If we have to use numerical procedure, the gradient would change nominally too,

$$\nabla J(\mathbf{w}) = 2(\mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{X}^T \mathbf{y} + \lambda \mathbf{w})$$

As long as $\lambda \geq 0$, the optimization is convex.
residual + \square **RSS**
quadratic at OLS

Optimization \hookrightarrow easy

Example: fitting data with polynomials

Our regression model

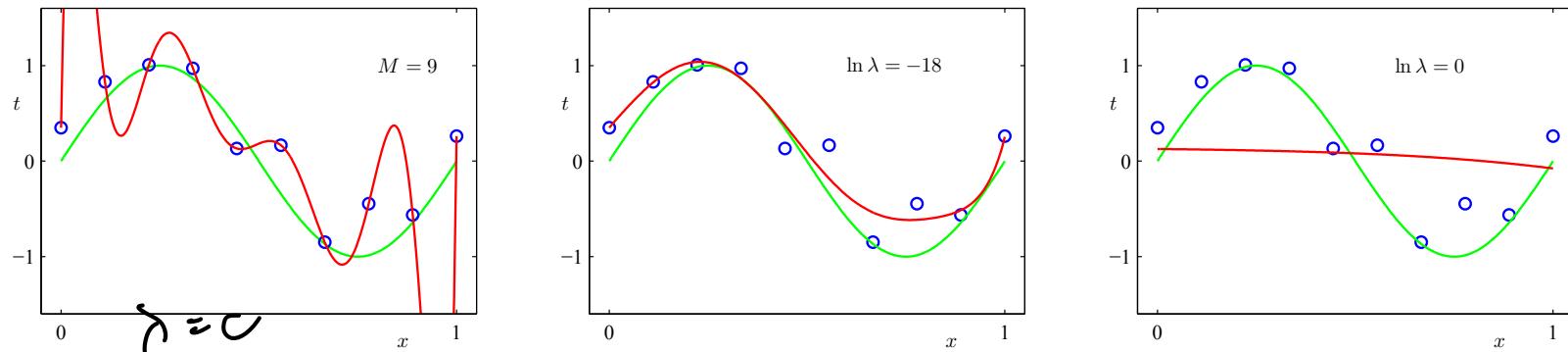
$$y = \sum_{m=1}^M w_m x^m$$

Regularization would discourage large parameter values as we saw with the OLS solution, thus potentially preventing overfitting.

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Overfitting in terms of λ

Overfitting is reduced from complex model to simpler one with the help of increasing regularizer



λ vs. residual error shows the difference of the model performance on training and testing dataset



The effect of λ

Large λ attenuates parameters towards 0 *when mapped to parameter values*

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
w_0	0.35	0.35	0.13
w_1	232.37	4.74	-0.05
w_2	-5321.83	-0.77	-0.06
w_3	48568.31	-31.97	-0.06
w_4	-231639.30	-3.89	-0.03
w_5	640042.26	55.28	-0.02
w_6	-1061800.52	41.32	-0.01
w_7	1042400.18	-45.95	-0.00
w_8	-557682.99	-91.53	0.00
w_9	125201.43	72.68	0.01

l_2 regularized logistic regression

general
→ can be applied
to any cost

Adding regularizer to the cost function for logistic regression

$$J(\mathbf{w}, b) = - \sum_n \{y_n \log h_{\mathbf{w}, b}(\mathbf{x}_n)$$
$$+ (1 - y_n) \log[1 - h_{\mathbf{w}, b}(\mathbf{x}_n)]\} + \lambda \|\mathbf{w}\|_2^2$$

—————
$$h_{\mathbf{w}, b}(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + b)$$

regularization

l_2 regularized logistic regression

Adding regularizer to the cost function for logistic regression

$$J(\mathbf{w}, b) = - \sum_n \{y_n \log h_{\mathbf{w}, b}(\mathbf{x}_n) + (1 - y_n) \log[1 - h_{\mathbf{w}, b}(\mathbf{x}_n)]\} + \underbrace{\lambda \|\mathbf{w}\|_2^2}_{\text{regularization}}$$

$$h_{\mathbf{w}, b}(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + b)$$

Numerical optimization

- Objective functions remains convex as long as $\lambda \geq 0$.
- Gradients and Hessians are changed marginally and can be easily derived.

How to choose the right amount of regularization?

Can we tune λ on the training dataset?

$$\min_{\lambda} \min_{(w,b)} \left[\text{RSS}(w, b) + \lambda \|w\|_2^2 \right]$$

training data

regularization

$$= \lambda \|w\|_2^2$$

$$\lambda = 0$$

How to choose the right amount of regularization?

λ - hyper parameter.

Can we tune λ on the training dataset?

No: as this will set λ to zero, i.e., without regularization, defeating our intention to use it to control model complexity and to gain better generalization.

λ is thus a **hyperparameter**. To tune it,

- We can use a development/validation dataset independent of training and testing dataset.

The procedure is similar to choosing K in the nearest neighbor classifiers.

How to choose the right amount of regularization?

carry λ , solve a new optimization prob

Can we tune λ on the training dataset?

No: as this will set λ to zero, i.e., without regularization, defeating our intention to use it to control model complexity and to gain better generalization.

λ is thus a hyperparameter. To tune it,

- We can use a development/validation dataset independent of training and testing dataset.

The procedure is similar to choosing K in the nearest neighbor classifiers.

For different λ , we get \hat{w} and evaluate the model on the development/validation dataset.
get a curve
now to you pick λ , measure or accuracy

We then plot the curve λ versus prediction error (accuracy, classification error) and find the place that the performance on the validation is the best.

Solution to linear regression

Compute the gradient of J and find the value of θ at which the gradient vanishes.

OLS cost fun
↓
 $\Rightarrow \nabla J(\theta) = 0$

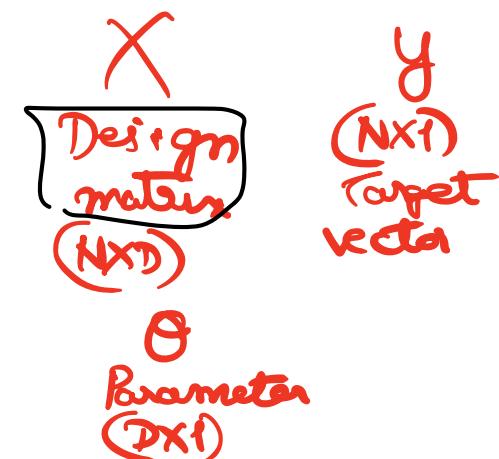
This leads us to solve the normal equations

$$Ax = b$$

$\Rightarrow X^T X \theta = X^T y$ *normal eqns.*

to obtain

$$\hat{\theta} = (X^T X)^{-1} X^T y \text{ soln.}$$



What if $X^T X$ is not invertible

sample / ~~scalars~~

use par. for
joining data

Answer 1: $N < D$. Intuitively, not enough data to estimate all the parameters.

new to deal w/ new lab. than source
can't be ind.

Answer 2: X columns are not linearly independent. Intuitively, there are two features that are perfectly correlated. In this case, solution is not unique.

new to
handle?

$$\hat{\theta} = \begin{cases} (X^T X)^{-1} X^T y & X^T X \text{ is invertible} \\ (X^T X + \lambda I)^{-1} X^T y & X^T X \text{ is not invertible} \end{cases}$$

l_2 regularized linear regression (ridge regression)

Solution

closed form solution

$$J(\theta) = \text{RSS}(\theta) + \lambda \|\theta\|_2^2$$

$$\hat{\theta} = (X^T X + \lambda I)^{-1} X^T y$$

OLS col. fit

This is equivalent to adding an extra term to $J(\theta)$

$$\overbrace{\|y - X\theta\|_2^2}^{\text{error}} + \underbrace{\lambda \sum_{d=1}^D \theta_d^2}_{\text{regularization}}$$

*2 benefits:
prevent overfitting*

Trade-off two quantities: minimize the error while keeping the weights small.

- $\sum_{d=1}^D \theta_d^2 = \sum_{d=1}^D w_d^2 = \|\mathbf{w}\|_2^2$ *col*
- l_2 -regularization uses the squared 2-norm

l_2 regularized linear regression (ridge regression)

Solution

$$\hat{\theta} = \underbrace{(X^T X + \lambda I)^{-1}}_{\text{red underline}} X^T y$$

This is equivalent to adding an extra term to $J(\theta)$

$$\overbrace{\|y - X\theta\|_2^2}^{J(\theta)} + \underbrace{\lambda \sum_{d=1}^D \theta_d^2}_{\text{regularization}}$$

$\log(\lambda)$

- $\sum_{d=1}^D \theta_d^2 = \sum_{d=1}^D w_d^2 = \|w\|_2^2$
- l_2 -regularization uses the squared 2-norm

Benefits

- Numerically more stable, invertible matrix
- Prevent overfitting

L_2 -regularization
 $\|\theta\|_2 = \sqrt{\sum_{d=1}^D \theta_d^2}$
leads to diff.
types of step

How to choose λ ?

λ is referred as *hyperparameter*

- In contrast θ is the parameter vector
- Use validation or cross-validation to find good choice of λ

Mini-summary

- l_2 regularized linear regression (ridge regression) can be helpful to avoid numerical instability. *stability* *new hyperparam. needs to be tuned*
- Only a minor modification to the OLS estimate
- New hyperparameter λ needs to be chosen using cross-validation.

Supervised learning

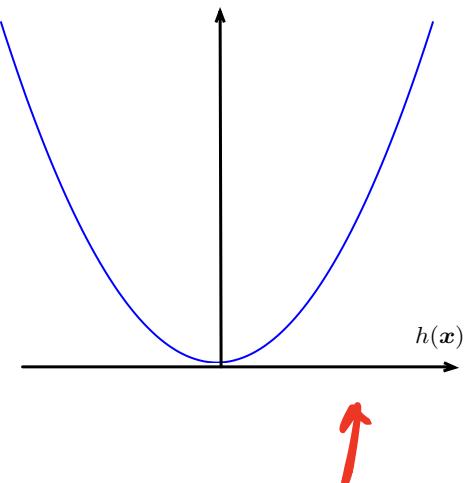
We aim to build a function $h(\underline{x})$ to predict the true value y associated with \underline{x} . If we make a mistake, we incur a **loss**

Loss Function $\rightarrow \ell(y, h(\underline{x}))$ *fun does not like loss*
true val. $=$ *prediction*

Example: squared loss function for regression when y is continuous

$$\ell(y, h(\underline{x})) = [h(\underline{x}) - y]^2$$

Ex: when $y = 0$



Other types of loss functions

For classification: 0/1 loss

$$\ell(y, h(x)) = \underbrace{\mathbf{1}\{y \neq h(x)\}}$$

$$\begin{array}{ll} y = h(x) & \ell(y, h(x)) = 0 \\ y \neq h(x) & \ell(y, h(x)) = 1 \end{array}$$

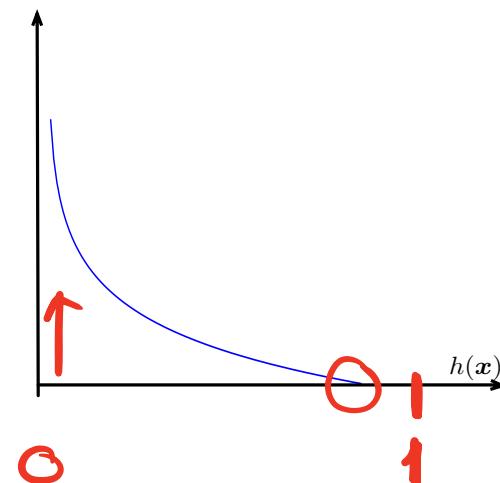
Other types of loss functions

For classification: *logistic* loss

$$\ell(y, h(\mathbf{x})) = -y \log h(\mathbf{x}) - (1-y) \log[1-h(\mathbf{x})]$$

Assumes \underline{h} outputs values in $[0, 1]$.

Ex: when $\underline{y} = 1$



Measure how good our hypothesis h is

Risk/ expected test loss: assume we know the true distribution of data $p(x, y)$, the *risk* is

how do we measure

$$\mathcal{R}[h(x)] = \sum_{x,y} \ell(h(x), y) p(x, y)$$

(inner loops)

$$\ell(\underline{h_1(x)}, \underline{y}) < \ell(h_2(\tilde{x}), \tilde{y})$$

high loss

$$\ell(h_1(\tilde{x}), \tilde{y}) > \ell(h_2(\tilde{x}), \tilde{y})$$

Measure how good our hypothesis h is

Risk/ expected test loss: assume we know the true distribution of data $p(x, y)$, the *risk* is

$$\mathcal{R}[h(x)] = \sum_{x,y} \ell(h(x), y) p(x, y)$$

is equal to this

given training dataset:

~~drop~~

However, we cannot compute $\mathcal{R}[h(x)]$, so we use *empirical risk/training error*, given a training dataset \mathcal{D}

$$\mathcal{R}^{\text{EMP}}[h(x)] = \frac{1}{N} \sum_n \ell(h(x_n), y_n)$$

look @ loss & average it

~~drop~~

Intuitively, as $N \rightarrow +\infty$, *as n becomes large*

$$\mathcal{R}^{\text{EMP}}[h(x)] \rightarrow \mathcal{R}[h(x)]$$

approx *true val*

Pick a good hypothesis h

A good hypothesis h is one that minizes risk/expected test loss $\mathcal{R}[h(\mathbf{x})]$ but we cannot even compute it!

Instead pick h that minimizes empirical risk/training error $\mathcal{R}^{\text{EMP}}[h(\mathbf{x})]$

This strategy is known as empirical risk minimization.

How this relates to what we have learned?

So far, we have been doing empirical risk minimization (ERM)

- For linear regression, $h_{w,b}(x) = w^T x + b$, and we use squared loss
- For logistic regression, $h_{w,b}(x) = \sigma(w^T x + b)$, and we use logistic loss
- Finding the best h is achieved by searching for (w, b) that minimizes the training error/empirical risk.

ERM might be problematic

- If $h(x)$ is complicated enough,

$$\mathcal{R}^{\text{EMP}}[h(x)] \rightarrow 0$$

empirical risk
→ 0
over generalization /
overfitting

- But then $h(x)$ is unlikely to do well in predicting things out of the training dataset \mathcal{D}
- This is called *poor generalization* or *overfitting*. We have just discussed approaches to address this issue.

Regularizer

Instead of $\underline{\underline{R^{\text{emp}}}}$, use

$$\begin{aligned} & \arg \min_{\mathbf{w}, b} \mathcal{R}^{\text{EMP}}[\underline{\underline{h_{\mathbf{w}, b}(x)}}] + \lambda \underline{\underline{R(\mathbf{w}, b)}} \\ &= \arg \min_{\mathbf{w}, b} \frac{1}{N} \sum_n \underline{\ell(y_n, \underline{\underline{h_{\mathbf{w}, b}(x_n)}})} + \lambda \underline{\underline{R(\mathbf{w}, b)}} \end{aligned} \quad (1)$$

denote empirical risk
involves encoder
regularizer

leads to diff learning vector

Loss functions ℓ , $\hat{y} = \mathbf{w}^T \mathbf{x} + b$

- Zero/One: $\ell(y, \hat{y}) = \mathbf{1}\{y \neq \hat{y}\} = \mathbf{1}\{y\hat{y} \leq 0\}$
- Squared: $\ell(y, \hat{y}) = [y - \hat{y}]^2$

Regularizer

Instead of \mathcal{R}^{emp} , use

$$\begin{aligned} & \arg \min_{\mathbf{w}, b} \mathcal{R}^{\text{EMP}}[h_{\mathbf{w}, b}(x)] + \lambda R(\mathbf{w}, b) \\ &= \arg \min_{\mathbf{w}, b} \frac{1}{N} \sum_n \ell(y_n, h_{\mathbf{w}, b}(\mathbf{x}_n)) + \lambda R(\mathbf{w}, b) \end{aligned} \quad (1)$$

Regularizer

\hookrightarrow recognizer

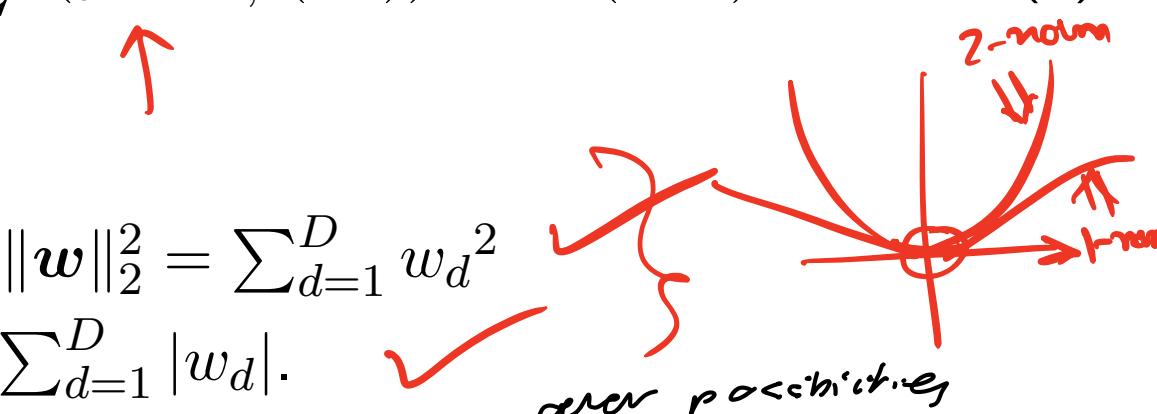
- Squared 2-norm: $R(\mathbf{w}, b) = \|\mathbf{w}\|_2^2 = \sum_{d=1}^D w_d^2$

- 1-norm: $R(\mathbf{w}, b) = \|\mathbf{w}\|_1 = \sum_{d=1}^D |w_d|$.

- 0-norm: $R(\mathbf{w}, b) = \sum_{d=1}^D \mathbf{1}\{w_d \neq 0\}$.

- p -norm: $R(\mathbf{w}, b) = \|\mathbf{w}\|_p = \left(\sum_{d=1}^D |w_d|^p \right)^{\frac{1}{p}}$

$\overbrace{\text{weights vector}}$
as small as poss.
 \hookrightarrow optimizing L2
 \hookrightarrow easier



when we choose
one over the others
to pick one over the others even
L2 norms, weights big, vs. L1 - linear
less sensitive to every large weights

Framework for supervised learning

area " framework

diff loss, save model

$$y \in \mathbb{R} \quad h_{w,b}(x) = w^T x + b$$

Squared loss

- Pick:

- Model/hypotheses ✓
- Loss function ✓ diff loss funcs
- Regularizer ✓
- Algorithm to solve optimization problem optimization probs.

$$\sum_n (y_n - h_{w,b}(x_n))$$

not all const. value
low to optimized

These choices lead to different learning algorithms

Framework for machine learning

With app to deal w/

- Application
 - ▶ Labeled vs unlabeled data
 - ▶ Labeled: supervised learning. Type of label: categorical (classification),
quantitative (regression)
class. prob
 - ▶ Unlabeled: unsupervised learning.
- Model/hypotheses
- Optimization problem
- Algorithm to solve optimization problem

Summary

- Hypotheses/models: linear functions of features.
- Objective: choose a function (*i.e.*, parameters for the function) that minimize a cost function.
 - ▶ Cost function measures loss or error of the predictions made by a model/hypothesis on training set.
 - ▶ Cost function depends on the learning problem.
- Probabilistic interpretation
 - ▶ Minimizing cost function equivalent to maximizing likelihood.
 - ▶ Allows us to understand assumptions and to generalize our models.
- How do we minimize the cost function ?
 - ▶ Numerical methods: gradient and stochastic gradient descent.
 - ▶ Sometimes analytical solutions are available.
 - ▶ Computational considerations influence the choice.
- Can allow non-linear models/hypotheses by transforming features using non-linear functions.

Summary

- Regularization: preferring a simpler model by penalizing large weights or many non-zero weights.
- Helpful to reduce overfitting and to prevent numerical instability.
- Can be added to any linear model.
- Only a minor modification to the unregularized algorithms.
- New hyperparameter λ needs to be chosen using cross-validation.

Inner products between features

Consider the inner products $\phi(x_m)^T \phi(x_n)$ for a pair of data points x_m and x_n .
 = *original vectors*
 = *feature vectors*

Polynomial-based nonlinear basis functions consider the following $\phi(x)$:

$$\phi: x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

This gives rise to an inner product in a special form,

$$\begin{aligned} \phi(x_m)^T \phi(x_n) &= x_{m1}^2 x_{n1}^2 + 2x_{m1}x_{m2}x_{n1}x_{n2} + x_{m2}^2 x_{n2}^2 \\ &= (x_{m1}x_{n1} + x_{m2}x_{n2})^2 = (\underline{x_m^T x_n})^2 \end{aligned}$$

without *feature map* *inner product* *of feature vectors*

The inner product can be computed by a function $(x_m^T x_n)^2$ defined in terms of the original features, *without computing* $\phi(\cdot)$.

a lot. more

Kernel functions

Some intuition

- To kernelize a machine learning algorithm, we need to be able to compute an inner product between a test point and *any* training point.
- Since we need this for any possible pair of points, we need a function that takes a pair of points and computes an inner product.
- This is the kernel function $\underline{k(\cdot, \cdot)}$.
- Given two inputs, $k(\cdot, \cdot)$ tells us how "similar" or "close" these inputs are in the space defined by the function ϕ .
measure of similarity
- 2 args

$$k(x_m, x_n) = \underbrace{(x_m^T x_n)}_{}^{\text{||}} = \Phi(x_m)^T \Phi(x_n)$$
$$x_m = \begin{pmatrix} x_{m1} \\ x_{m2} \end{pmatrix}$$
$$\Phi(x_m) = \begin{pmatrix} x_{m1} \\ x_{m2} \\ x_{m1}^2 \\ x_{m2}^2 \\ C_1 x_{m1} x_{m2} \\ C_2 x_{m1}^2 x_{m2}^2 \end{pmatrix}$$

Common kernel functions

Polynomial kernel function with degree of d

in pair of inputs

comparable

vector

$\mathbf{x}_m, \mathbf{x}_n \in \mathbb{R}^D$

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^\top \mathbf{x}_n + c)^d$$

for $\mathbf{x}_m, \mathbf{x}_n \in \mathbb{R}^D$, $c \geq 0$ and d is a positive integer.

$$k(\mathbf{x}_m, \mathbf{x}_n) = \underline{\underline{\Phi}}(\mathbf{x}_m)^\top \underline{\underline{\Phi}}(\mathbf{x}_n)$$

kernel:

$$c=0, d=2$$

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^\top \mathbf{x}_n)^2$$

inner prod.

$$\mathbf{x}_m, \mathbf{x}_n \in \mathbb{R}^D$$

even power
varied as c^d

$$c=0, d=3$$

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^\top \mathbf{x}_n)^3$$

inner prod.
out cubic

$$c > 0, d=3$$

$$\underline{\underline{\Phi}}(\mathbf{x}_m) = ?$$

analytic expression

$$\begin{aligned} & \text{it will be } \sqrt{x_{m1}} \\ & \sqrt{2} x_{m1} x_{m2} \\ & \sqrt{2} x_{m1} x_{m3} \\ & \vdots \\ & \sqrt{2} x_{m1} x_{mD} \\ & x_{m2}^2 \\ & \vdots \end{aligned}$$

$$\mathbf{x}_m = \begin{pmatrix} x_{m1} \\ x_{m2} \\ x_{m3} \\ \vdots \\ x_{mD} \end{pmatrix}$$

Common kernel functions

Polynomial kernel function with degree of d

$$k(x_m, x_n) = (\underline{x_m^T x_n} + c)^d$$

for $x_m, x_n \in \mathbb{R}^D$, $c \geq 0$ and d is a positive integer.

$$k(x_m, x_n) \stackrel{?}{=} \underline{\underline{\phi(x_n)}^T \phi(x_m)}$$

Gaussian kernel, RBF kernel, or Gaussian RBF kernel

Radial Basis Function

$$k(x_m, x_n) = e^{-\frac{\|x_m - x_n\|_2^2}{2\sigma^2}}$$

computes squared euclidean distance

$e^0 \uparrow 1$
- Positive number to

- Only depends on difference between two inputs
- Corresponds to a feature space with infinite dimensions (but we can work directly with the original features)!

-> or: a set
representations under
mappings

Common kernel functions

Polynomial kernel function with degree of d

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^T \mathbf{x}_n + c)^d$$

for $\mathbf{x}_m, \mathbf{x}_n \in \mathbb{R}^D$, $c \geq 0$ and d is a positive integer.

Gaussian kernel, RBF kernel, or Gaussian RBF kernel

$$k(\mathbf{x}_m, \mathbf{x}_n) = e^{-\|\mathbf{x}_m - \mathbf{x}_n\|_2^2 / 2\sigma^2}$$

- Only depends on difference between two inputs
- Corresponds to a feature space with *infinite* dimensions (but we can work directly with the original features)!

These kernels have hyperparameters to be tuned: d , c , σ^2

Kernel functions

Definition: a kernel function $k(\cdot, \cdot)$ is a bivariate function that satisfies the following properties. For any \mathbf{x}_m and \mathbf{x}_n ,

$$k(\mathbf{x}_m, \mathbf{x}_n) = k(\mathbf{x}_n, \mathbf{x}_m) \text{ and } k(\mathbf{x}_m, \mathbf{x}_n) = \underline{\underline{\phi}}(\mathbf{x}_m)^T \phi(\mathbf{x}_n)$$

for *some* function $\phi(\cdot)$.

Not very useful though

Examples we have seen

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^T \mathbf{x}_n)^2$$

$$k(\mathbf{x}_m, \mathbf{x}_n) = e^{-\|\mathbf{x}_m - \mathbf{x}_n\|_2^2 / 2\sigma^2}$$

Example that is not a kernel

$$k(\mathbf{x}_m, \mathbf{x}_n) = \|\mathbf{x}_m - \mathbf{x}_n\|_2^2$$

Conditions for being a positive semidefinite kernel function

Mercer theorem (loosely), a bivariate function $k(\cdot, \cdot)$ is a kernel function, if and only if, for *any N and any $\underline{x_1, x_2, \dots, x_N}$* , the matrix

$$K = \begin{pmatrix} k(\underline{x_1, x_1}) & k(\underline{x_1, x_2}) & \cdots & k(\underline{x_1, x_N}) \\ k(x_2, x_1) & k(\underline{x_2, x_2}) & \cdots & k(\underline{x_2, x_N}) \\ \vdots & \vdots & \vdots & \vdots \\ k(x_N, x_1) & k(x_N, x_2) & \cdots & k(x_N, x_N) \end{pmatrix}$$

is positive semidefinite.

$$K \succeq 0$$

$$\underline{z}^T K z \geq 0$$

all eigenvalues of $K \geq 0$

Why $\|x_m - x_n\|_2^2$ is not a positive semidefinite kernel?

Use the definition of positive semidefinite kernel function. We choose $\underline{N = 2}$, and compute the matrix

$$K = \begin{pmatrix} & \overset{(x_m, x_m)}{\circled{0}} & \|x_1 - x_2\|_2^2 \\ \overset{x_m}{\uparrow} & 0 & \underset{0}{\downarrow} \\ \overset{x_2}{\uparrow} & \|x_1 - x_2\|_2^2 & \end{pmatrix}$$

This matrix cannot be positive semidefinite as it has both *negative* and positive eigenvalues.

K is not positive semidefinite

$$k(x_m, x_n) = \|x_m - x_n\|_2^2$$

Recap: why use kernel functions? *- avoid building matrix*

Without specifying $\phi(\cdot)$, the kernel matrix

$$K = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_N) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_N) \\ \vdots & \vdots & \vdots & \vdots \\ k(x_N, x_1) & k(x_N, x_2) & \cdots & k(x_N, x_N) \end{pmatrix}$$

is exactly the same as

$$\begin{aligned} K &= \cancel{\Phi \Phi^T} \\ &= \begin{pmatrix} \cancel{\phi(x_1)^T \phi(x_1)} & \phi(x_1)^T \phi(x_2) & \cdots & \phi(x_1)^T \phi(x_N) \\ \phi(x_2)^T \phi(x_1) & \cancel{\phi(x_2)^T \phi(x_2)} & \cdots & \phi(x_2)^T \phi(x_N) \\ \cdots & \cdots & \cdots & \cdots \\ \phi(x_N)^T \phi(x_1) & \phi(x_N)^T \phi(x_2) & \cdots & \phi(x_N)^T \phi(x_N) \end{pmatrix} \end{aligned}$$

'Kernel trick'

Many learning methods depend on computing *inner products* between features — we have seen the example of nearest neighbors. For those methods, we can use a kernel function in the place of the inner products, i.e., “*kernelizing*” the methods, thus, introducing nonlinear features.

computing inner products?

There are infinite numbers of kernels to use!

Rules of composing kernels (this is just a partial list)

- if $\underline{k(x_m, x_n)}$ is a kernel, then $\underline{ck(x_m, x_n)}$ is also if $\underline{c > 0}$.
- if both $\underline{k_1(x_m, x_n)}$ and $\underline{k_2(x_m, x_n)}$ are kernels, then $\underline{\alpha k_1(x_m, x_n) + \beta k_2(x_m, x_n)}$ are also if $\alpha, \beta \geq 0$
- if both $\underline{k_1(x_m, x_n)}$ and $\underline{k_2(x_m, x_n)}$ are kernels, then $\underline{k_1(x_m, x_n)k_2(x_m, x_n)}$ are also.
- if $\underline{k(x_m, x_n)}$ is a kernel, then $\underline{e^{k(x_m, x_n)}}$ is also.
- ...

In practice, choosing an appropriate kernel is an “art”

People typically start with polynomial and Gaussian RBF kernels or incorporate domain knowledge.

Support vector machines

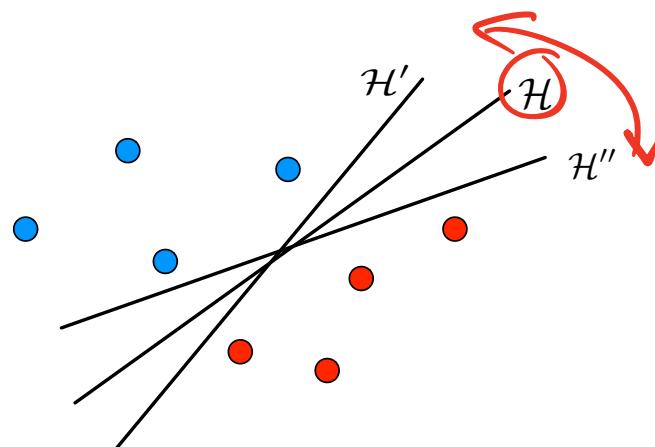
- One of the most commonly used classification algorithms
- Good generalization in theory and practice
- Can be kernelized

Setup

- Input: $\underline{\underline{x}} \in \mathbb{R}^D$
- Output: $\underline{\underline{y}} \in \{-1, +1\}$
- Hypotheses/Model: $h_{w,b} : \underline{\underline{x}} \rightarrow y$, with $h_{w,b}(\underline{\underline{x}}) = \text{SIGN}(b + \underline{\underline{w}}^T \phi(\underline{\underline{x}}))$
 $w = [w_1 \ w_2 \ \cdots \ w_M]^T$: *weights, parameters, or parameter vector*
weights & bias term
 b is called *bias*.
- Training data: $\mathcal{D} = \{(\underline{\underline{x}}_n, y_n), n = 1, 2, \dots, N\}$

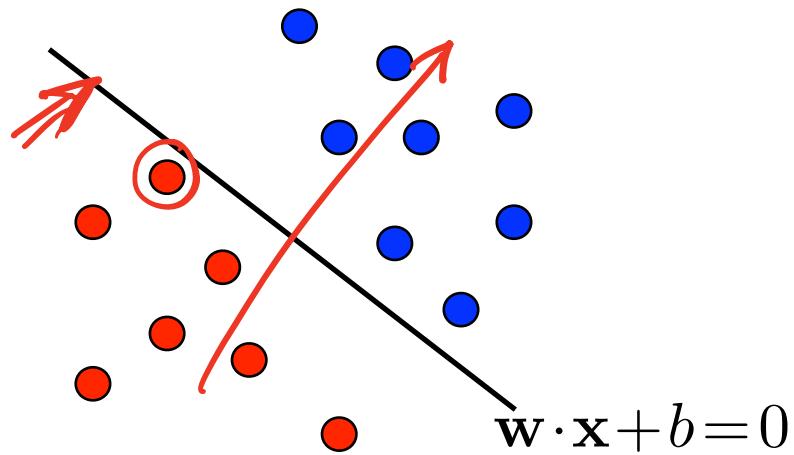
Intuition: where to put the decision boundary?

Consider the following *separable* training dataset, i.e., we assume there exists a decision boundary that separates the two classes perfectly. There are an *infinite* number of decision boundaries $\mathcal{H} : \mathbf{w}^T \phi(\mathbf{x}) + b = 0$!

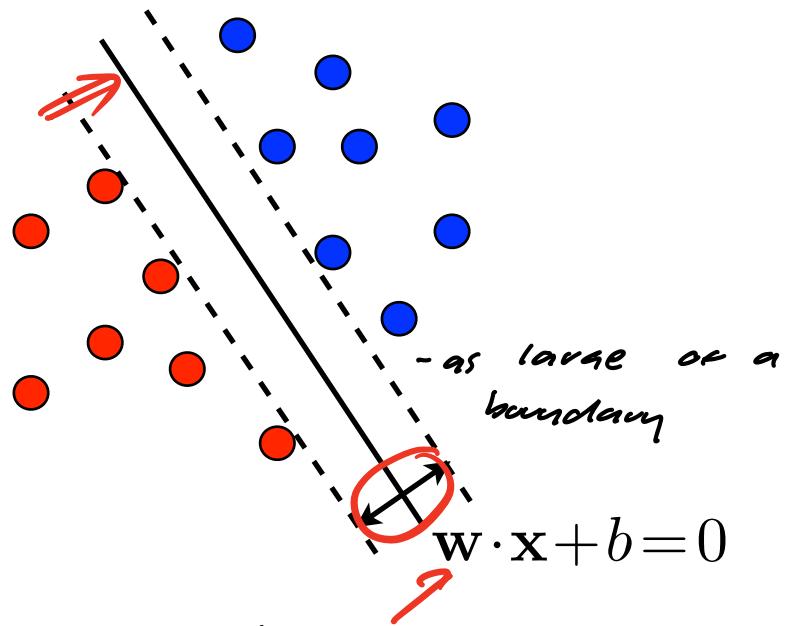


Which one should we pick?

Review of hyperplane



normal
padding
vector
'l'

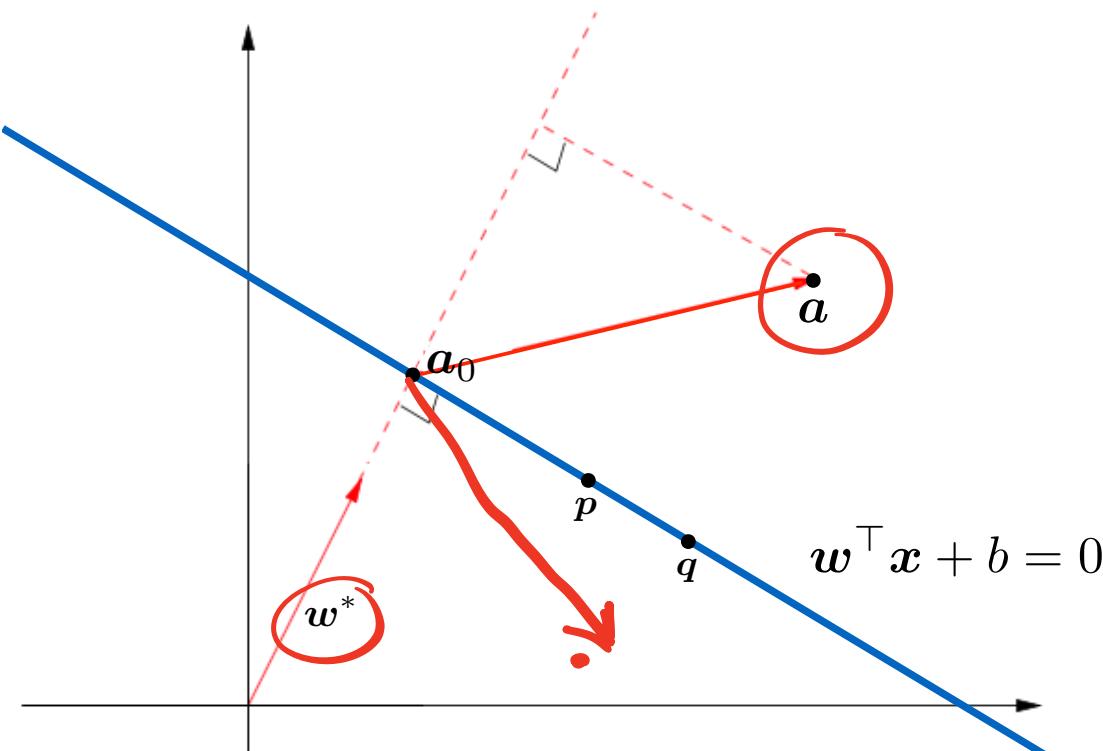


new far away

2 times
- a + - error rate
as hyper

- General equation is $w^\top x + b = 0$
- $w \in \mathbb{R}^d$ is a non-zero normal vector, b is a scalar intercept
- Divides the space in half, i.e., $w^\top x + b > 0$ and $w^\top x + b < 0$
- A hyperplane is a line in 2D and a plane in 3D

Properties of hyperplanes



How to compute signed distance from a to the hyperplane?

- If we define point a_0 on the hyperplane, then this distance corresponds to length of $a - a_0$ in direction of w^* , which equals $w^{*\top}(a - a_0) = \frac{w}{\|w\|}^\top(a - a_0) = \frac{w^\top a - w^\top a_0}{\|w\|}$
- Since $w^\top a_0 = -b$, the distance equals $\frac{1}{\|w\|}(w^\top a + b)$

Distance from a point to decision boundary

The *unsigned* distance from a point $\phi(x)$ to decision boundary (hyperplane) \mathcal{H} is

$$d_{\mathcal{H}}(\phi(x)) = \frac{|\mathbf{w}^T \phi(x) + b|}{\|\mathbf{w}\|_2}$$

We can remove the absolute value $|\cdot|$ by exploiting the fact that the decision boundary classifies every point in the training dataset correctly.

Namely, $(\mathbf{w}^T \phi(x) + b)$ and x 's label y must have the same sign, so:

$$\underline{d_{\mathcal{H}}(\phi(x))} = \frac{\underline{y}[\mathbf{w}^T \phi(x) + b]}{\|\mathbf{w}\|_2}$$

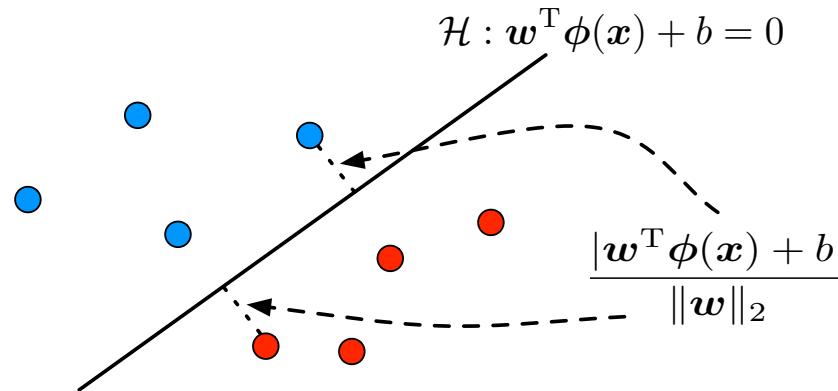
$$y \in \{+1, -1\} \quad \uparrow$$

Optimizing the Margin

Margin Smallest distance between the hyperplane and all training points

$$\max_{(w,b)}$$

$$\text{MARGIN}(w, b) = \min_n \frac{y_n [w^T \phi(x_n) + b]}{\|w\|_2}$$



How should we pick (w, b) based on its margin?

We want a decision boundary that is as far away from all training points as possible, so we want to *maximize* the margin!

$$\max_{w,b} \min_n \frac{y_n [w^T \phi(x_n) + b]}{\|w\|_2} = \max_{w,b} \frac{1}{\|w\|_2} \min_n y_n [w^T \phi(x_n) + b]$$

Rescaled Margin

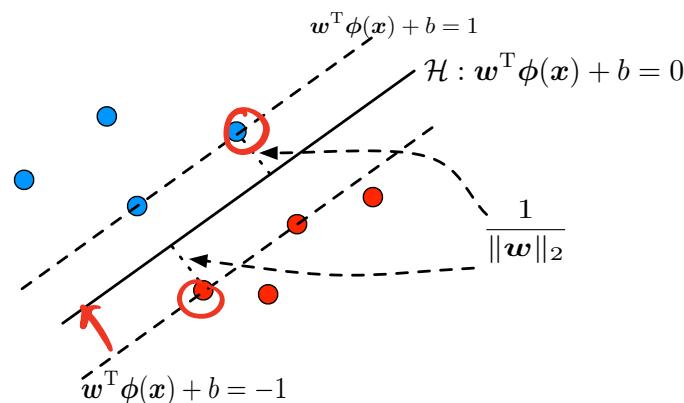
We can further constrain the problem by scaling (\mathbf{w}, b) such that

$$\min_n y_n [\mathbf{w}^T \phi(\mathbf{x}_n) + b] = 1$$

We've fixed the numerator in the $\text{MARGIN}(\mathbf{w}, b)$ equation, and we have:

$$\text{MARGIN}(\mathbf{w}, b) = \frac{1}{\|\mathbf{w}\|_2}$$

Hence the points closest to the decision boundary are at distance $\frac{1}{\|\mathbf{w}\|_2}$!



SVM: max margin formulation for separable data

Assuming separable training data, we thus want to solve:

$$\max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|_2} \quad \text{such that} \quad \min_n y_n [\mathbf{w}^T \phi(\mathbf{x}_n) + b] = 1$$

This is equivalent to

$$\max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|_2} \quad \text{such that} \quad y_n [\mathbf{w}^T \phi(\mathbf{x}_n) + b] \geq 1, \quad n \in \{1, \dots, N\}$$

$\min_{(\mathbf{w}, b)} \frac{1}{2} \|\mathbf{w}\|_2^2$

This is equivalent to

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|_2^2$$

s.t. $y_n [\mathbf{w}^T \phi(\mathbf{x}_n) + b] \geq 1, \quad n \in \{1, \dots, N\}$

Given our geometric intuition, SVM is called a *max margin* (or large margin) classifier. The constraints are called *large margin constraints*.

Convex optimization problem (also called convex program)

$$\begin{aligned} & \min_{\underline{\underline{x}}} \underline{\underline{f_0(x)}} \\ \text{s.t. } & \underline{\underline{f_i(x) \leq 0}}, \quad i = 1, \dots, \underline{\underline{m}} \\ & \underline{\underline{a_i^T x = b_i}}, \quad i = 1, \dots, \underline{\underline{p}} \end{aligned}$$

where $\underline{\underline{x}} \in \mathbb{R}^n$, f_0, \dots, f_m are convex functions.

- A point x that satisfies all constraints is **feasible**.
- If there exists at least one feasible point, the problem is **feasible**. Otherwise, it is **infeasible**.

SVM for separable data

$$\max_{(w,b)} \frac{1}{2} \|w\|_2^2$$
$$\min \underline{\underline{\frac{1}{2} \|w\|_2^2}}$$

Constrained optimization problem

$$\begin{aligned} & \min_{w,b} \frac{1}{2} \|w\|_2^2 \\ \text{s.t. } & -y_n [w^T \phi(x_n) + b] + 1 \leq 0, \quad n \in \{1, \dots, N\} \end{aligned}$$

$f_0(\mathbf{x})$

$\overbrace{\hspace{20em}}^{= f_i(\mathbf{x})}$

- Objective function: convex in (w, b)
- Inequality constraints are linear in (w, b) . Hence, convex.

SVM for non-separable data

SVM formulation for separable data

$$\min_{\mathbf{w}, b} \quad \frac{1}{2} \|\mathbf{w}\|_2^2$$

$$\text{s.t. } y_n [\mathbf{w}^T \phi(\mathbf{x}_n) + b] \geq 1, \quad n \in \{1, \dots, N\}$$

Non-separable setting In practice our training data will not be separable. What issues arise with the optimization problem above when data is not separable?

- For every \mathbf{w} there exists a training point \mathbf{x}_i such that

$$y_i [\mathbf{w}^T \phi(\mathbf{x}_i) + b] \leq 0$$

- There is no feasible (\mathbf{w}, b) as at least one of our constraints is violated!

SVM for non-separable data

Constraints in separable setting

$$y_n [\mathbf{w}^T \phi(\mathbf{x}_n) + b] \geq 1, \quad n \in \{1, \dots, N\}$$

Constraints in non-separable setting

Idea: modify our constraints to account for non-separability! Specifically, we introduce **slack variables** $\xi_n \geq 0$:

$$y_n [\mathbf{w}^T \phi(\mathbf{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1, \dots, N\}$$

- For “hard” training points, we can increase ξ_n until the above inequalities are met
- What does it mean when ξ_n is very large?

Soft-margin SVM formulation

We do not want ξ_n to grow too large, and we can control their size by incorporating them into our optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}, b, \xi} \quad & \frac{1}{2} \|\boldsymbol{w}\|_2^2 + C \sum_n \xi_n \\ \text{s.t.} \quad & y_n [\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1, \dots, N\} \\ & \xi_n \geq 0, \quad n \in \{1, \dots, N\} \end{aligned}$$

What is the role of C ?

- User-defined hyperparameter
- Trades off between the two terms in our objective
- Same idea as the regularization term in ridge regression, i.e., $C = \frac{1}{\lambda}$

How to solve this problem?

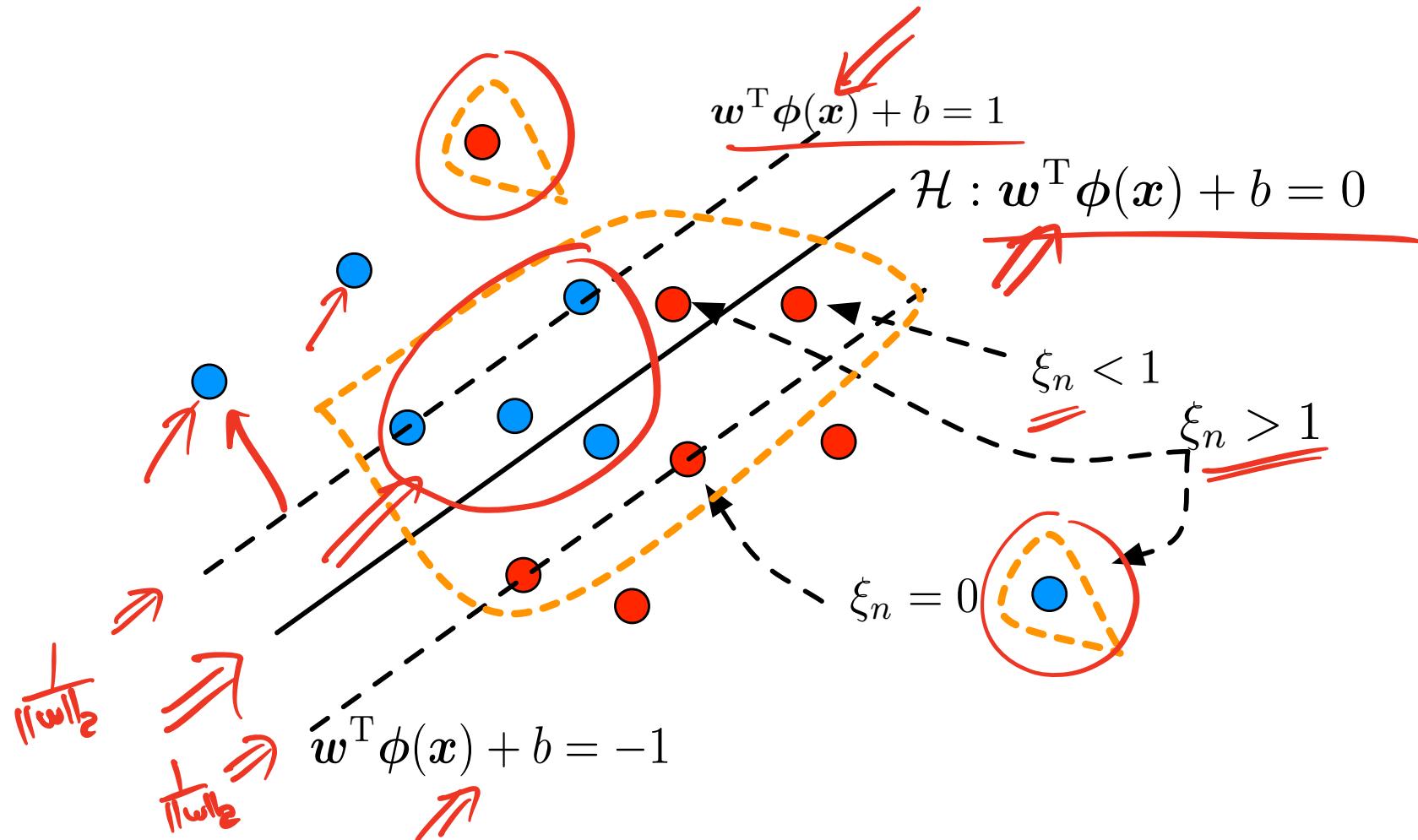
$$\begin{aligned} \min_{\boldsymbol{w}, b, \xi} \quad & \frac{1}{2} \|\boldsymbol{w}\|_2^2 + C \sum_n \xi_n \\ \text{s.t.} \quad & y_n [\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1, \dots, N\} \\ & \xi_n \geq 0, \quad n \in \{1, \dots, N\} \end{aligned}$$

- This is a *convex problem*: the objective function is quadratic in \boldsymbol{w} and linear in ξ and the constraints are linear (inequality) constraints in \boldsymbol{w} , b and ξ_n .
- Given $\phi(\cdot)$, we can solve the optimization problem using general-purpose solvers.
- There are several specialized methods for solving this problem, taking advantage of the special structure of the objective function and the constraints (we will not discuss them). Most existing SVM implementation/packages leverage these methods.

Meaning of “support vectors” in SVMs

- The SVM solution is only determined by a subset of the training instances.
- These instances are called *support vectors*
- All other training points do not affect the optimal solution, i.e., if remove the other points and construct another SVM classifier on the reduced dataset, the optimal solution will be the same

Visualization of how training data points are categorized



Support vectors are highlighted by the dotted orange lines

Support Vector Machine

- A linear classifier (hyperplane) that maximizes the margin.
- An alternative view of SVMs.

A general view of supervised learning

Definition Assume $y \in \{-1, 1\}$ and the decision rule is

$$h(\mathbf{x}) = \text{SIGN}(a(\mathbf{x})) \text{ with } a(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

linear
- class
models

For classification: 0/1 loss

sign
of activation function

$$\ell^{0/1}(y, a(\mathbf{x})) = \begin{cases} 0 & \text{if } ya(\mathbf{x}) \geq 0 \\ 1 & \text{otherwise} \end{cases}$$

Minimize weighted sum of empirical risk and regularizer

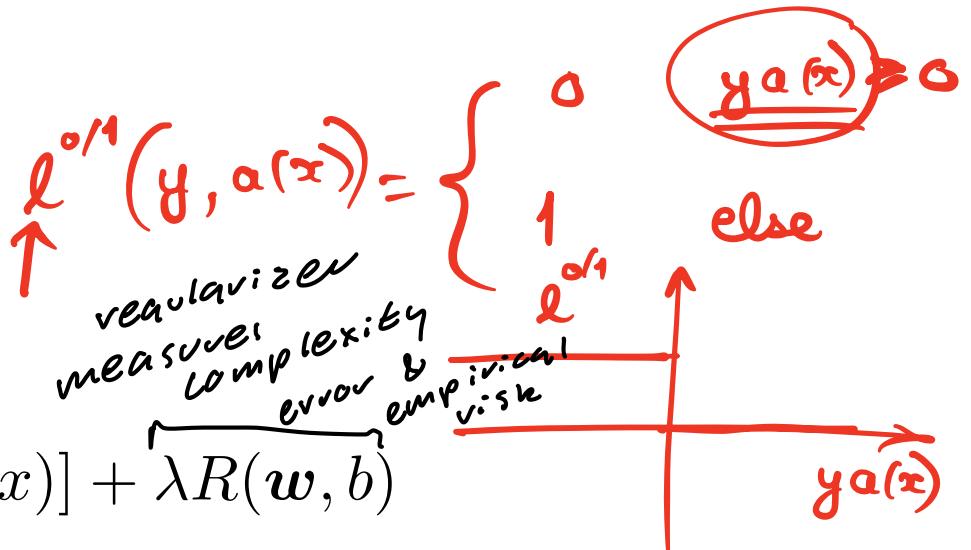
loss function

- find parameters
then minimize

empirical risk

$$\arg \min_{w,b} R^{\text{EMP}}[h_{w,b}(x)] + \underbrace{\lambda R(w,b)}_{\text{regularizer}}$$

$$= \arg \min_{w,b} \frac{1}{N} \sum_n \ell(y_n, a(x_n)) + \lambda R(w, b) \quad (1)$$



- Problem with minimizing the 0/1 loss ?

- overly complex

Hinge loss

Definition Assume $y \in \{-1, 1\}$ and the decision rule is $h(\mathbf{x}) = \text{SIGN}(a(\mathbf{x}))$ with $a(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$,

$$\ell^{\text{HINGE}}(y, a(\mathbf{x})) = \begin{cases} 0 & \text{if } ya(\mathbf{x}) \geq 1 \\ 1 - ya(\mathbf{x}) & \text{otherwise} \end{cases}$$

Intuition

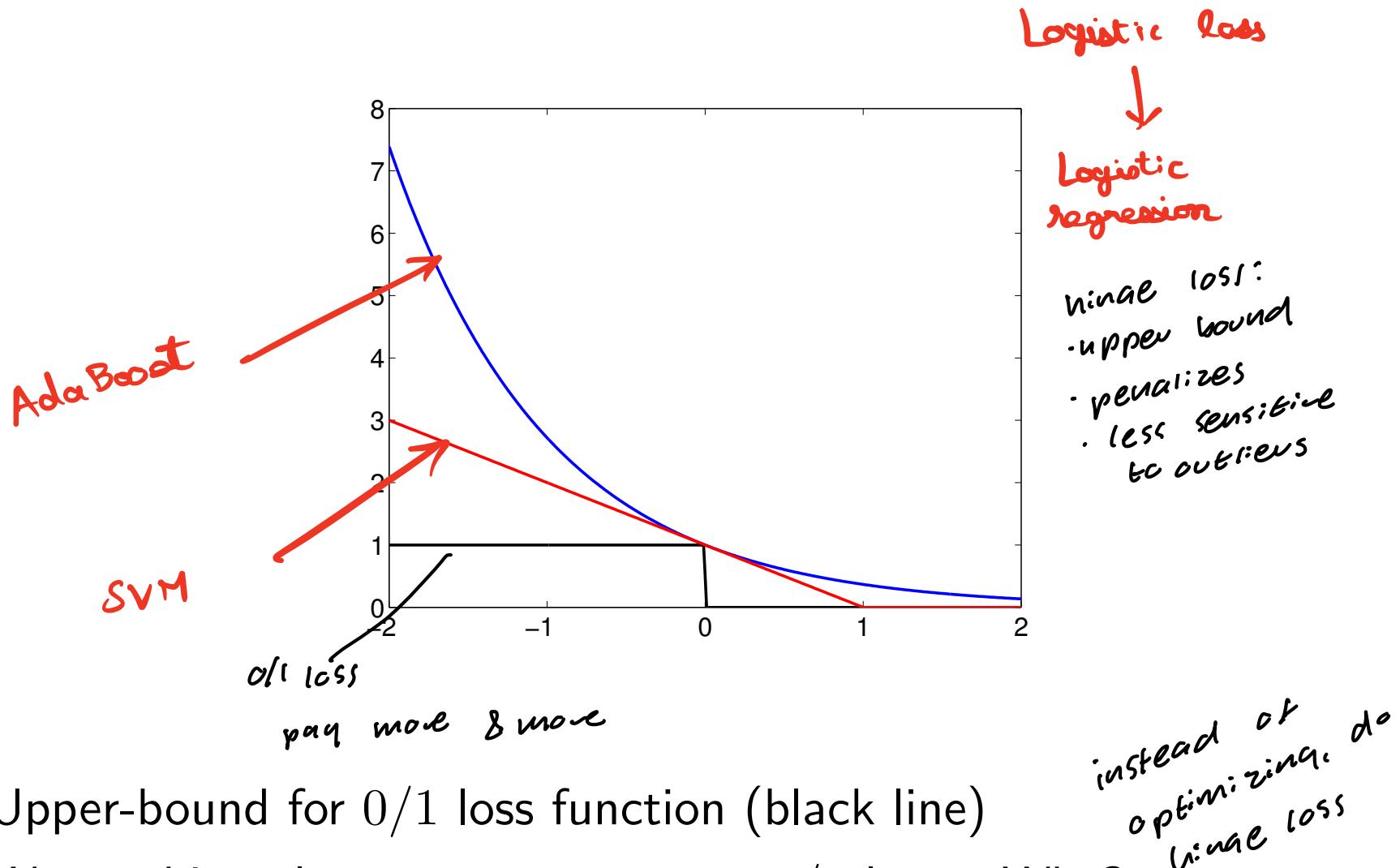
- No penalty if raw output, $a(\mathbf{x})$, has same sign and is far enough from decision boundary (i.e., if ‘margin’ is large enough)
- Otherwise pay a growing penalty, between 0 and 1 if signs match, and greater than one otherwise

Convenient shorthand

$$\ell^{\text{HINGE}}(y, a(\mathbf{x})) = \max(0, 1 - ya(\mathbf{x})) = \underline{(1 - ya(\mathbf{x}))}_+ \quad \text{confidence}$$

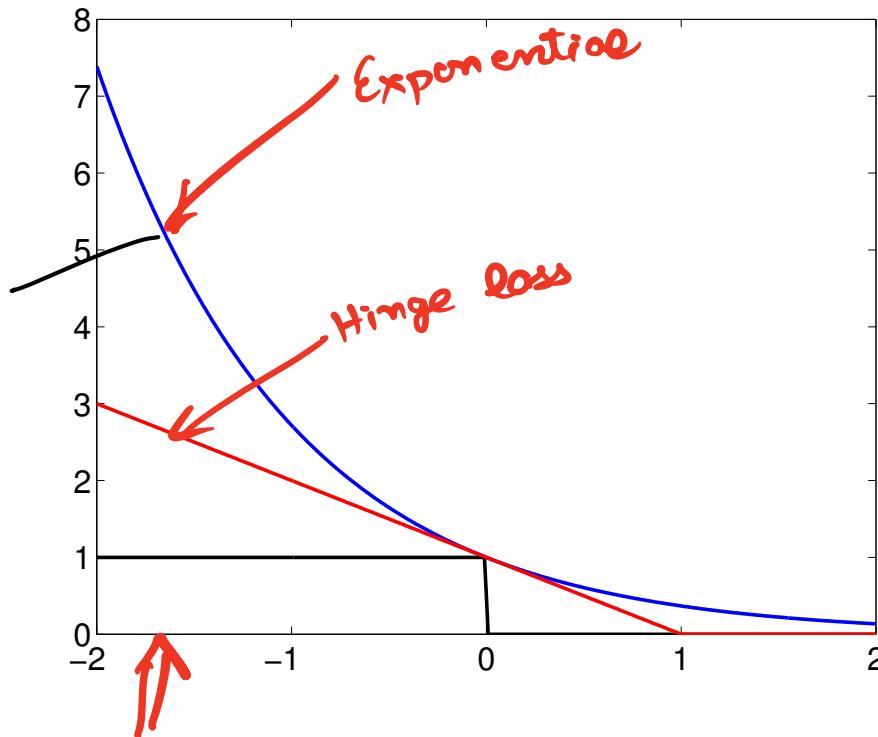
no penalty/loss
label: “true label”
loss ↑ [0, 1]
acc above if
signal doesn't
match

Visualization and Properties



- Upper-bound for 0/1 loss function (black line)
- We use hinge loss as a *surrogate* to 0/1 loss – Why?
- Hinge loss is convex, and thus easier to work with.

Visualization and Properties



- Other **surrogate losses** can be used, e.g., exponential loss for Adaboost (in blue), logistic loss (not shown) for logistic regression
- Hinge loss less sensitive to outliers than exponential (or logistic) loss

Primal formulation of support vector machines (SVM)

Minimizing the total hinge loss on all the training data with l_2 regularization

diff optimization

$$\min_{w,b} \sum_n \max(0, 1 - y_n [w^T \phi(x_n) + b]) + \frac{\lambda}{2} \|w\|_2^2$$

model complexity

$\ell(x) = [w^T \phi(x) + b]$

minimizing hinge loss

Analogous to l_2 regularized least squares, as we balance between two terms (the loss and the regularizer).

Previously, we used geometric arguments to derive:

Margin Slack variables

$$\min_{w,b,\xi} \frac{1}{2} \|w\|_2^2 + C \sum_n \xi_n$$

s.t. $y_n [w^T \phi(x_n) + b] \geq 1 - \xi_n$ and $\xi_n \geq 0, n \in \{1, \dots, N\}$

applying loss function

Do these yield the same solution?

sum minimize

same answer

prediction: support vector machine

Recovering our previous SVM formulation

Minimizing the total hinge loss on all the training data

$$\min_{\mathbf{w}, b} \sum_n \max(0, 1 - y_n[\mathbf{w}^T \phi(\mathbf{x}_n) + b]) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

Define $C = 1/\lambda$:

$$\min_{\mathbf{w}, b} C \sum_n \max(0, 1 - y_n[\mathbf{w}^T \phi(\mathbf{x}_n) + b]) + \frac{1}{2} \|\mathbf{w}\|_2^2$$

*I break all terms in
summarize.*

Define $\xi_n = \max(0, 1 - y_n[\mathbf{w}^T \phi(\mathbf{x}_n) + b])$

$$\min_{\mathbf{w}, b, \xi} C \sum_n \xi_n + \frac{1}{2} \|\mathbf{w}\|_2^2$$

$$\text{s.t. } \max(0, 1 - y_n[\mathbf{w}^T \phi(\mathbf{x}_n) + b]) = \xi_n, \quad n \in \{1, \dots, N\}$$

Recovering our previous SVM formulation

- similar
- objective function
- check regularizer
- same the same : C you see constraints

• 1 constraint

Minimizing hinge loss

$$\min_{w, b, \xi} C \sum_n \xi_n + \frac{1}{2} \|w\|_2^2$$

s.t.

$$\max(0, 1 - y_n [w^T \phi(x_n) + b]) = \xi_n \quad n \in \{1, \dots, N\}$$

is equivalent to

SVM

$$\min_{w, b, \xi} [C \sum_n \xi_n + \frac{1}{2} \|w\|_2^2]$$

s.t.

$$1 - y_n [w^T \phi(x_n) + b] \leq \xi_n, \quad n \in \{1, \dots, N\}$$

$$0 \leq \xi_n, \quad n \in \{1, \dots, N\}$$

any solution to hinge is solution to SVM

$$\xi_n = \max(0, 1 - y_n (w^T \phi(x_n) + b))$$

$$\xi_n^* = \max(0, 1 - y_n (w^* \phi(x_n) + b^*))$$

slack variables (> 0)

At optimal solution constraints are active so we have equality! Why?

- If $\xi_n^* > \max(0, 1 - y_n f(x_n))$, we could choose $\bar{\xi}_n < \xi_n^*$ and still satisfy the constraint while reducing our objective function!
- Since $c \geq \max(a, b) \iff c \geq a, c \geq b$, we recover previous formulation

new value for slack variable
new solution to satisfy constraint
replace optional start

SVM: hyperplane with largest margin
s.t. minimizes hinge loss + ℓ_2 -regularization

Outline

1 Review of last lecture

2 SVM – Hinge loss

3 Kernelized SVMs

*can be much easier
we can get convex algs*

4 Evaluating ML algorithms

Goal: Kernelize SVMs

$$\underline{\phi(x_n)}$$

completely avoid
compute optimization number

- Rewrite the SVM optimization problem so that it no longer depends on $\phi(x_n)$ but instead depends only on inner products $\underline{\phi(x_n)^T \phi(x_m)}$.
- If we can do this, we can then replace $\underline{\phi(x_n)^T \phi(x_m)}$ with a kernel function $\underline{k(x_n, x_m)}$.

Primal formulation of support vector machines (SVM)

discipline for
variables
regularization

$$\min_{w,b,\xi} \frac{1}{2} \|w\|_2^2 + C \sum_n \xi_n$$

s.t. $y_n [w^T \phi(x_n) + b] \geq 1 - \xi_n$ and $\xi_n \geq 0, n \in \{1, \dots, N\}$

This can be converted to its *dual form*.

Dual formulation of SVM

Let $\underline{\underline{w}_*}$ be the minimizer of the SVM problem for training data: $\{(x_n, y_n)\}$.

Then $w_* = \sum_n \alpha_n y_n \phi(x_n)$ for a new set of dual variables: $\alpha_n \geq 0$.

Primal
Dual

Dual formulation of SVM

$y_m \in \{+1, -1\}$
reduce vectors

Dual is also a convex program

$$\begin{aligned} & \max_{\alpha} \left(\sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(x_m)^T \phi(x_n) \right) \\ \text{s.t. } & 0 \leq \alpha_n \leq C, \quad n \in 1, \dots, N \\ & \sum_n \alpha_n y_n = 0 \end{aligned}$$

labels w/ each training vector

$\alpha = (\alpha_1, \dots, \alpha_N)$

There are N dual variable $\underline{\alpha_n}$, one for each constraint in the primal formulation

Kernel SVM

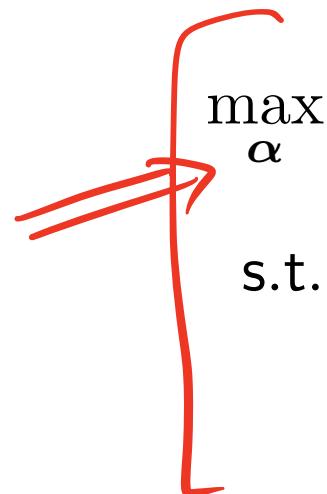
We replace the inner products $\phi(x_m)^T \phi(x_n)$ with a kernel function

$$\max_{\alpha} \quad \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n k(x_m, x_n)$$

s.t. $0 \leq \alpha_n \leq C, \quad n \in 1, \dots, N$

$$\sum_n \alpha_n y_n = 0$$

replicates of kernels
→ new linear kernels



We can define a kernel function to work with nonlinear features and learn a nonlinear decision surface

Recovering solution to the primal formulation

original SVM: $\underline{\underline{w}}^T \underline{\underline{\phi(x_n)}} + \underline{\underline{b}}$ $k(x, x_n)$

Weights $\underline{\underline{w}}$ (primal variables) relation to $\underline{\underline{\alpha}}$ (dual variables)

$\underline{\underline{w}} = \sum_n y_n \underline{\underline{\alpha}_n} \phi(x_n)$ ← Linear combination of the input features

Prediction on a test point $\underline{\underline{x}}$

$$h(\underline{\underline{x}}) = \text{SIGN}(\underline{\underline{w}}^T \underline{\underline{\phi(x)}} + b) = \text{SIGN}\left(\sum_n y_n \underline{\underline{\alpha}_n} k(x_n, \underline{\underline{x}}) + b\right)$$

At test time it suffices to know the kernel function!

$$\begin{aligned} \underline{\underline{w}} &= \sum_n y_n \underline{\underline{\alpha}_n} \phi(x_n) \\ &= (\sum_n y_n \underline{\underline{\alpha}_n} \phi(x_n))^T \phi(\underline{\underline{x}}) \\ &= \sum_n y_n \underline{\underline{\alpha}_n} \underline{\underline{\phi(x_n)}}^T \underline{\underline{\phi(x)}} \end{aligned}$$

Recovering solution to the primal formulation

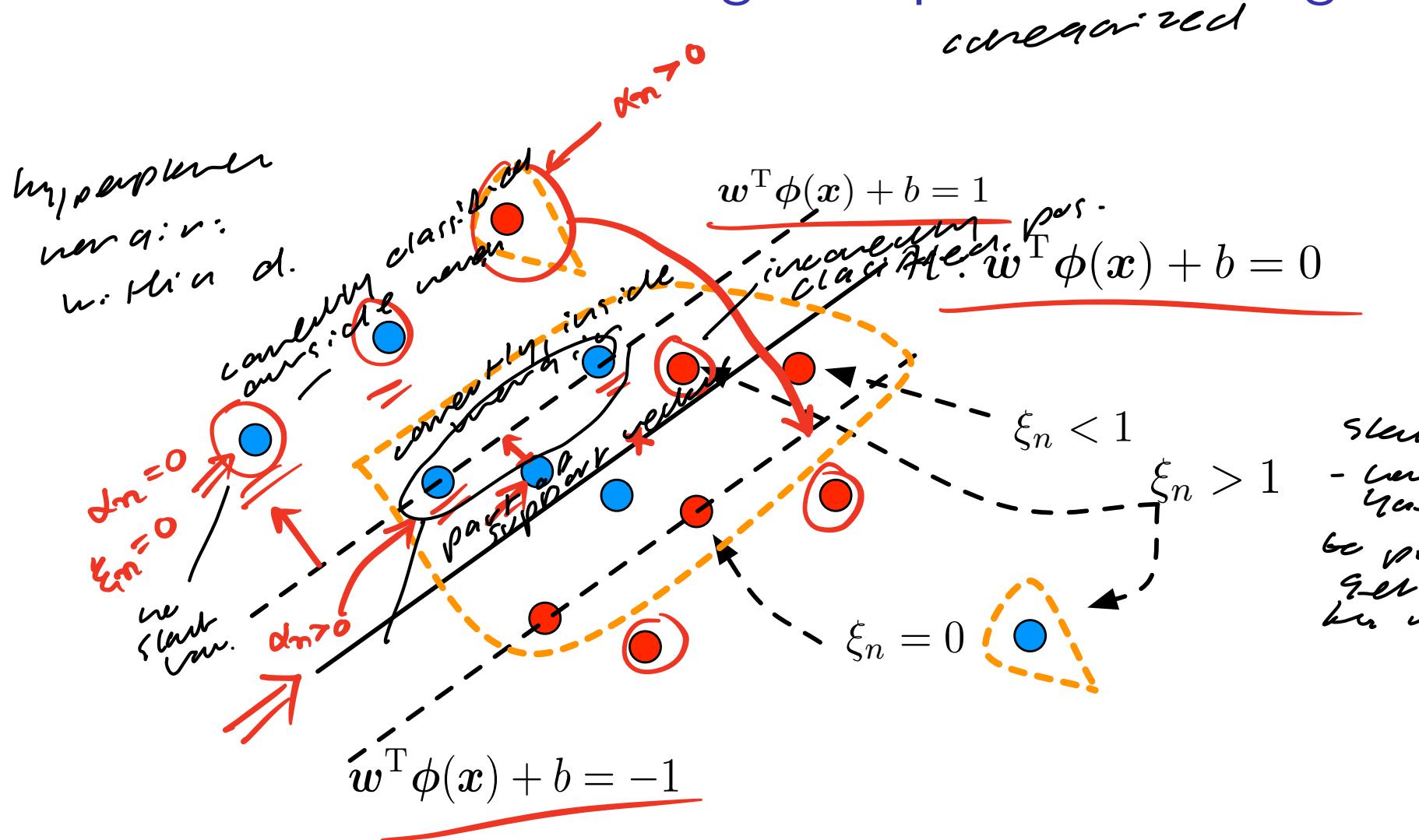
We already identified the primal variable w as

$$w = \sum_n \alpha_n y_n \phi(x_n)$$

$$\underline{0 \leq \alpha_n \leq C}$$

- From the dual, we know that $\underline{0 \leq \alpha_n \leq C}$.
- If $\underline{\alpha_n = 0}$, then the example n does not affect the hyperplane/decision boundary computed by SVM.
 $y_n [w^T \phi(x_n) + b] \geq 1$.
- Only those examples with $\underline{\alpha_n > 0}$ affect the hyperplane/decision boundary. These examples are termed **support vectors**.
- When will $\underline{\alpha_n > 0}$? $\xi_n \geq 0$
 - $\alpha_n = C \Rightarrow y_n [w^T \phi(x_n) + b] \leq 1$.
 - $0 < \alpha_n < C \Rightarrow y_n [w^T \phi(x_n) + b] = 1$.

Visualization of how training data points are categorized



Support vectors are highlighted by the dotted orange lines

Binary classification

Goal is to achieve high accuracy

Accuracy: defined to be the mean of the 0/1 loss $l(y, \hat{y})$.

$$\ell(y, h(\mathbf{x})) = \mathbf{1}\{y \neq h(\mathbf{x})\}$$

$$R^{\text{EMP}}[h(\mathbf{x})] = \frac{1}{N} \sum_n \ell(h(\mathbf{x}_n), y_n)$$

Assumes $y \in \{0, 1\}$ and h outputs values in $\{0, 1\}$.

May not be what we want

Is this email spam?

Does the patient have cancer?

Confusion matrix

Given a dataset of P positive instances and N negative instances,

		Predicted class		
		1	0	
Actual class	1	TP	FN	P
	0	FP	TN	N

P = Positive, N = Negative

TN = True negative, FP = False positive

TP = True positive, FN = False negative

Spam classification

- Positive : spam, Negative: non-spam
- TP: Spam email classified as spam
- FP: Non-spam email classified as spam

Confusion matrix

Given a dataset of P positive instances and N negative instances,

		Predicted class			
		1	0		
Actual class	1	TP	FN	P	N
	0	FP	TN		

P = Positive, N = Negative

TN = True negative, FP = False positive

TP = True positive, FN = False negative

- Accuracy = $\frac{TP+TN}{P+N}$
- Error = $1 - \text{Accuracy}$

Confusion matrix

Given a dataset of P positive instances and N negative instances,

		Predicted class		
		1	0	
Actual class	1	TP	FN	P
	0	FP	TN	N

P = Positive, N = Negative

TN = True negative, FP = False positive

TP = True positive, FN = False negative

- True Positive Rate (TPR) = Recall = Sensitivity = $\frac{TP}{P}$
- Probability of classifying a spam email as spam.

Confusion matrix

Given a dataset of P positive instances and N negative instances,

		Predicted class			
		1	0		
Actual class	1	TP	FN	P	N
	0	FP	TN		

P = Positive, N = Negative

TN = True negative, FP = False positive

TP = True positive, FN = False negative

- Specificity = $\frac{TN}{N}$
- Probability of classifying a non-spam email as non-spam.
- False Positive Rate (FPR) = $1 - \text{Specificity}$

Confusion matrix

Given a dataset of P positive instances and N negative instances,

		Predicted class			
		1	0		
Actual class	1	TP	FN	P	N
	0	FP	TN	N	

P = Positive, N = Negative

TN = True negative, FP = False positive

TP = True positive, FN = False negative

- Precision = Positive Predictive Value (PPV) = $\frac{TP}{TP+FP}$
- Probability that an email classified as spam is spam.
- **Different from specificity!**

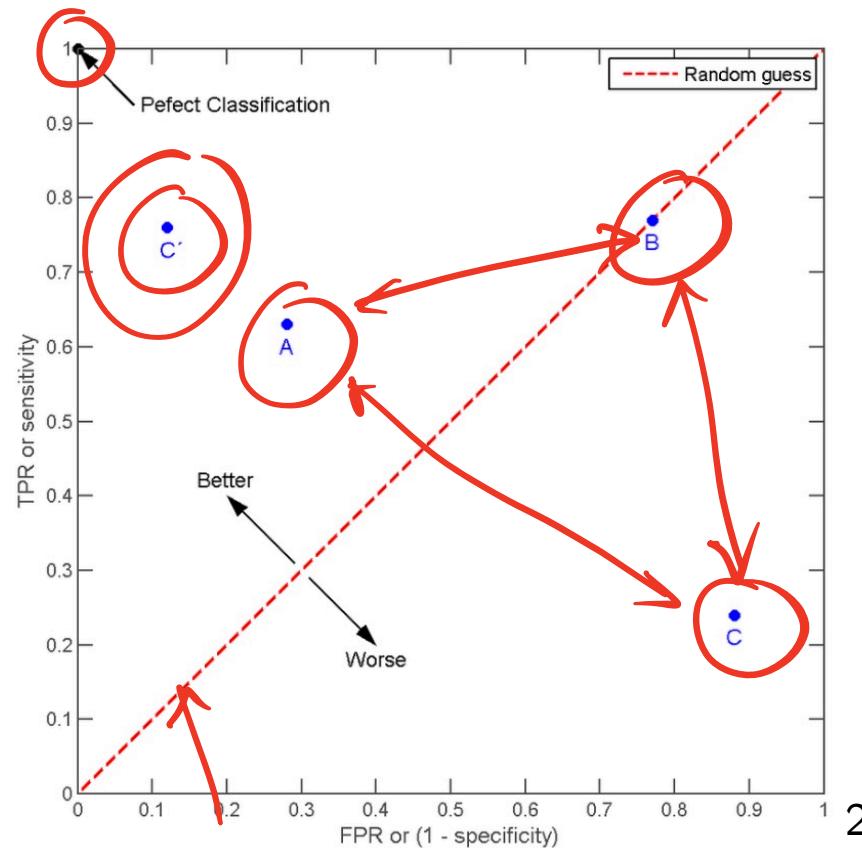
Visualizing performance

Many different performance measures

A		B		C		C'		
TP=63	FN=37	100	TP=77	FN=23	100	TP=24	FN=76	100
FP=28	TN=72	100	FP=77	TN=23	100	FP=88	TN=12	100
91	109	200	154	46	200	112	88	200
								1

ROC curve

TPR (sensitivity) vs FPR (1-specificity)



*perfect classification :
top left, we want
points to be as
close as possible*

- Best possible method is in the upper left corner.
- Random guess would fall on the diagonal.

Sometimes we care about the confidence of predictions

*confidence associated w/
prediction*

- For linear models (e.g., perceptron, logistic regression), the activation $a(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$ gives us a ranking of the predictions.
- In the case of the perceptron, given $\underline{\mathbf{x}_1}$ and $\underline{\mathbf{x}_2}$ such that $a(\underline{\mathbf{x}_1}) > a(\underline{\mathbf{x}_2}) > 0$.
 - ▶ Both $\underline{\mathbf{x}_1}$ and $\underline{\mathbf{x}_2}$ are classified as positive.
 - ▶ Perceptron is more confident in the prediction for \mathbf{x}_1 than for \mathbf{x}_2 .

ROC curve allows us to visualize the ranking of predictions

Building a ROC curve

- Sort the instances according to the confidence of instance being in class 1 (positive class).
 - ▶ How confident is the classifier that an email is spam?
- Step through the sorted list from high to low confidence.
- As you step through, build a binary classifier that predicts the top k instances as belonging to class 1 and the remaining to class 0.
 - ▶ How aggressively do you want to mark a message as spam ?
- Compute the TPR (True Positive Rate) and FPR (False Positive Rate).
 - ▶ High up in the list: Low TPR and low FPR.
 - ▶ As we move down, TPR increases but so does FPR.
- Plot TPR vs FPR.
 - c and

move down
list

Example

class to
1 ← in pos-
class: given

10

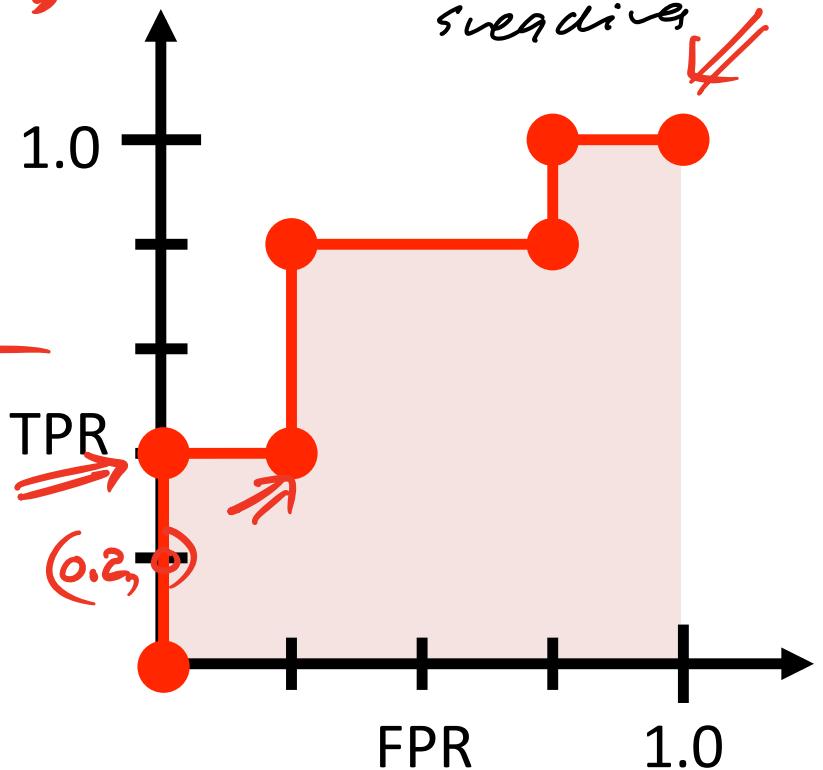
Instance	Confidence in positive	Actual class
9	0.99	1
7	0.98	1
1	0.72	0
2	0.70	1
6	0.65	1
10	0.51	0
3	0.39	0
5	0.24	1
4	0.11	0
8	0.01	0

3: $TPR = \frac{2}{5}$

$$TPR = \frac{TP}{\text{Positives}} \quad FPR = 1 - \frac{TN}{\text{Negatives}}$$

$$= \frac{1}{5} \quad = 1 - \frac{5}{5} = 0$$

0's

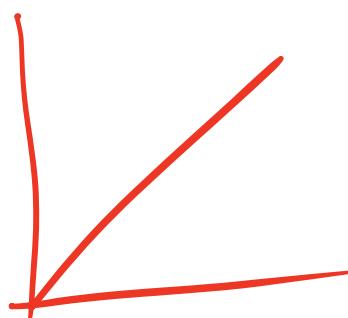


$FPR = \frac{1}{5}$

Interpreting ROC curves

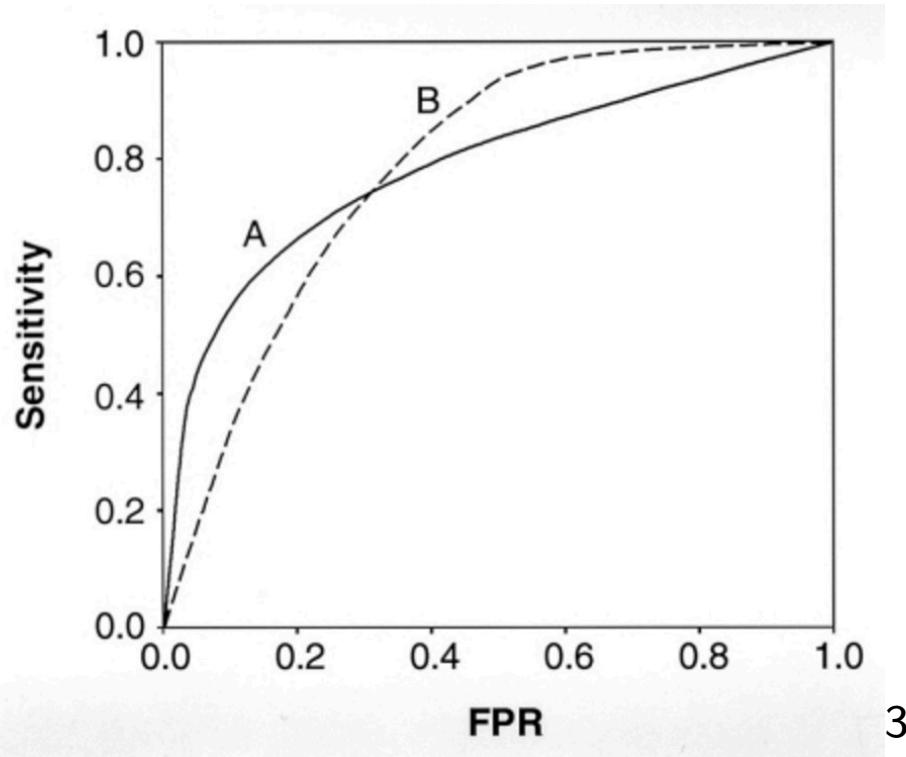
Area under the ROC Curve (AUC or AUROC)

- The probability that a classifier will rank a randomly chosen positive instance (instance with class label 1) higher than a randomly chosen negative one (instance with class label 0).
- A measure of the overall performance of a classifier.
- A random classifier has $\text{AUROC} = \frac{1}{2}$.
- A perfect classifier has $\text{AUROC} = 1$.



Interpreting ROC curves

Classifiers A and B with equal AUROC



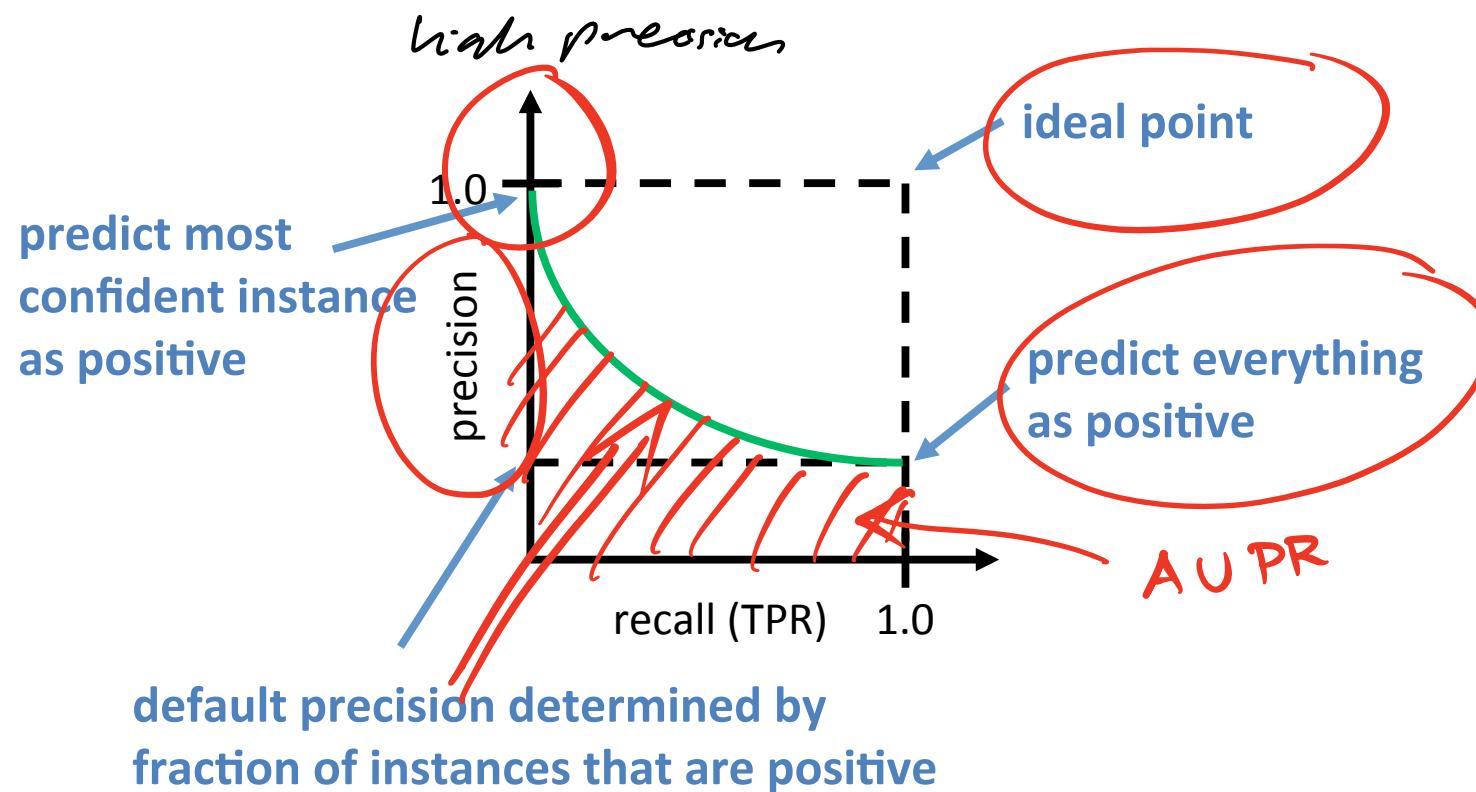
Which one do we choose?

Depends on whether we care about sensitivity or FPR

Other trade-offs

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Precision-recall curve



Interpreting precision-recall curves

Area under the precision-recall curve (AUPR)

- Can compute the area under the precision-recall curve.

Interpreting precision-recall curves

F₁-score

$$F_1 = \frac{1}{\frac{1}{2} \frac{1}{\text{PRECISION}} + \frac{1}{2} \frac{1}{\text{RECALL}}}$$

- Want F₁ to be high.
- The Harmonic mean of precision and recall.
- A single number that trades off precision and recall.
- If a classifier has high precision and low recall (or other way round), its F₁-score will be low.
- Encourages precision and recall values that are similar.

Summary

- Many ways to measure accuracy.
- Think carefully about which ones are most relevant.
- Often there are trade-offs
 - ▶ TPR vs FPR, Precision vs recall
- Graphical displays useful.
 - ▶ ROC and Precision-recall curves display performance at various levels of confidence.
 - ▶ Can be summarized by area under the curve (AUROC, AUPR) as well as numbers such as the F_1 -score.

ensemble methods: multi class classifications

Idea

build several ML alg.

classification predict - multi class

- Consider a set of predictors (**base learners**): h_1, \dots, h_L .
- Combine their predictions to get a more accurate predictor H : (**ensemble**).

When might this work ?

different models \rightarrow more accurate

and overfitting

assumptions:

not all the time

models are same pred. \rightarrow id matching

Idea

- Consider a set of predictors (**base learners**): h_1, \dots, h_L .
- Combine their predictions to get a more accurate predictor H : (**ensemble**).

When might this work ?

diff models \rightarrow comb. better than ,

The predictors make different types of mistakes.

Practical application: the Netflix prize

design a user's prediction based on ratings

Goal: predict how a user will rate a movie

- Based on other user's ratings of the movie.
- Based on this user's ratings of other movies.
- Application called **collaborative filtering**.
- Netflix prize: 1M prize for the first team to do 10% better than Netflix system.
10% more
- Winner: an ensemble of more than 800 rating systems.

ensemble rating systems & combining

Train multiple classifiers on the same dataset

- Train h_1, \dots, h_L on a given training data (e.g., decision trees, logistic regression).
- How to combine their predictions ?
- Majority vote : $h = \text{SIGN}(h_1 + \dots + h_L)$, where $h_l \in \{+1, -1\}$.
- Weighted majority vote: $h = \text{SIGN}(w_1 h_1 + \dots + w_L h_L)$, where $h_l \in \{+1, -1\}$. How do we set the weights ?
 - ▶ Compute weights on validation data, i.e., data not used for learning h_1, \dots, h_L .

$h_1 \dots h_L$

classifiers

new model \rightarrow combines each

of regression models

how to find hyperplane.

split into d.P.C. classes

Train multiple classifiers on the same dataset

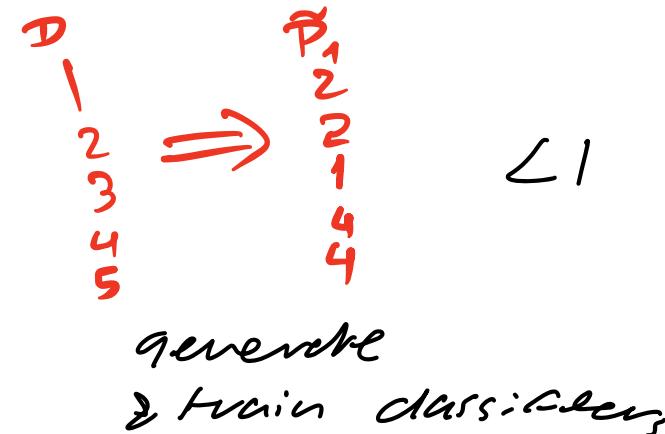
- Train h_1, \dots, h_L on a given training data (e.g., decision trees, logistic regression).
- How to combine their predictions ?
- Majority vote : $h = \text{SIGN}(h_1 + \dots + h_L)$, where $h_l \in \{+1, -1\}$.
- Weighted majority vote: $h = \text{SIGN}(w_1 h_1 + \dots + w_L h_L)$, where $h_l \in \{+1, -1\}$. How do we set the weights ?
 - ▶ Compute weights on validation data, *i.e.*, data not used for learning h_1, \dots, h_L .
 - ▶ What if we are trying to learn a regression model ?
- Can use mean, weighted mean, or median.

Bagging (Bootstrap Aggregation)

Training same classifier on multiple datasets

- Where do we get multiple datasets ?
- One idea: split original training dataset into multiple datasets and train a classifier on each.
 - ▶ Each classifier is trained on a small part of the data and so might not generalize well.

Bagging (Bootstrap Aggregation)



Training same classifier on multiple datasets

- Where do we get multiple datasets ?

Bootstrap resampling

- Training data with N instances: \mathcal{D}
- Create B bootstrap training datasets: $\tilde{\mathcal{D}}_1, \dots, \tilde{\mathcal{D}}_B$.
- Each $\tilde{\mathcal{D}}_b$ contains N training examples drawn randomly from \mathcal{D} with replacement.
- Train a classifier h_b on each of $\tilde{\mathcal{D}}_b$.
- Use a majority vote of h_1, \dots, h_B to classify test data.

$B \leq 100$

generated from original dataset
& sample n times
w/ replacement
(variation)
↑
slightly diff. from
original dataset
Sampling w/
replacement

Boosting

High-level idea: combine a lot of classifiers

- Construct / identify these classifiers one at a time
- Use weak or base classifiers to arrive at complex decision boundaries
(strong classifiers)

Our plan

- Describe AdaBoost algorithm
- Derive the algorithm

relatively simple decision
boundaries
in incremental fashion

in incremental fashion

Adaboost Algorithm

$$w_t(n) = \frac{1}{N}$$

' ' ' same weight

- Given: N samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T (Round) pre-specified
 - Train a weak classifier $h_t(x)$ using current weights $w_t(n)$, by minimizing the weighted classification error

$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(x_n)]$$

random
weighted
classification

mismatch? —
Indicator = $\begin{cases} 1 & \text{if condition is true} \\ 0 & \text{else} \end{cases}$

minimizer

meta algorithm

weak & base

→ learn & construct

can we take weak & combine to strong

misclassified

with adaptability due to all
after t iterations

Adaboost Algorithm

what is the weight of the classifier?

- Given: N samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = \underline{1}$ to T
 - Train a weak classifier $h_t(x)$ using current weights $w_t(n)$, by minimizing the weighted classification error

$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(x_n)]$$

weak t , based on ever

2 Compute contribution for this classifier:

$$\beta_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

$\epsilon_t \downarrow 0$ $\log \frac{1 - \epsilon_t}{\epsilon_t}$ contribution

more accurate, & contributions

how to weight them?

everything starts w/ equal weights

Adaboost Algorithm

*weak classifier
- never below $\frac{1}{2}$ error*

- Given: N samples $\{\mathbf{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
 - Train a weak classifier $h_t(\mathbf{x})$ using current weights $w_t(n)$, by minimizing the weighted classification error

*same
at each
iteration*

$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)]$$

- Compute contribution for this classifier: $\underline{\beta_t} = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
- Update weights on training points

$$\underline{w_{t+1}(n)} \propto w_t(n) e^{-\underline{\beta_t} y_n h_t(\mathbf{x}_n)}$$

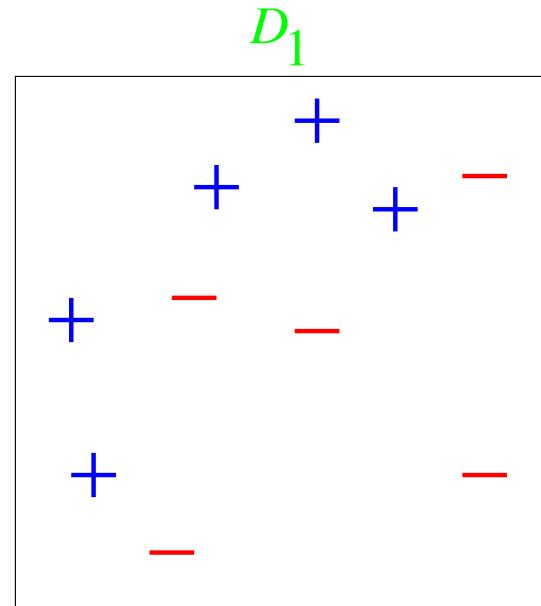
and normalize them such that $\sum_n w_{t+1}(n) = 1$

- Output the final classifier

$$h[\mathbf{x}] = \text{sign} \left[\sum_{t=1}^T \underline{\beta_t} h_t(\mathbf{x}) \right]$$

Example

10 data points and 2 features



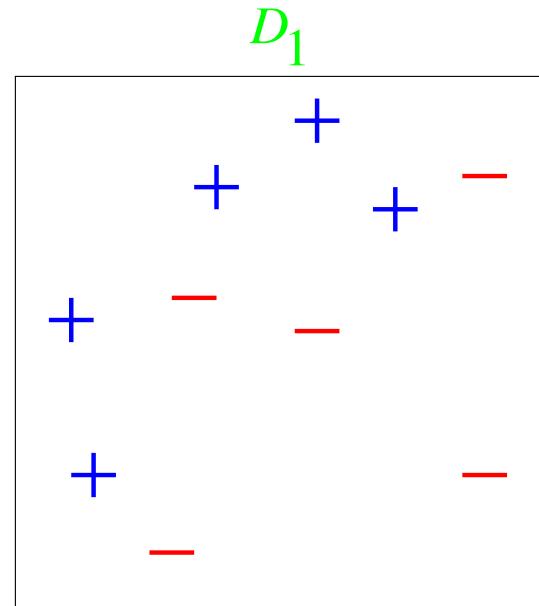
horizontal /
vertical lines

- The data points are clearly not linearly separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")
- Base classifier $h(\cdot)$: either horizontal or vertical lines

Example

10 data points and 2 features

not linearly separable
= weights
needs a weak classifier



↑ version on x or y axis

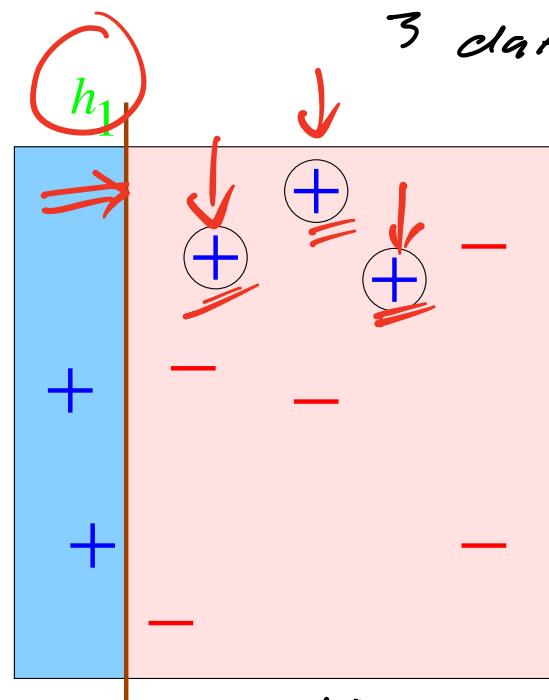
decision stumps:
trees w/ one
interval node

horizontal/vertical
lines

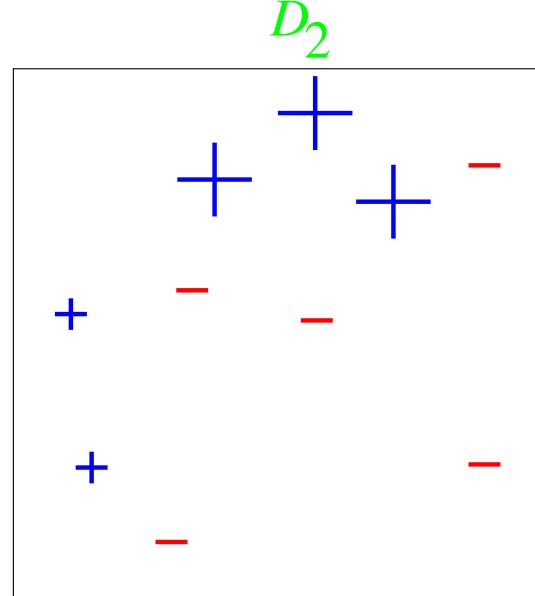
- The data points are clearly not linearly separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")
- Base classifier $h(\cdot)$: either horizontal or vertical lines
 - ▶ These 'decision stumps' are just trees with a single internal node, i.e., they classifying data based on a single attribute

Round 1: $t = 1$

$$\sum_n w_1(n) \cdot \mathbb{1}_{\{y_n \neq h_1(x_n)\}}$$



3 data points misclassified



misclassified:

$$\epsilon_1 = \frac{3}{10}$$

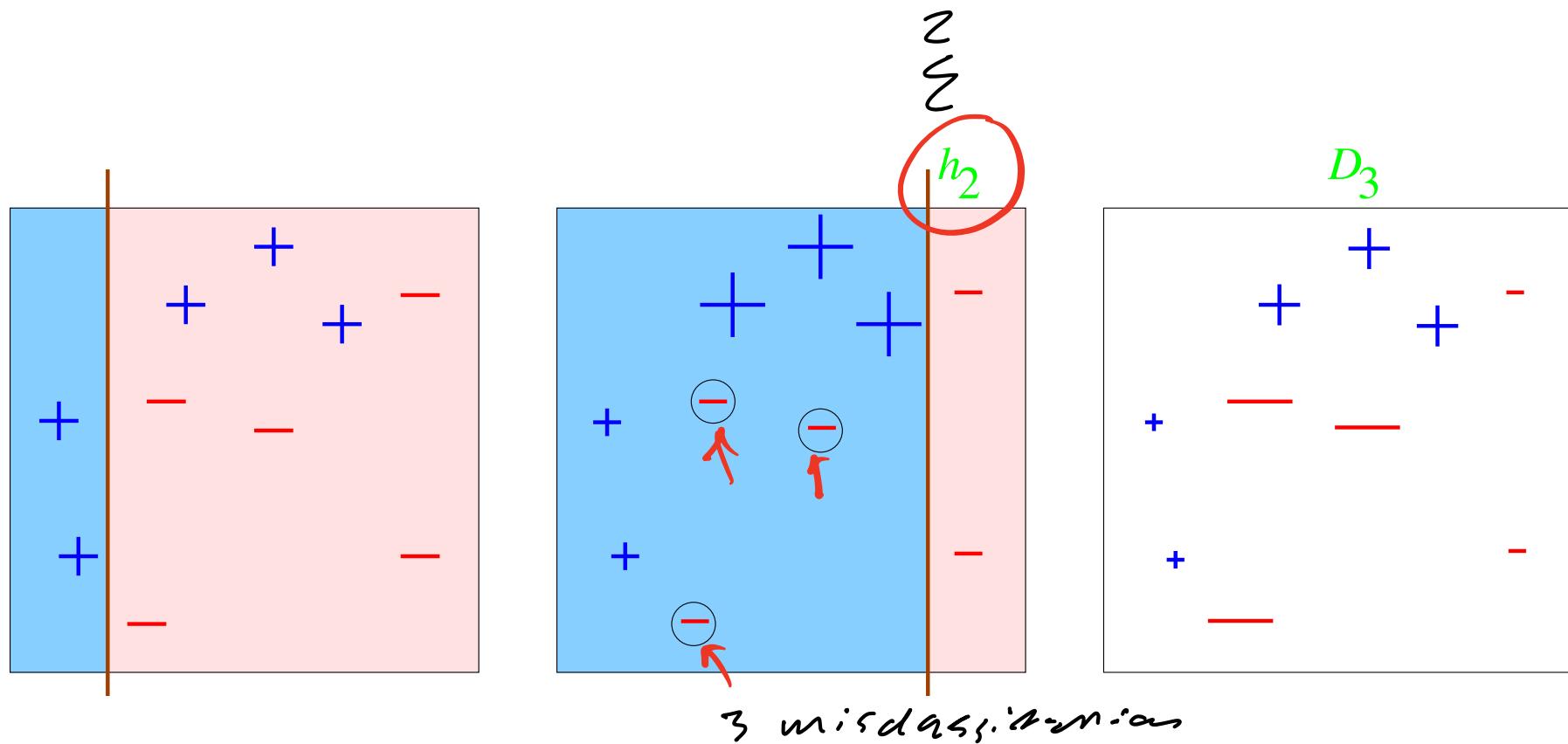


weighted classification error

$$\beta_1 = \frac{1}{2} \log \left(\frac{1-\epsilon_1}{\epsilon_1} \right)$$

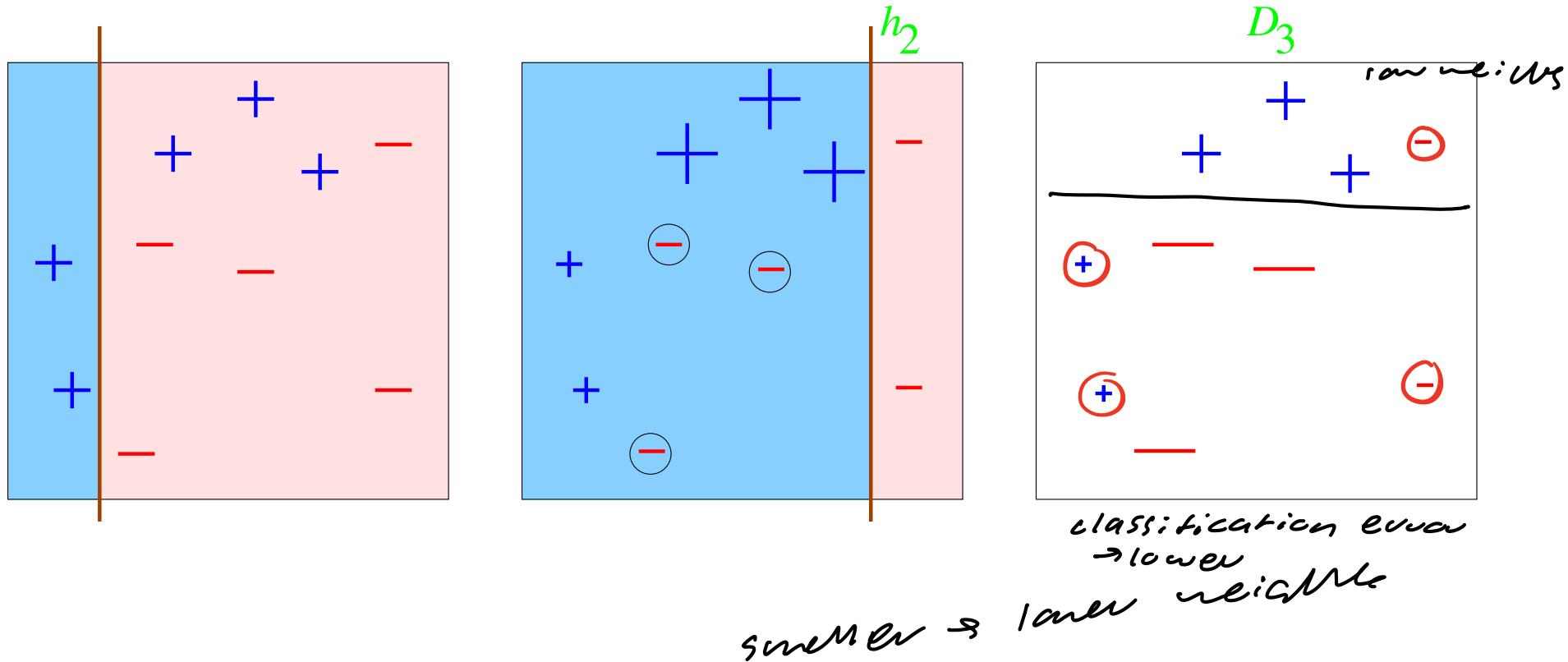
choose based on lowest classification error

Round 2: $t = 2$



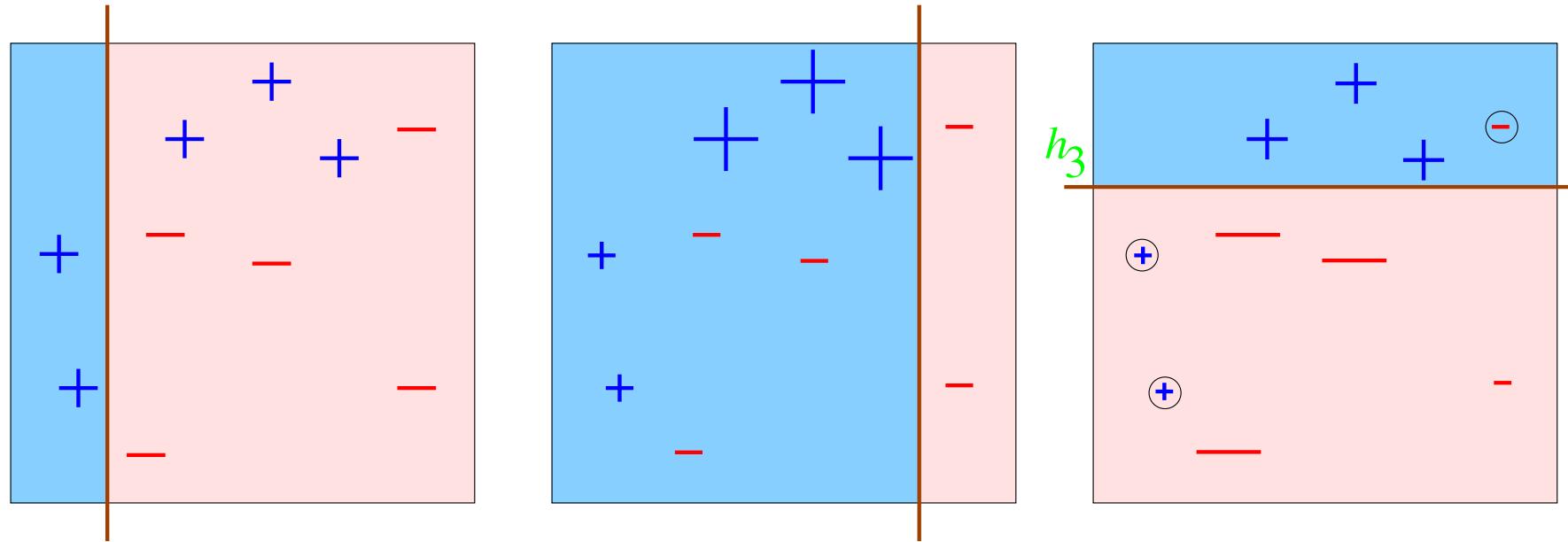
- 3 misclassified (with circles): $\epsilon_1 = 0.3 \rightarrow \beta_1 = \underline{\underline{0.42}}$.
- Weights recomputed; the 3 misclassified data points receive larger weights

Round 2: $t = 2$



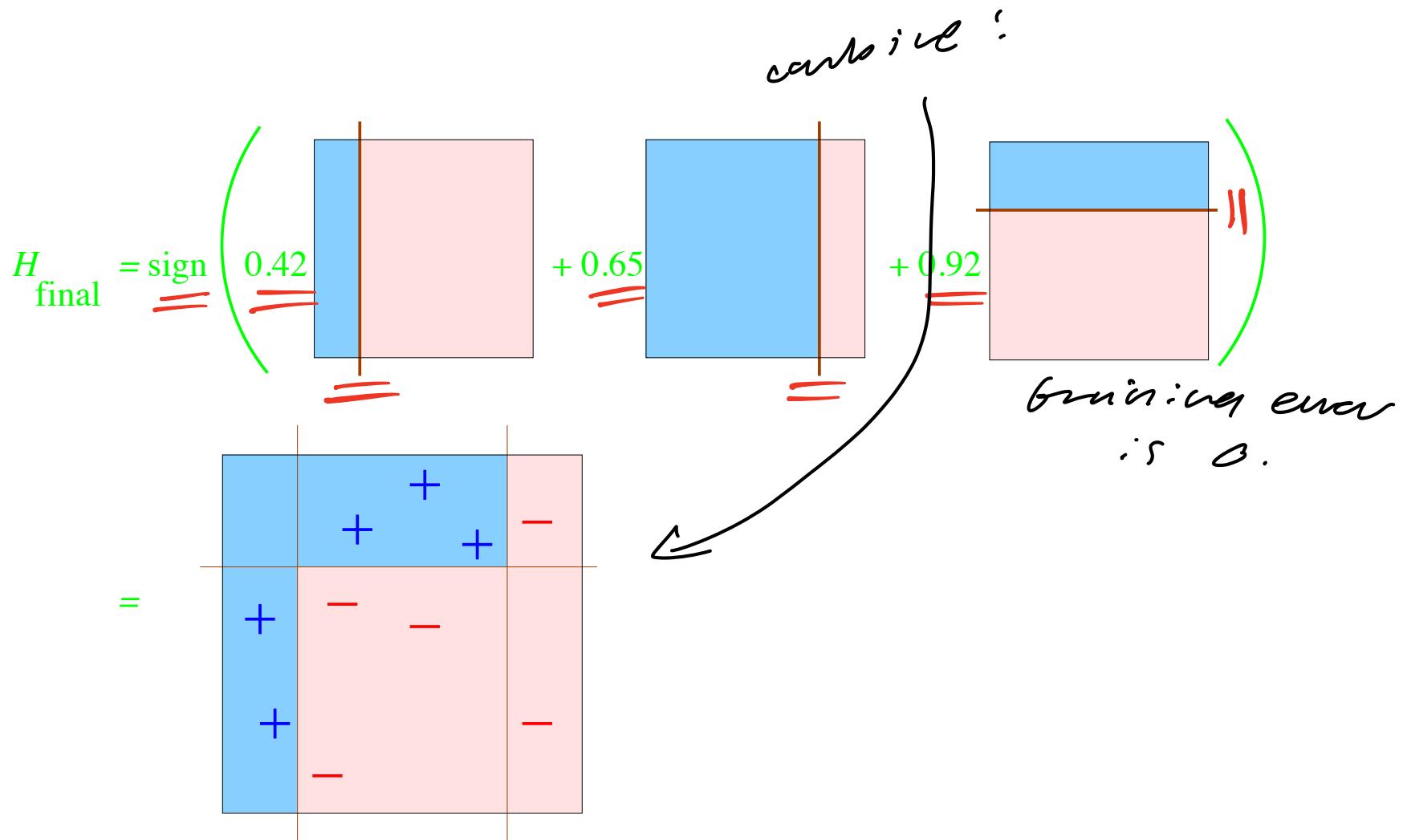
- 3 misclassified (with circles): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
Note that $\epsilon_2 \neq 0.3$ as those 3 data points have weights less than $1/10$
- 3 misclassified data points get larger weights
- Data points classified correctly in both rounds have small weights

Round 3: $t = 3$



- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low; what's the intuition?
 - ▶ Since they have been consistently classified correctly, this round's mistake will hopefully not have a huge impact on the overall prediction

Final classifier: combining 3 classifiers



- All data points are now classified correctly!

Why AdaBoost works?

Start w/ weak classifiers?

utilizing view

adaBoost - minimizing loss function

It minimizes a loss function related to the classification error.

0/1 loss

- Suppose we want to have a classifier

$$h(\mathbf{x}) = \text{sign}[a(\mathbf{x})] = \begin{cases} 1 & \text{if } a(\mathbf{x}) > 0 \\ -1 & \text{if } a(\mathbf{x}) < 0 \end{cases}$$

- Our loss function is thus

$$\ell(y, h(\mathbf{x})) = \begin{cases} 0 & \text{if } \underline{ya(\mathbf{x})} > 0 \\ 1 & \text{if } \underline{ya(\mathbf{x})} < 0 \end{cases}$$

Namely, the function $a(\mathbf{x})$ and the target label y should have the same sign to avoid a loss of 1.

Surrogate loss

0 – 1 loss function $\ell(h(x), y)$ is non-convex and difficult to optimize.

We can instead use a surrogate loss – what are examples?

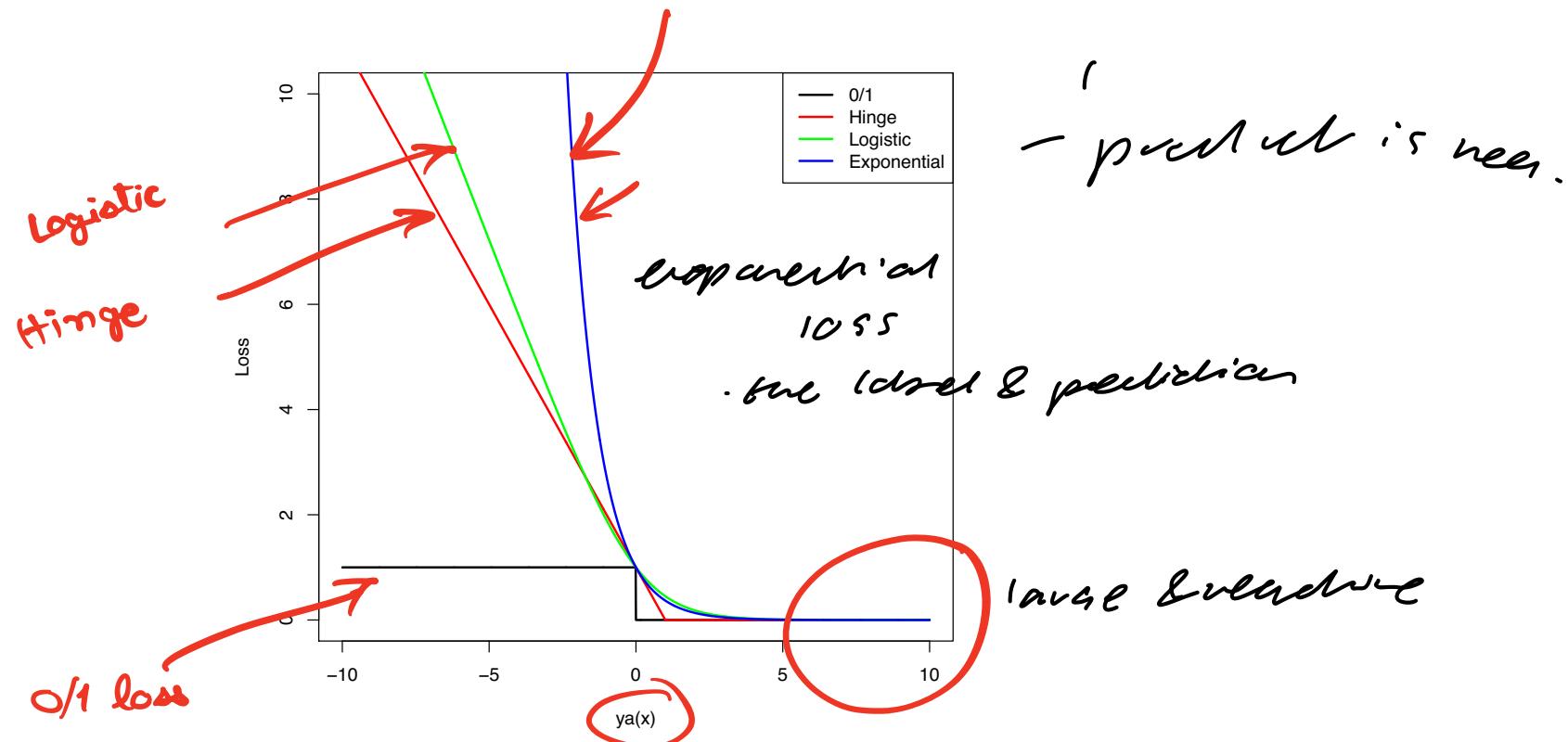
Exponential Loss

$$\ell^{\text{EXP}}(y, h(x)) = e^{-ya(x)}$$

$\ell^{\text{EXP}}(y, h(x))$ is easier to optimize as it is differentiable

$$e^{-ya(x)} \rightarrow 0$$

$ya(x) > 0$



Choosing the t -th classifier

Suppose we have built a classifier $a_{t-1}(\mathbf{x})$, and we want to improve it by adding a weak learner $\underline{h_t(\mathbf{x})}$

$$\underline{a(\mathbf{x})} = \underline{a_{t-1}(\mathbf{x})} + \beta_t \underline{h_t(\mathbf{x})}$$

*associates exponential loss
with weak learners*

How can we choose optimally the new classifier $\underline{h_t(\mathbf{x})}$ and the combination coefficient β_t ?

Adaboost greedily *minimizes the empirical risk of the exponential loss function.*

$$(\underline{h_t^*(\mathbf{x})}, \underline{\beta_t^*}) = \arg \min_{(h_t(\mathbf{x}), \beta_t)} \sum_n e^{-y_n \underline{a(\mathbf{x}_n)}}$$

maximized

*add a
new
classifier w/
a weight*

Choosing the t -th classifier

Suppose we have built a classifier $a_{t-1}(\mathbf{x})$, and we want to improve it by adding a weak learner $h_t(\mathbf{x})$

$$a(\mathbf{x}) = \underline{\underline{a_{t-1}(\mathbf{x})}} + \underline{\underline{\beta_t h_t(\mathbf{x})}}$$

How can we choose optimally the new classifier $h_t(\mathbf{x})$ and the combination coefficient β_t ?

Adaboost greedily *minimizes the empirical risk of the exponential loss function.*

$$\begin{aligned}(h_t^*(\mathbf{x}), \beta_t^*) &= \arg \min_{(h_t(\mathbf{x}), \beta_t)} \sum_n e^{-y_n \underline{\underline{a(\mathbf{x}_n)}}} \\&= \arg \min_{(h_t(\mathbf{x}), \beta_t)} \sum_n e^{-y_n [\underline{\underline{a_{t-1}(\mathbf{x}_n)}} + \underline{\underline{\beta_t h_t(\mathbf{x}_n)}}]} \\&\quad \sum_n \underline{\underline{e^{-y_n a_{t-1}(\mathbf{x}_n)}}} \times \underline{\underline{e^{-y_n \beta_t h_t(\mathbf{x}_n)}}}\end{aligned}$$

Choosing the t -th classifier

Suppose we have built a classifier $a_{t-1}(\mathbf{x})$, and we want to improve it by adding a weak learner $h_t(\mathbf{x})$

$$a(\mathbf{x}) = a_{t-1}(\mathbf{x}) + \beta_t h_t(\mathbf{x})$$

How can we choose optimally the new classifier $h_t(\mathbf{x})$ and the combination coefficient β_t ?

Adaboost greedily *minimizes the empirical risk of the exponential loss function.*

$$\begin{aligned}(h_t^*(\mathbf{x}), \beta_t^*) &= \arg \min_{(h_t(\mathbf{x}), \beta_t)} \sum_n e^{-y_n a(\mathbf{x}_n)} \\&= \arg \min_{(h_t(\mathbf{x}), \beta_t)} \sum_n e^{-y_n [a_{t-1}(\mathbf{x}_n) + \beta_t h_t(\mathbf{x}_n)]} \\&= \arg \min_{(h_t(\mathbf{x}), \beta_t)} \sum_n \underbrace{w_t(n)}_{\text{which associated}} e^{-y_n \beta_t h_t(\mathbf{x}_n)} \quad \text{class weight}\end{aligned}$$

where we have used $w_t(n)$ as a shorthand for $e^{-y_n a_{t-1}(\mathbf{x}_n)}$

The new classifier

$$h_t(x_n) \in \{-1, +1\}$$

$$y_n \in \{-1, +1\}$$

We can decompose the *weighted* loss function into two parts

$$\sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}$$

$$\text{minimizing } (\beta_t, h_t(x)) \text{ with respect to } \begin{cases} \mathbb{1}\{y_n = h_t(x_n)\} \\ + \mathbb{1}\{\underline{y_n} \neq h_t(x_n)\} \end{cases}$$

$$= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(x_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(x_n)]$$

$$= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(x_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(x_n)])$$

$$= (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(x_n)] + e^{-\beta_t} \sum_n w_t(n)$$

$$e^{\beta_t} \sum_n w_t(n) \mathbb{I}\{\underline{y_n} \neq h_t(x_n)\}$$

$$- e^{-\beta_t} \sum_n w_t(n) \mathbb{I}\{\underline{y_n} \neq h_t(x_n)\}$$

$$\sum_n w_t(n) e^{-\beta_t} - \sum_n w_t(n) e^{-\beta_t} \times \mathbb{I}\{\underline{y_n} \neq h_t(x_n)\}$$

$$\sum_n w_t(n) e^{-\beta_t} \times \mathbb{I}\{\underline{y_n} \neq h_t(x_n)\}$$

The new classifier

We can decompose the *weighted* loss function into two parts

$$\begin{aligned} & \sum_n w_t(n) e^{-y_n \beta_t h_t(\mathbf{x}_n)} \\ &= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\mathbf{x}_n)] \\ &= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)]) \\ &= (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)] + e^{-\beta_t} \sum_n w_t(n) \end{aligned}$$

We have used the following properties to derive the above

- $y_n h_t(\mathbf{x}_n)$ is either 1 or -1 as $h_t(\mathbf{x}_n)$ is the output of a binary classifier
- The indicator function $\mathbb{I}[y_n = h_t(\mathbf{x}_n)]$ is either 0 or 1, so it equals $1 - \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)]$

Finding the optimal weak learner

Summary

$$\begin{aligned}(h_t^*(\mathbf{x}), \beta_t^*) &= \arg \min_{(h_t(\mathbf{x}), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\mathbf{x}_n)} \\&= \arg \min_{(h_t(\mathbf{x}), \beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)] \\&\quad + e^{-\beta_t} \sum_n w_t(n)\end{aligned}$$

What term(s) must we optimize to choose $h_t(\mathbf{x}_n)$?

$$h_t^*(\mathbf{x}) = \arg \min_{h_t(\mathbf{x})} \epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)]$$

Minimize weighted classification error as noted in step 1 of Adaboost!

How to choose β_t ?

Summary

compute β_t which
minimizes the
error

$$\begin{aligned}(h_t^*(\mathbf{x}), \beta_t^*) &= \arg \min_{(h_t(\mathbf{x}), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\mathbf{x}_n)} \\ &= \arg \min_{(h_t(\mathbf{x}), \beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)] \\ &\quad + e^{-\beta_t} \sum_n w_t(n)\end{aligned}$$

What term(s) must we optimize to choose β_t ?

We need to minimize the entire objective function with respect to β_t !

We can do this by taking derivative with respect to β_t , setting to zero, and solving for β_t . After some calculation and using $\sum_n w_t(n) = 1$, we find:

$$\beta_t^* = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

which is precisely step 2 of Adaboost! (*Exercise – verify the solution*)

Updating the weights

Once we find the optimal weak learner we can update our classifier:

$$a(\mathbf{x}) = a_{t-1}(\mathbf{x}) + \beta_t^* h_t^*(\mathbf{x})$$

We then need to compute the weights for the above classifier as:

$$\begin{aligned} w_{t+1}(n) &= e^{-y_n a(\mathbf{x}_n)} = e^{-y_n [a_{t-1}(\mathbf{x}) + \beta_t^* h_t^*(\mathbf{x}_n)]} \\ &= w_t(n) e^{-y_n \beta_t^* h_t^*(\mathbf{x}_n)} = \begin{cases} \frac{w_t(n) e^{\beta_t^*}}{\text{compare from prev. round}} & \text{if } y_n \neq h_t^*(\mathbf{x}_n) \\ \underline{\underline{w_t(n) e^{-\beta_t^*}}} & \text{if } y_n = h_t^*(\mathbf{x}_n) \end{cases} \end{aligned}$$

*compute weight_{new} classifier
from prev. round*

Intuition Misclassified data points will get their weights increased, while correctly classified data points will get their weight decreased

Meta-Algorithm

Note that the AdaBoost algorithm itself never specifies how we would get $h_t^*(x)$ as long as it minimizes the weighted classification error

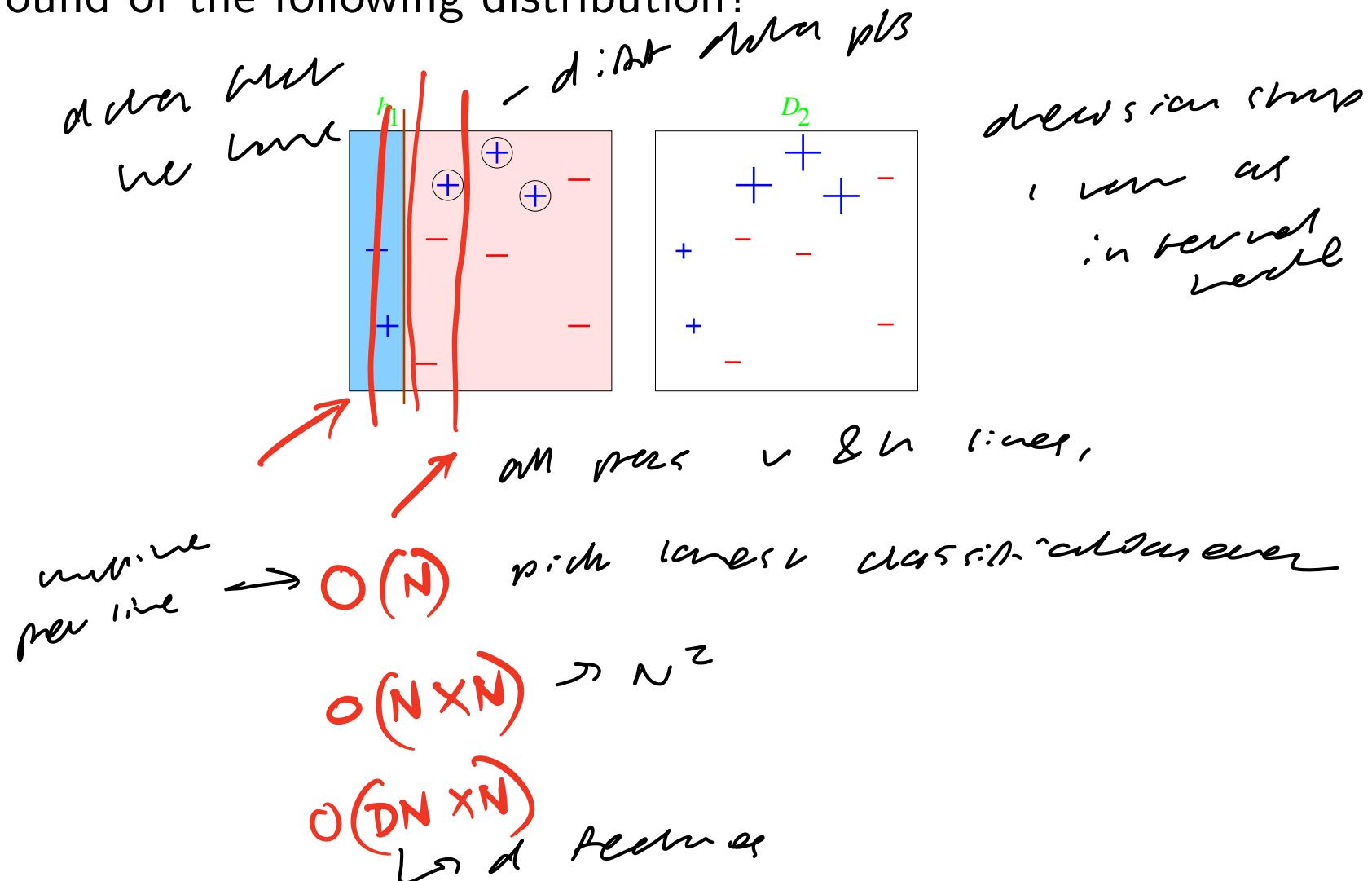
$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t^*(x_n)]$$

*any type of
classifier
- it minimizes
the weight*

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any type of classifier

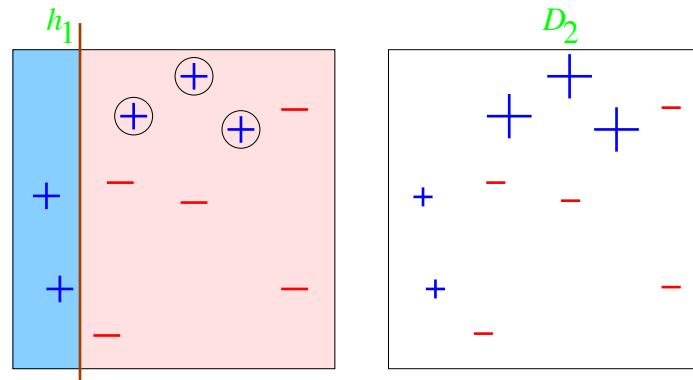
Decision Stumps

How do we choose the decision stump classifier given the weights at the second round of the following distribution?



Decision Stumps

How do we choose the decision stump classifier given the weights at the second round of the following distribution?



computationally easy
time

We can simply enumerate all possible ways of putting vertical and horizontal lines to separate the data points into two classes and find the one with the smallest weighted classification error! Runtime?

- Presort data by each feature in $O(dN \log N)$ time; in ~~ACP~~ ^{even} ~~ACP~~ ^{ACP} occurs
- Evaluate $N + 1$ thresholds for each feature at each round in $O(dN)$ time
- In total $O(dN \log N + dNT)$ time – this efficiency is an attractive quality of boosting!
*simple decision stump → weak
and & stronger learners*

Interpreting boosting as learning nonlinear basis

Two-stage process

- Get $\text{SIGN}[a_1(\mathbf{x})], \text{SIGN}[a_2(\mathbf{x})], \dots,$
- Combine into a linear classification model

$$y = \text{SIGN} \left\{ \sum_t \beta_t \text{SIGN}[a_t(\mathbf{x})] \right\}$$

In other words, each stage learns a nonlinear basis $\phi_t(\mathbf{x}) = \text{SIGN}[a_t(\mathbf{x})]$

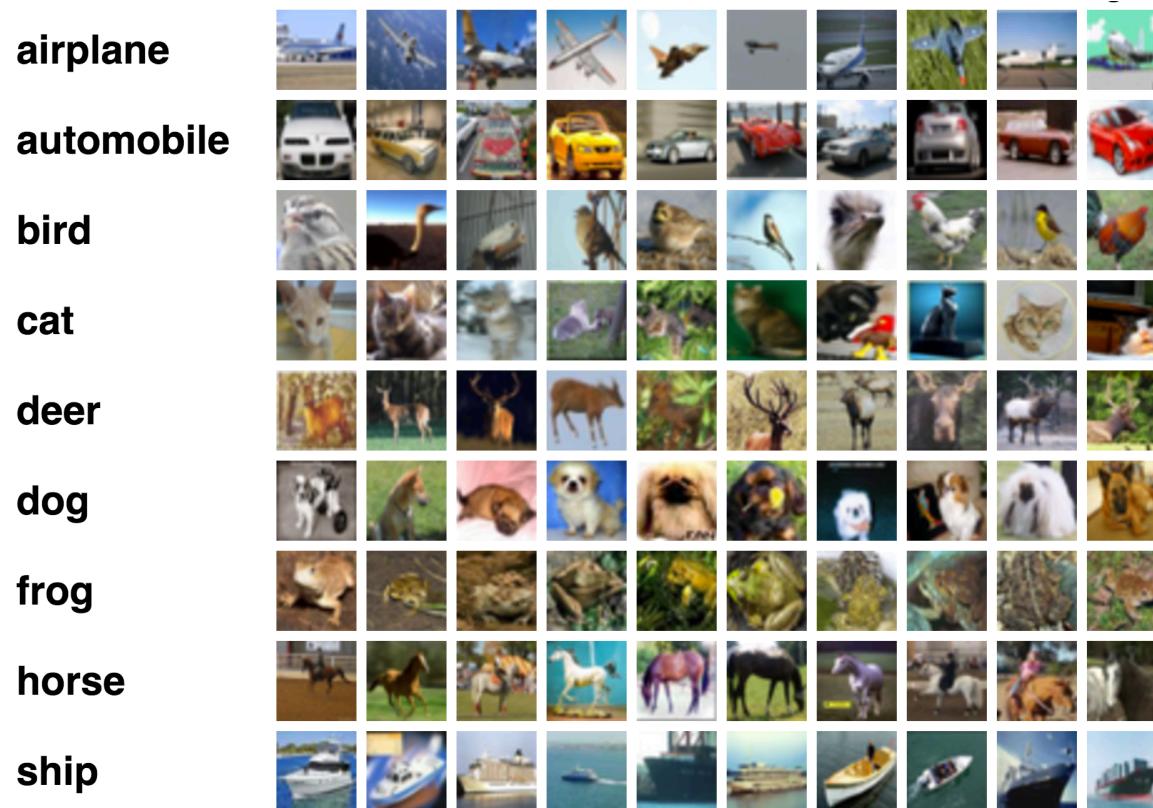
- This is an alternative way to introduce non-linearity aside from kernel methods

Setup

Suppose we need to predict multiple classes/outcomes:

$$C_1, C_2, \dots, C_K$$

- Weather prediction: sunny, cloudy, raining, etc
- Optical character recognition: 10 digits + 26 characters (lower and upper cases) + special characters, etc
- Object recognition



Two key ideas to solve multi-class classification

- Reduction to binary
 - ▶ Easy to implement
 - ▶ Uses binary classification as a black-box (binary classification is well-understood in theory and practice).
- Single classifier: consider all classes simultaneously.
 - ▶ Directly minimize the empirical risk.
 - ▶ Easy to add constraints and domain knowledge.

minimizes empirical risk

One versus the rest

- For each class C_k , change the problem into binary classification
 - ① Relabel training data with label C_k , into POSITIVE (or '1')
 - ② Relabel all the rest data into NEGATIVE (or '0')

This step is often called *1-of-K* encoding. That is, only one is nonzero and everything else is zero.

Example: for class C_2 , data go through the following change

$$(\mathbf{x}_1, C_1) \rightarrow (\mathbf{x}_1, 0), (\mathbf{x}_2, C_3) \rightarrow (\mathbf{x}_2, 0), \dots, (\mathbf{x}_n, C_2) \rightarrow (\mathbf{x}_n, 1), \dots,$$

- Train K binary classifiers to differentiate the two classes

One versus the rest

- For each class C_k , change the problem into binary classification
 - ➊ Relabel training data with label C_k , into POSITIVE (or '1')
 - ➋ Relabel all the rest data into NEGATIVE (or '0')

This step is often called *1-of-K* encoding. That is, only one is nonzero and everything else is zero.

Example: for class C_2 , data go through the following change

$$(\mathbf{x}_1, C_1) \rightarrow (\mathbf{x}_1, 0), (\mathbf{x}_2, C_3) \rightarrow (\mathbf{x}_2, 0), \dots, (\mathbf{x}_n, C_2) \rightarrow (\mathbf{x}_n, 1), \dots,$$

- Train K binary classifiers to differentiate the two classes
- When predicting on \mathbf{x} , combine the outputs of all binary classifiers
 - ➊ What if all the classifiers say NEGATIVE?
 - ➋ What if multiple classifiers say POSITIVE?

OneVersusRestTrain

w.: q'ind dataset
 $D^{multi} = \{(x_1, \textcolor{red}{y_1}), \dots, (x_N, \textcolor{red}{y_N})\}$ *viewing classes as
dataset*

Algorithm 1 OneVersusRestTrain ($D^{multi}, BinaryTrain$)

- 1: **for** $k = 1 \dots K$ **do**
- 2: $\underline{D^k} \leftarrow Relabel(\underline{D^{multi}}, \underline{C_k})$
- 3: $\underline{f_k} \leftarrow BinaryTrain(\underline{D^k})$
- 4: **end for**
- 5: **return** f_1, \dots, f_K

in view classes: by
 $Relabel(D^{multi}, C_k)$: Relabel training data with label C_k , into POSITIVE (or '1'). Relabel all the rest data into NEGATIVE (or '0')

OneVersusRestPredict

x_{binary}
✓ score / decision
voted. v?

Algorithm 2 OneVersusRestPredict ($f_1, \dots, f_K, \underline{x}$)

```
1:  $score \leftarrow (0, \dots, 0)$  score  
2: for  $k = 1 \dots K$  do  
3:    $\hat{y} \leftarrow f_k(\underline{x})$  f_k(x)  
4:    $score_k \leftarrow score_k + \hat{y}$  score over each classifier  
5: end for  
6: return  $\operatorname{argmax}_k score_k$  largest score
```

One versus one

- For each *pair* of classes C_k and $\underline{C_{k'}}$, change the problem into binary classification
 - 1 Relabel training data with label C_k , into POSITIVE (or '1')
 - 2 Relabel training data with label $\underline{C_{k'}}$ into NEGATIVE (or '0')
 - 3 *Disregard* all other data

Ex: for class C_1 and C_2 ,

$$(\mathbf{x}_1, \underline{C_1}), (\mathbf{x}_2, \cancel{\underline{C_3}}), (\mathbf{x}_3, \underline{C_2}), \dots \rightarrow (\mathbf{x}_1, 1), (\mathbf{x}_3, 0), \dots$$

One versus one

- For each *pair* of classes C_k and $C_{k'}$, change the problem into binary classification

- 1 Relabel training data with label C_k , into POSITIVE (or '1')
- 2 Relabel training data with label $C_{k'}$ into NEGATIVE (or '0')
- 3 *Disregard* all other data

Ex: for class C_1 and C_2 ,

$$(\mathbf{x}_1, C_1), (\mathbf{x}_2, C_3), (\mathbf{x}_3, C_2), \dots \rightarrow (\mathbf{x}_1, 1), (\mathbf{x}_3, 0), \dots$$

- Train $K(K - 1)/2$ binary classifiers to differentiate the two classes

One versus one

- For each *pair* of classes C_k and $C_{k'}$, change the problem into binary classification
 - 1 Relabel training data with label C_k , into POSITIVE (or '1')
 - 2 Relabel training data with label $C_{k'}$ into NEGATIVE (or '0')
 - 3 *Disregard* all other data

Ex: for class C_1 and C_2 ,

$$(\mathbf{x}_1, C_1), (\mathbf{x}_2, C_3), (\mathbf{x}_3, C_2), \dots \rightarrow (\mathbf{x}_1, 1), (\mathbf{x}_3, 0), \dots$$

- Train $K(K - 1)/2$ binary classifiers to differentiate the two classes
- When predicting on \mathbf{x} , combine the outputs of all binary classifiers
There are $K(K - 1)/2$ votes.

OneVersusOneTrain

$$D^{multi} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$$

Algorithm 3 OneVersusOneTrain ($D^{multi}, BinaryTrain$)

```
1: for Pairs  $(l, k)$  from  $1, \dots, K$  do
2:    $D^k \leftarrow Relabel(D^{multi}, C_l, C_k)$ 
3:    $f_{lk} \leftarrow BinaryTrain(D^k)$ 
4: end for
5: return  $f_{lk}$ 
```

$Relabel(D^{multi}, c_l, C_k)$: Relabel training data with label C_k , into POSITIVE (or '1'). Relabel training data with label $C_{k'}$ into NEGATIVE (or '0'). *Disregard* all other data

OneVersusOnePredict

Algorithm 4 OneVersusOnePredict (f_{lk}, \underline{x})

```
1:  $score \leftarrow (0, \dots, 0)$ 
2: for Pairs  $(l, k)$  from  $1, \dots, K$  do
3:    $\hat{y} \leftarrow f_{lk}(\underline{x})$ 
4:    $score_l \leftarrow score_l + \hat{y}$            increment of  
the score
5:    $score_k \leftarrow score_k - \hat{y}$        sum
6: end for
7: return  $\text{argmax}_k score_k$       output class w/ max score
```

Contrast these two approaches

Pros and cons of each approach

- *one versus the rest*: only needs to train K classifiers.
 - ▶ Makes a *big* difference if you have a lot of *classes* to go through.
 - ▶ Assumption: Each class individually separable from all others.
- *one versus one*: only needs to train a smaller subset of data (only those labeled with those two classes would be involved).
 - ▶ Makes a *big* difference if you have a lot of *data* to go through. Can lead to overfitting if you do not have a lot of data.
 - ▶ Assumption: Each pair of classes separable.

Bad about both of them

Combining classifiers' outputs seem to be a bit tricky.

Any other good methods?

Binary logistic regression

Intuition

between c & 1

$$P(Y=1 | \mathbf{x}; \boldsymbol{\theta}) = \underline{\sigma(\boldsymbol{\theta}^T \mathbf{x})}$$

$$\hat{y} = \arg \max_{c \in \{0,1\}} p(y=c | \mathbf{x})$$

$$= \begin{cases} 1 & \underline{p(y=1 | \mathbf{x}; \boldsymbol{\theta})} > \frac{1}{2} \\ 0 & \underline{p(y=1 | \mathbf{x}; \boldsymbol{\theta})} \leq \frac{1}{2} \end{cases}$$

$$= \begin{cases} 1 & \text{red arrow} \underline{p(y=1 | \mathbf{x}; \boldsymbol{\theta})} > \frac{1}{2} \\ 0 & \text{red arrow} \underline{p(y=0 | \mathbf{x}; \boldsymbol{\theta})} \geq \frac{1}{2} \end{cases}$$

$$= \begin{cases} 1 & \sigma(\boldsymbol{\theta}^T \mathbf{x}) > \frac{1}{2} \\ 0 & 1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}) \geq \frac{1}{2} \end{cases}$$

and same &
variance, i.e.

$$\begin{aligned} P(Y=0 | \mathbf{x}; \boldsymbol{\theta}) &= 1 - \underline{\sigma(\boldsymbol{\theta}^T \mathbf{x})} \\ &\geq \frac{1}{2} \end{aligned}$$

Multinomial logistic regression

Intuition

$$\hat{y} = \arg \max_c p(y = c | \mathbf{x}; \boldsymbol{\theta}_c) \quad (1)$$

$$= \arg \max_c \boldsymbol{\theta}_c^T \mathbf{x} \quad (2)$$

Essentially, we are comparing

$$\boldsymbol{\theta}_1^T \mathbf{x}, \boldsymbol{\theta}_2^T \mathbf{x}, \dots, \boldsymbol{\theta}_K^T \mathbf{x} \quad (3)$$

with *one* for each category.

First try

Can we define the following model?

$$p(y = c|x; \theta_c) = \sigma[\underline{\theta_c^T x}]$$

This would *not* work at least for the reason

$$\sum_c p(y = c|x; \theta_c) = \sum_c \sigma[\theta_c^T x] \neq 1$$

as each summand can be any number (independently) between 0 and 1.

But we are close!

We can learn the k linear models jointly to ensure this property holds!

Definition of multinomial logistic regression

Model

For each class C_k , we have a parameter vector $\underline{\theta}_k$ and model the probability of class C_k as

$$p(y = C_k | \mathbf{x}; \theta_1, \dots, \theta_K) = \frac{e^{\theta_k^T \mathbf{x}}}{\sum_{k'} e^{\theta_{k'}^T \mathbf{x}}} \quad \leftarrow \text{This is called softmax function}$$

$$\frac{\exp(\theta_k^T \mathbf{x})}{\sum_{k'} \exp(\theta_{k'}^T \mathbf{x})}$$

Definition of multinomial logistic regression

Model

For each class C_k , we have a parameter vector θ_k and model the probability of class C_k as

$$p(y = C_k | \mathbf{x}; \theta_1, \dots, \theta_K) = \frac{e^{\theta_k^T \mathbf{x}}}{\sum_{k'} e^{\theta_{k'}^T \mathbf{x}}} \quad \leftarrow \text{This is called } \textit{softmax} \text{ function}$$

Decision boundary: assign \mathbf{x} with the label that is the maximum of

$$\arg \max_k P(y = C_k | \mathbf{x}; \theta_1, \dots, \theta_K) \rightarrow \arg \max_k \theta_k^T \mathbf{x}$$

How does the softmax function behave?

Suppose we have

$$\theta_1^T x = 100, \theta_2^T x = 50, \theta_3^T x = -20$$

$\uparrow \quad \equiv \quad \uparrow \quad t$

We could have picked the *winning* class label 1 with certainty according to our classification rule.

Softmax translates these scores into well-formed conditional probabilities

$$p(y=1|x) = \frac{e^{100}}{e^{100} + e^{50} + e^{-20}} < 1$$

$\underbrace{\hspace{10em}}$ $\underbrace{\hspace{10em}}$

- preserves relative ordering of scores
- maps scores to values between 0 and 1 that also sum to 1

Sanity check

Multinomial model reduces to binary logistic regression when $K = 2$

$$\begin{aligned} p(y = C_1 | x; \theta_1, \dots, \theta_2) &= \frac{e^{\theta_1^T x} \times e^{-\theta_2^T x}}{(e^{\theta_1^T x} + e^{\theta_2^T x})} = \frac{1}{1 + e^{-(\theta_1 - \theta_2)^T x}} \\ &= \frac{1}{1 + e^{-\theta^T x}} \end{aligned}$$

Multinomial thus generalizes the (binary) logistic regression to deal with multiple classes.

$$\theta = (\theta_1 - \theta_2)$$

Parameter estimation

Maximize likelihood

$$\begin{aligned}\mathcal{LL}(\theta_1, \theta_2, \dots, \theta_K) &= \log P(\mathcal{D}) \\ &= \sum_n \log P(y_n | \mathbf{x}_n; \theta_1, \theta_2, \dots, \theta_K)\end{aligned}$$

We will change y_n to $\mathbf{y}_n = [y_{n1} \ y_{n2} \ \cdots \ y_{nK}]^T$, a K -dimensional vector using 1-of-K encoding.

$$y_{nk} = \begin{cases} 1 & \text{if } y_n = k \\ 0 & \text{otherwise} \end{cases}$$

If $y_n = 2$, then, $\mathbf{y}_n = [0 \ 1 \ 0 \ 0 \ \cdots \ 0]^T$.

$$\begin{aligned}\sum_n \log P(y_n | \mathbf{x}_n; \theta_1, \dots, \theta_K) &= \sum_n \log \prod_{k=1}^K P(y = C_k | \mathbf{x}_n; \theta_1, \dots, \theta_K)^{y_{nk}} \\ &= \sum_n \sum_k y_{nk} \log P(y = C_k | \mathbf{x}_n; \theta_1, \dots, \theta_K)\end{aligned}$$

Cost function for multinomial logistic regression

Definition: Negative log likelihood

$$\begin{aligned} J(\theta_1, \theta_2, \dots, \theta_K) &= - \sum_n \sum_k y_{nk} \log P(y = C_k | \mathbf{x}_n; \theta_1, \theta_2, \dots, \theta_K) \\ &= - \sum_n \sum_k y_{nk} \log \left(\frac{e^{\theta_k^T \mathbf{x}}}{\sum_{k'} e^{\theta_{k'}^T \mathbf{x}}} \right) \\ &= - \sum_n \sum_k y_{nk} \theta_k^T \mathbf{x} - y_{nk} \log \left(\sum_{k'} e^{\theta_{k'}^T \mathbf{x}} \right) \end{aligned}$$

Properties

- Convex
- Optimization requires numerical procedures, analogous to those used for binary logistic regression

Summary

Two broad approaches to multi-class classification

- Reduction to binary
 - ▶ Easy to implement
 - ▶ Uses binary classification as a black-box (binary classification is well-understood in theory and practice).
 - ▶ Combining predictions can be challenging.
 - ▶ Not theoretically well-justified.
- Single classifier: consider all classes simultaneously.
 - ▶ Directly minimize the empirical risk.
 - ▶ Easy to add constraints and domain knowledge.
 - ▶ Need to re-engineer learning algorithms.

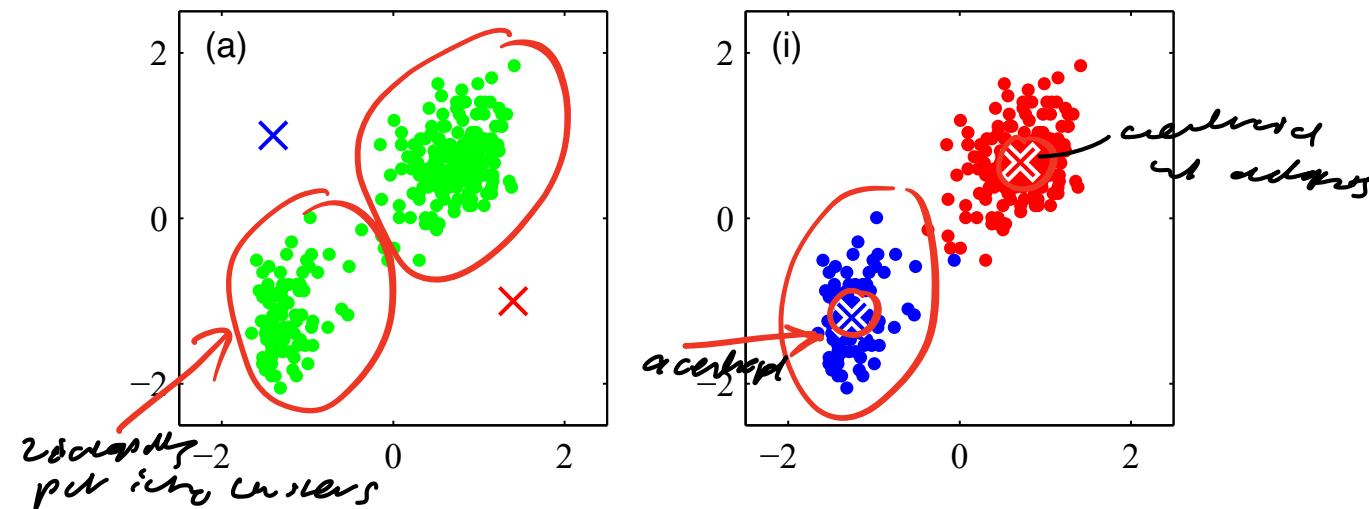
Clustering

Setup Given $\mathcal{D} = \{x_n\}_{n=1}^N$ and K , we want to output

- $\{\mu_k\}_{k=1}^K$: prototypes (or centroids) of clusters
- $A(x_n) \in \{1, 2, \dots, K\}$: the cluster membership, i.e., the cluster ID assigned to x_n

a grouping of datapoints in K groups

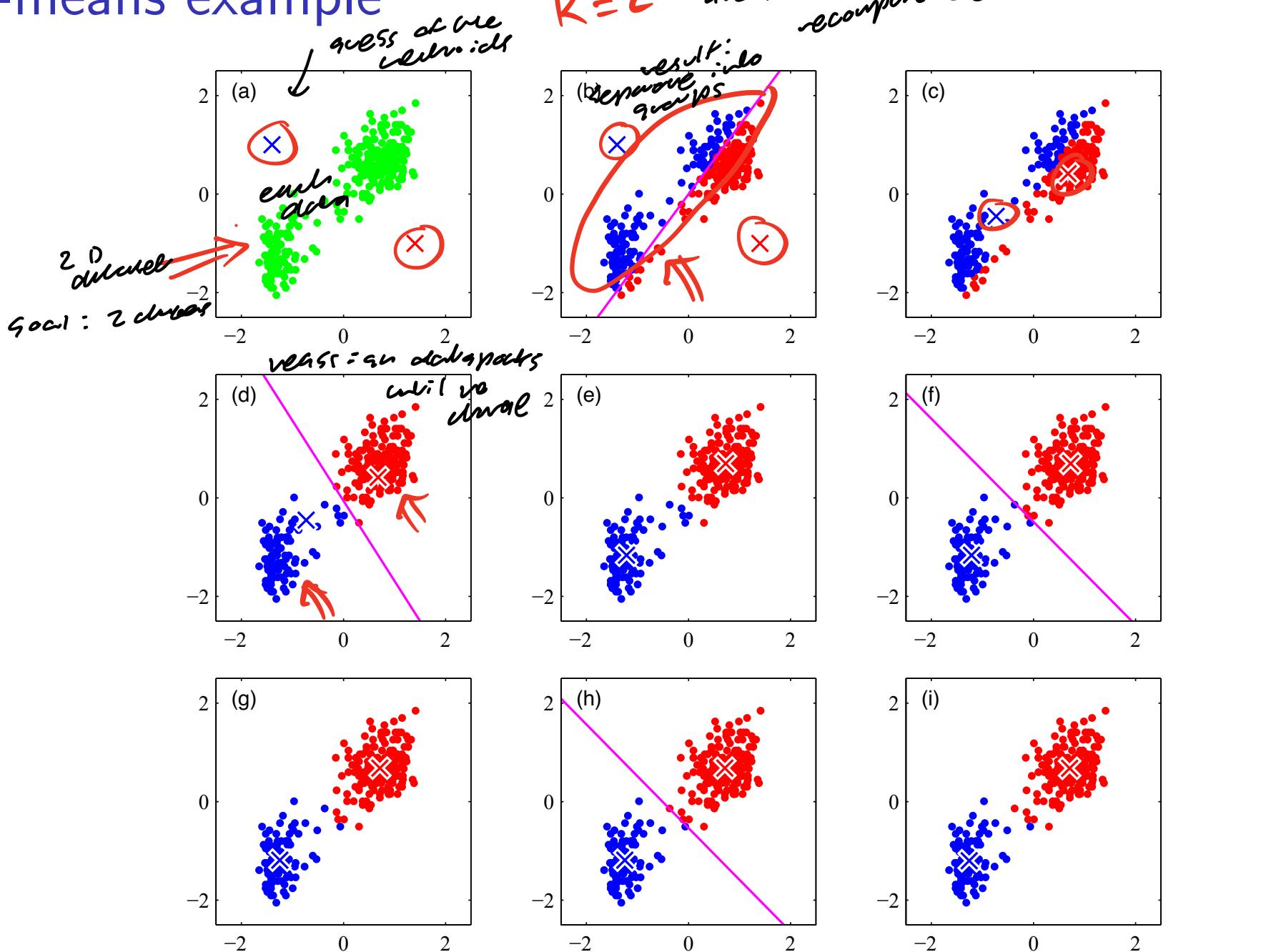
Toy Example Cluster data into two clusters.



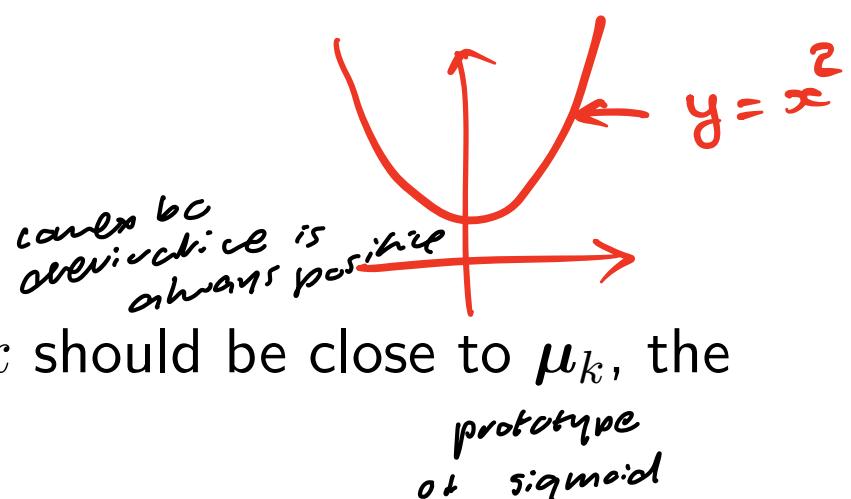
Applications

- Identify communities within social networks
- Find topics in news stories *documents \rightarrow topics*
- Group similar sequences into gene families *biology \rightarrow real sequences*

K-means example



K-means clustering



Intuition Data points assigned to cluster k should be close to μ_k , the prototype.

Distortion measure (clustering objective function, cost function)

2 sets of vars which cluster?
 r_{nk} μ_k no overlap
 indicator variable prototypes assignments squared Euclidean dist.
 where $r_{nk} \in \{0, 1\}$ is an indicator variable

$$J(\{r_{nk}\}, \{\mu_k\}) = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|x_n - \mu_k\|_2^2$$

is it a convex objective function?
 not convex

$$\underline{r_{nk}} = 1 \quad \text{if and only if } A(x_n) = k$$

every problem has a solution? J

r_{nk} μ_k assign a given point to a centroid
 optimize w.r.t respect to cluster

$$J(\{r_{nk}\}, \{\mu_k\}) = \sum_n \sum_k r_{nk} \|x_n - \mu_k\|_2^2$$

indicator var. be closest centroid
 minimize

K-means clustering

K-means objective

$$\operatorname{argmin}_{\{r_{nk}\}, \{\mu_k\}} J(\{r_{nk}\}, \{\mu_k\}) = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|x_n - \mu_k\|_2^2$$

where $r_{nk} \in \{0, 1\}$ is an indicator variable

$$r_{nk} = 1 \quad \text{if and only if } A(x_n) = k$$

- Is a non-convex objective function.
- Minimizing the K-means objective function is NP-hard.

Lloyd's algorithm for minimizing the K-means objective

Often simply called the K-means algorithm

Minimize cost function alternative optimization between $\{r_{nk}\}$ and $\{\mu_k\}$

objective function $J(\{r_{nk}\}, \{\mu_k\}) = \sum_n \sum_k r_{nk} \|x_n - \mu_k\|_2^2$ compute and
update each cluster's
centeroid
= optimal value

- **Step 0** Initialize $\{\mu_k\}$ to some values
- **Step 1** Assume the current value of $\{\mu_k\}$ fixed, minimize J over $\{r_{nk}\}$, which leads to the following cluster assignment rule

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg \min_j \|x_n - \mu_j\|_2^2 \\ 0 & \text{otherwise} \end{cases} \quad \begin{matrix} \text{based on} \\ \text{clusters} \end{matrix}$$

- **Step 2** Assume the current value of $\{r_{nk}\}$ fixed, minimize J over $\{\mu_k\}$, which leads to the following rule to update the prototypes of the clusters

adding up feature vectors *updating centroid:*

$$\Rightarrow \mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}} \quad \begin{matrix} \# \text{ pts assigned to } k \\ \text{since we mean a cluster} \end{matrix}$$

- **Step 3** Stop if the objective function J stays the same or return to Step 1

Remarks

- Prototype $\underline{\mu}_k$ is the mean of data points assigned to the cluster k , hence 'K-means'
objective function also \rightarrow columns improving
- The procedure reduces \underline{J} in both Step 1 and Step 2 and thus makes improvements on each iteration

Application: vector quantization

- Replace data point with associated prototype $\underline{\mu}_k$
- In other words, compress the data points into i) a codebook of all the prototypes; ii) a list of indices to the codebook for the data points
- Lossy compression, especially for small K

*throwing away
original values*

*image data, replace
w/ prototype*



large K



small K

Clustering pixels and vector quantizing them. From left to right: Original image, quantized with large K , medium K , and a small K . Details are missing due to the higher compression (smaller K).

Properties of the K-means algorithm

not exactly minimizing because

$$J(\{x_m\}, \{\mu_k\}) =$$

$$\sum_m \sum_k r_{mk} \|x_m - \mu_k\|_2^2$$

always nonnegative $r_{mk} \in \{0, 1\}$

every step of algorithm, objective func is getting smaller ≥ 0

since ya reach 0, we were setting already to zero

- Does the K-means algorithm converge (i.e., terminate)?
 - ▶ Yes.
- How long does it take to converge ?
 - ▶ In the worst case, exponential in the number of data points.
 - ▶ In practice, very quick.

Properties of the K-means algorithm

How good is the K-means solution?

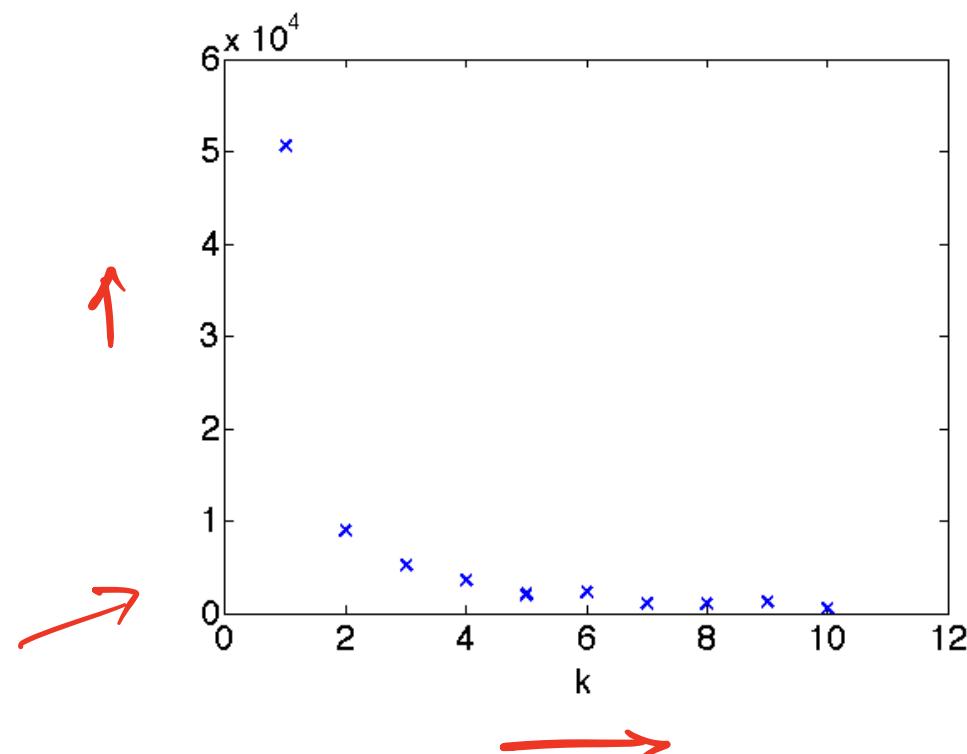
- Converges to a local minimum.
- The solution depends on the initialization.
- In practice, run many times with different initializations and pick the best.
- K-means++ is a neat approximation algorithm that has theoretical guarantees on the final value of the objective.
 - ▶ Still no guarantee that you will reach the global minimum
 - ▶ You are guaranteed to get reasonably close (approximation guarantee on the final value).

Other practical issues

Choosing K

the we value of k for an objective function

- Increasing K will always decrease the optimal value of the K-means objective.
 - ▶ Analogous to overfitting in supervised learning.
- Information criteria that effectively regularize more complex models.



K-medoids

K-means will try & fit to outliers

- K-means is sensitive to outliers.
- In some applications we want the prototypes to be one of the points.
*and prototypes to be
of noise*
- Leads to K-medoids.

K-medoids



- **Step 0** Initialize $\{\mu_k\}$ by randomly selecting K of the N points
- **Step 1** Assume the current value of $\{\mu_k\}$ fixed, assign points to clusters:

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg \min_j \|x_n - \mu_j\|_2^2 \\ 0 & \text{otherwise} \end{cases}$$

pick k

current members

update cluster

- **Step 2** Assume the current value of $\{r_{nk}\}$ fixed, update the prototype of cluster k . In K-medoids, the prototype for a cluster is the data point that is closest to all other data points in the cluster

$$\underline{k^*} = \arg \min_{m: r_{mk}=1} \sum_n r_{nk} \|x_n - x_m\|_2^2$$
$$\mu_k = \underline{x_{k^*}}$$

closest

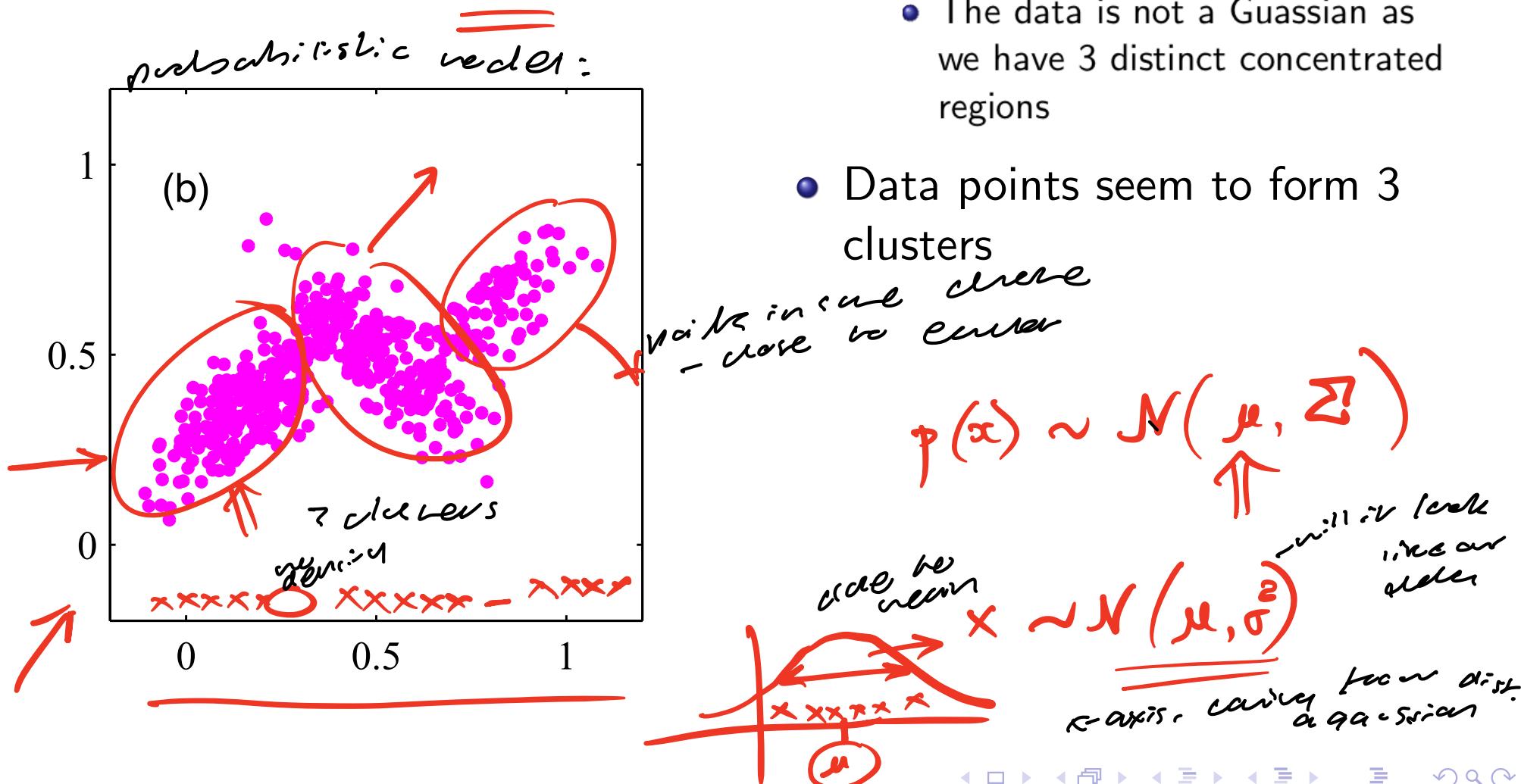
data point

- **Step 3** Stop if the objective function J stays the same or return to Step 1

Probabilistic interpretation of clustering?

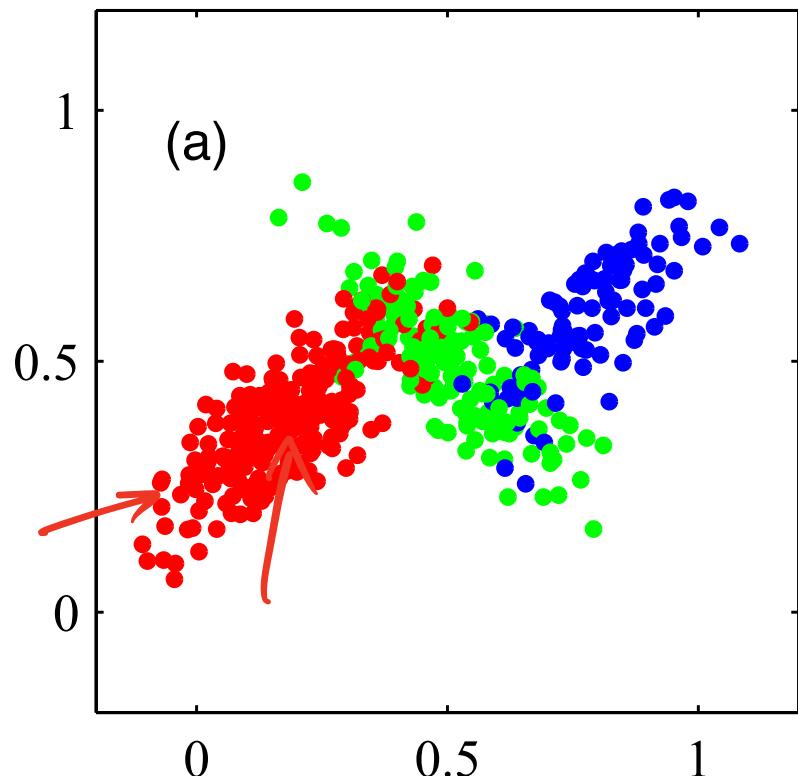
We can impose a probabilistic interpretation of our intuition that points stay close to their cluster centers

- How can we model $p(x)$ to reflect this?



Gaussian mixture models: intuition

for each pair: $RGB?$
 $mean?$
 $covariance?$



We can model each region with a distinct distribution

- Common to use Gaussians, i.e., Gaussian mixture models (GMMs) or mixture of Gaussians (MoGs).
- We don't know *cluster assignments* (label) or *parameters* of Gaussians or *mixture components*!
- We need to learn them all from our *unlabeled* data

$$\mathcal{D} = \{\underline{\underline{x}_n}\}_{n=1}^N$$

Gaussian mixture models: formal definition

A Gaussian mixture model has the following density function for x

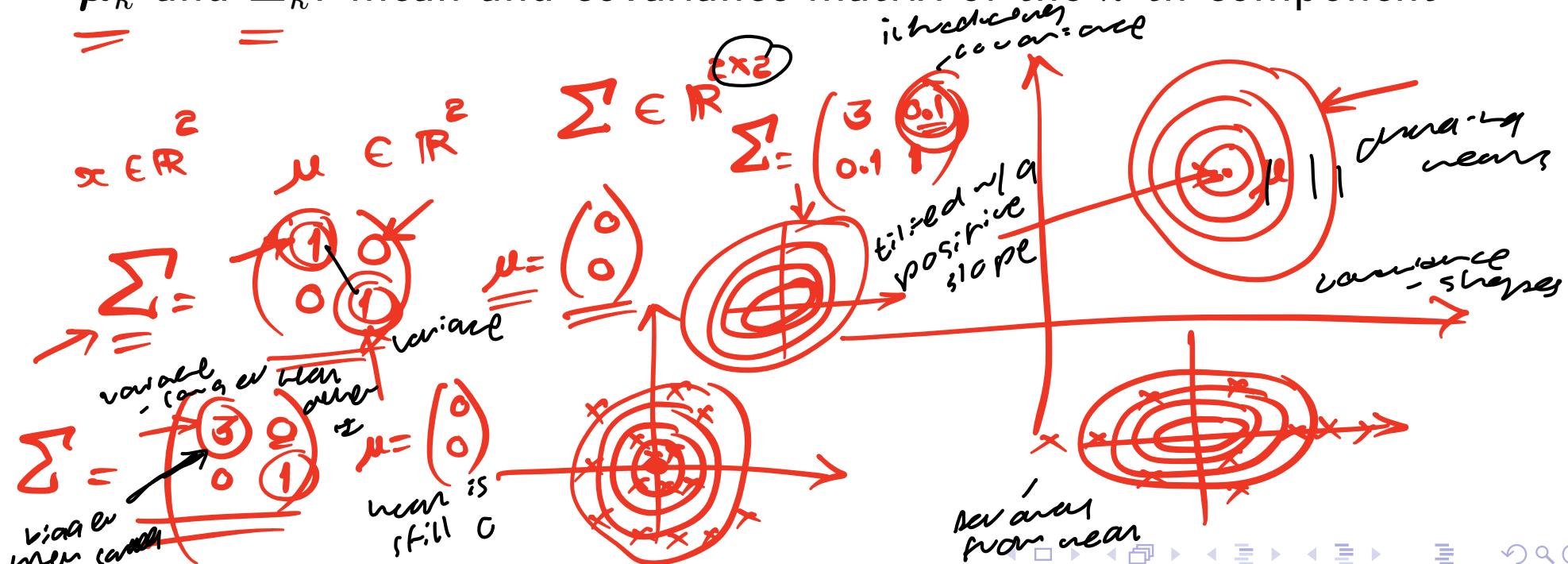
$$p(x) = \sum_{k=1}^K \omega_k N(x | \mu_k, \Sigma_k)$$

of mixed comp

mean, *covar*, $x \in \mathbb{R}^D$

non-spherical

- K : the number of Gaussians — they are called (mixture) components
- μ_k and Σ_k : mean and covariance matrix of the k -th component



Gaussian mixture models: formal definition

A Gaussian mixture model has the following density function for \mathbf{x}

$$p(\mathbf{x}) = \sum_{k=1}^K \omega_k N(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- K : the number of Gaussians — they are called (mixture) components
- $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$: mean and covariance matrix of the k -th component
- ω_k : mixture weights – they represent how much each component contributes to the final distribution. It satisfies two properties:

$$\forall k, \omega_k > 0, \quad \text{and} \quad \sum_k \omega_k = 1$$

. should add up to 1

The properties ensure $p(\mathbf{x})$ is a properly normalized probability density function.

GMM as the marginal distribution of a joint distribution

Consider the following joint distribution

$$p(x, z) = p(z)p(x|z)$$

- conditional P

where z is a discrete random variable taking values between 1 and K .

Denote

$$\omega_k = \underline{p(z=k)}$$

prob of the k event

Now, assume the conditional distributions are Gaussian distributions

$$p(x|z=k) = N(x|\mu_k, \Sigma_k)$$

*cond. P (x|z)
diff mean & diff cov.
gaussian*

$$\begin{aligned} p(x) &= p(x, z=1) + p(x, z=2) \\ &\quad + \dots + p(x, z=K) \\ &= \sum_k p(x, z=k) = \sum_k \underline{p(z=k)} \underline{p(x|z=k)} \end{aligned}$$

GMM as the marginal distribution of a joint distribution

Consider the following joint distribution

$$p(\underline{\underline{x}}, z) = p(z)p(\underline{\underline{x}}|z)$$

where z is a discrete random variable taking values between 1 and K .

Denote

$$\omega_k = p(z = k)$$

Now, assume the conditional distributions are Gaussian distributions

$$p(\underline{\underline{x}}|z = k) = N(\underline{\underline{x}}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Then, the marginal distribution of $\underline{\underline{x}}$ is

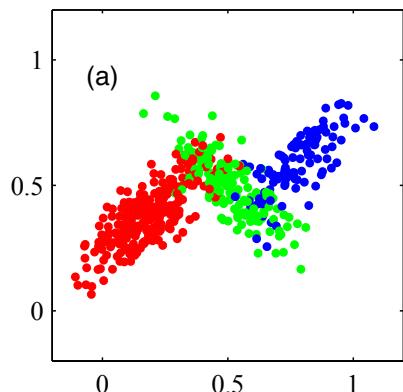
gaussian dist.

$$p(\underline{\underline{x}}) = \sum_{k=1}^K \omega_k N(\underline{\underline{x}}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

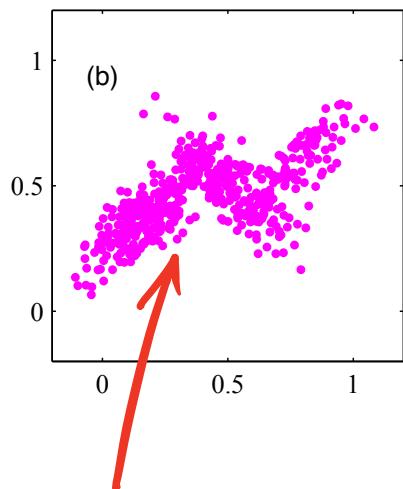
Namely, the Gaussian mixture model

GMMs: example

The conditional distribution between x and z (representing color) are



$$\left\{ \begin{array}{l} \Rightarrow p(\mathbf{x}|z = \text{red}) = N(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \\ \Rightarrow p(\mathbf{x}|z = \text{blue}) = N(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \\ \Rightarrow p(\mathbf{x}|z = \text{green}) = N(\mathbf{x}|\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3) \end{array} \right.$$



The marginal distribution is thus
↑ of observing each color

$$p(\mathbf{x}) = p(\text{red})N(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + p(\text{blue})N(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) + p(\text{green})N(\mathbf{x}|\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$$

and we're to observe colors:

Parameter estimation for Gaussian mixture models

The parameters in GMMs are $\theta = \{\omega_k, \mu_k, \Sigma_k\}_{k=1}^K$. To estimate, consider the simple (and unrealistic) case first.

We have labels z If we assume z is observed for every x , then our estimation problem is easier to solve. Our training data is augmented:

$$\mathcal{D}' = \{\underline{x}_n, z_n\}_{n=1}^N$$

likelihood maximum

z_n denotes the component where x_n comes from. \mathcal{D}' is the *complete* data and \mathcal{D} the *incomplete* data. How can we learn our parameters?

Given \mathcal{D}' , the maximum likelihood estimation of the θ is given by

$$\theta = \arg \max \log P(\mathcal{D}') = \sum \log p(\underline{x}_n, z_n)$$

log likelihood
revise
log of β or each datapoint

Parameter estimation for GMMs: complete data

The *complete* likelihood is decomposable

$$\sum_n \log p(\mathbf{x}_n, z_n) = \sum_n \log p(z_n) p(\mathbf{x}_n | z_n) = \sum_k \sum_{n:z_n=k} \log p(z_n) p(\mathbf{x}_n | z_n)$$

where we have grouped data by its values z_n . Let us introduce a binary variable $\gamma_{nk} \in \{0, 1\}$ to indicate whether $z_n = k$. We then have

$$\sum_n \log p(\mathbf{x}_n, z_n) = \sum_k \sum_n \gamma_{nk} \log p(z = k) p(\mathbf{x}_n | z = k)$$

We use a “dummy” variable z to denote all the possible values cluster assignment values for \mathbf{x}_n

$$\begin{aligned} & \sum_n \sum_z \gamma_{nz} \log p(z = k) p(\mathbf{x}_n | z = k) \\ &= \log(\omega_k) + \log N(\mathbf{x}_n; \mu_k, \Sigma_k) \end{aligned}$$

\mathcal{D}' specifies this value in the complete data setting

Parameter estimation for GMMs: complete data

We now have

$$\sum_n \log p(\mathbf{x}_n, z_n) = \sum_k \sum_n \gamma_{nk} [\log \omega_k + \log N(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)]$$

Regrouping, we have

$$\sum_n \log p(\mathbf{x}_n, z_n) = \sum_k \sum_n \gamma_{nk} \log \omega_k + \sum_k \left\{ \sum_n \gamma_{nk} \log N(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

The term inside the braces depends on k -th component's parameters. It can be shown that the MLE is:

$$\begin{aligned}\omega_k &= \frac{\sum_n \gamma_{nk}}{\sum_k \sum_n \gamma_{nk}}, & \text{new } \gamma_{nk} \in \text{for it} \\ \boldsymbol{\mu}_k &= \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} \mathbf{x}_n \\ \boldsymbol{\Sigma}_k &= \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T\end{aligned}$$

What's the intuition?

Intuition

Since γ_{nk} is binary, the previous solution is nothing but

- For ω_k : count the number of data points whose z_n is k and divide by the total number of data points (note that $\sum_k \sum_n \gamma_{nk} = N$)
- For μ_k : get all the data points whose z_n is k , compute their mean
- For Σ_k : get all the data points whose z_n is k , compute their covariance matrix

This intuition is going to help us to develop an algorithm for estimating θ when we do not know z_n (incomplete data).

val mu Σ

Parameter estimation for GMMs: incomplete data

$$P(z_n=k | \underline{x_n})$$

complete p with $\xrightarrow{\text{z} \rightarrow \text{c}, \text{ given}}$
a:er + n

When z_n is not given, we can guess it via the posterior probability

$$p(z_n = k | x_n) = \frac{p(x_n | z_n = k)p(z_n = k)}{p(x_n)} = \frac{p(x_n | z_n = k)p(z_n = k)}{\sum_{k'=1}^K p(x_n | z_n = k')p(z_n = k')}$$

$$P(y|x) = \frac{P(\underline{x|y}) P(\underline{y})}{P(\underline{x})}$$

(Bayes theorem)

Random variable Random variable

$$\begin{aligned} P(z_n=k | x_n) &= \frac{P(x_n | z_n=k) P(z_n=k)}{P(x_n)} \\ &= \mathcal{N}(x_n; \mu_k, \Sigma_k) \omega_k \end{aligned}$$

Parameter estimation for GMMs: incomplete data

When z_n is not given, we can guess it via the posterior probability

$$p(z_n = k | \mathbf{x}_n) = \frac{p(\mathbf{x}_n | z_n = k)p(z_n = k)}{p(\mathbf{x}_n)} = \frac{p(\mathbf{x}_n | z_n = k)p(z_n = k)}{\sum_{k'=1}^K p(\mathbf{x}_n | z_n = k')p(z_n = k')}$$

To compute the posterior probability, we need to know the parameters θ !

Let's pretend we know the value of the parameters so we can compute the posterior probability.

How is that going to help us?

Estimation with soft γ_{nk}

We define $\gamma_{nk} = p(z_n = k | \mathbf{x}_n)$

- Recall that $\underline{\gamma_{nk}}$ was binary.
- Now it's a "soft" assignment of \mathbf{x}_n to k -th component
- Each \mathbf{x}_n is assigned to a component fractionally according to
 $\underline{p(z_n = k | \mathbf{x}_n)}$

Parameter estimation for GMMs: incomplete data

With the soft assignment γ_{nk} plugged into the complete data log likelihood, we now have:

$$\sum_k \sum_n \underline{\gamma_{nk}} [\log \omega_k + \log N(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)]$$

Regrouping, we have

$$\sum_k \sum_n \gamma_{nk} \log \omega_k + \sum_k \left\{ \sum_n \gamma_{nk} \log N(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

We now get the same expression for the MLE as before!

$$\underline{\underline{\omega_k}} = \frac{\sum_n \gamma_{nk}}{\sum_k \sum_n \gamma_{nk}}, \quad \underline{\underline{\boldsymbol{\mu}_k}} = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} \mathbf{x}_n$$

$$\underline{\underline{\boldsymbol{\Sigma}_k}} = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T$$

But remember, we're 'cheating' by using θ to compute γ_{nk} !

Iterative procedure

$$\underline{\theta} = \underline{(\underline{\omega_k}, \underline{\mu_k}, \underline{\Sigma_k})}_{k=1}^K.$$

We can alternate between estimating $\underline{\gamma_{nk}}$ and using the estimated $\underline{\gamma_{nk}}$ to compute the parameters (same idea as with K -means!)

- Step 0: initialize $\underline{\theta}$ with some values (random or otherwise)
- Step 1: compute $\underline{\gamma_{nk}}$ using the current $\underline{\theta}$
- Step 2: update $\underline{\theta}$ using the just computed $\underline{\gamma_{nk}}$
- Step 3: go back to Step 1

*a answer
update
current*

Questions:

- Is this procedure reasonable, i.e., are we optimizing a sensible criteria?
- Will this procedure converge?

The answers lie in the *EM algorithm* — a powerful procedure for model estimation with unknown data.

Parameter estimation for GMMs: complete data

GMM Parameters

$$\theta = \{\omega_k, \mu_k, \Sigma_k\}_{k=1}^K$$

Complete Data: We (unrealistically) assume z is observed for every x ,

$$\mathcal{D}' = \{x_n, z_n\}_{n=1}^N$$

$\stackrel{x \leftarrow \mathcal{X}}{=}$ $\stackrel{z \leftarrow \mathcal{Z}}{=}$

observed w/ φ

MLE: Maximize the **complete likelihood**

$$\theta = \arg \max \log P(\mathcal{D}') = \sum_n \log \overbrace{p(x_n, z_n)}^{\substack{\text{MLE - maximizing} \\ \text{, learned}}}$$

Leads to closed-form solution!

Parameter estimation for GMMs: Incomplete data

GMM Parameters

$$\theta = \{\omega_k, \mu_k, \Sigma_k\}_{k=1}^K$$

Incomplete Data

Our data contains observed and unobserved random variables, and hence is incomplete

- Observed: $\mathcal{D} = \{x_n\}$ ** observed (visible)*
- Unobserved (hidden): $\{z_n\}$ ** unobserved (latent)*

Goal Obtain the maximum likelihood estimate of θ :

$$\begin{aligned}\hat{\theta} &= \arg \max \underline{\ell(\theta)} = \arg \max \log \underline{P(\mathcal{D})} = \arg \max \sum_n \log \underline{p(x_n | \theta)} \\ &= \arg \max \sum_n \log \sum_{z_n} p(x_n, z_n | \theta)\end{aligned}$$

observed value

unseen latent

The objective function $\ell(\theta)$ is called the *incomplete* log-likelihood.



Issue with Incomplete log-likelihood

No simple way to optimize the incomplete log-likelihood

expensive numerical

Expectation-Maximization (EM) algorithm provides a strategy for iteratively optimizing this function

Two steps as they apply to GMM:

- E-step: 'guess' values of the z_n using existing values of θ
- M-step: solve for new values of θ given imputed values for z_n
(maximize complete likelihood!)

re step maximization

E-step: Soft cluster assignments

$$0 \leq \gamma_{nk} \leq 1$$

We define $\underline{\gamma_{nk}}$ as $p(z_n = k | \mathbf{x}_n, \theta)$

- This is the posterior distribution of z_n given \mathbf{x}_n and θ
- Recall that in complete data setting γ_{nk} was binary
- Now it's a "soft" assignment of \mathbf{x}_n to k -th component, with \mathbf{x}_n assigned to each component with some probability

Given $\theta = \{\omega_k, \mu_k, \Sigma_k\}_{k=1}^K$, we can compute γ_{nk} using Bayes theorem:

$$\begin{aligned}\gamma_{nk} &= p(z_n = k | \mathbf{x}_n) \\ &= \frac{p(\mathbf{x}_n | z_n = k)p(z_n = k)}{p(\mathbf{x}_n)} \\ &= \frac{p(\mathbf{x}_n | z_n = k)p(z_n = k)}{\sum_{k'=1}^K p(\mathbf{x}_n | z_n = k')p(z_n = k')} = \frac{\mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)\omega_k}{\sum_{k'=1}^K \mathcal{N}(\mathbf{x}_n | \mu_{k'}, \Sigma_{k'})\omega_{k'}}\end{aligned}$$

Bayes' rule.

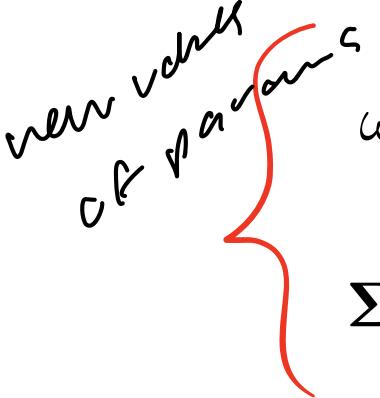
M-step: Maximize complete likelihood

Recall definition of complete likelihood from earlier:

$$\sum_n \log p(\mathbf{x}_n, z_n) = \sum_k \sum_n \gamma_{nk} \log \omega_k + \sum_k \left\{ \sum_n \gamma_{nk} \log \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

Previously γ_{nk} was binary, but now we define $\gamma_{nk} = p(z_n = k | \mathbf{x}_n)$ (E-step)

We get the same simple expression for the MLE as before!


$$\omega_k = \frac{\sum_n \gamma_{nk}}{\sum_k \sum_n \gamma_{nk}}, \quad \boldsymbol{\mu}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} \mathbf{x}_n$$
$$\boldsymbol{\Sigma}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T$$

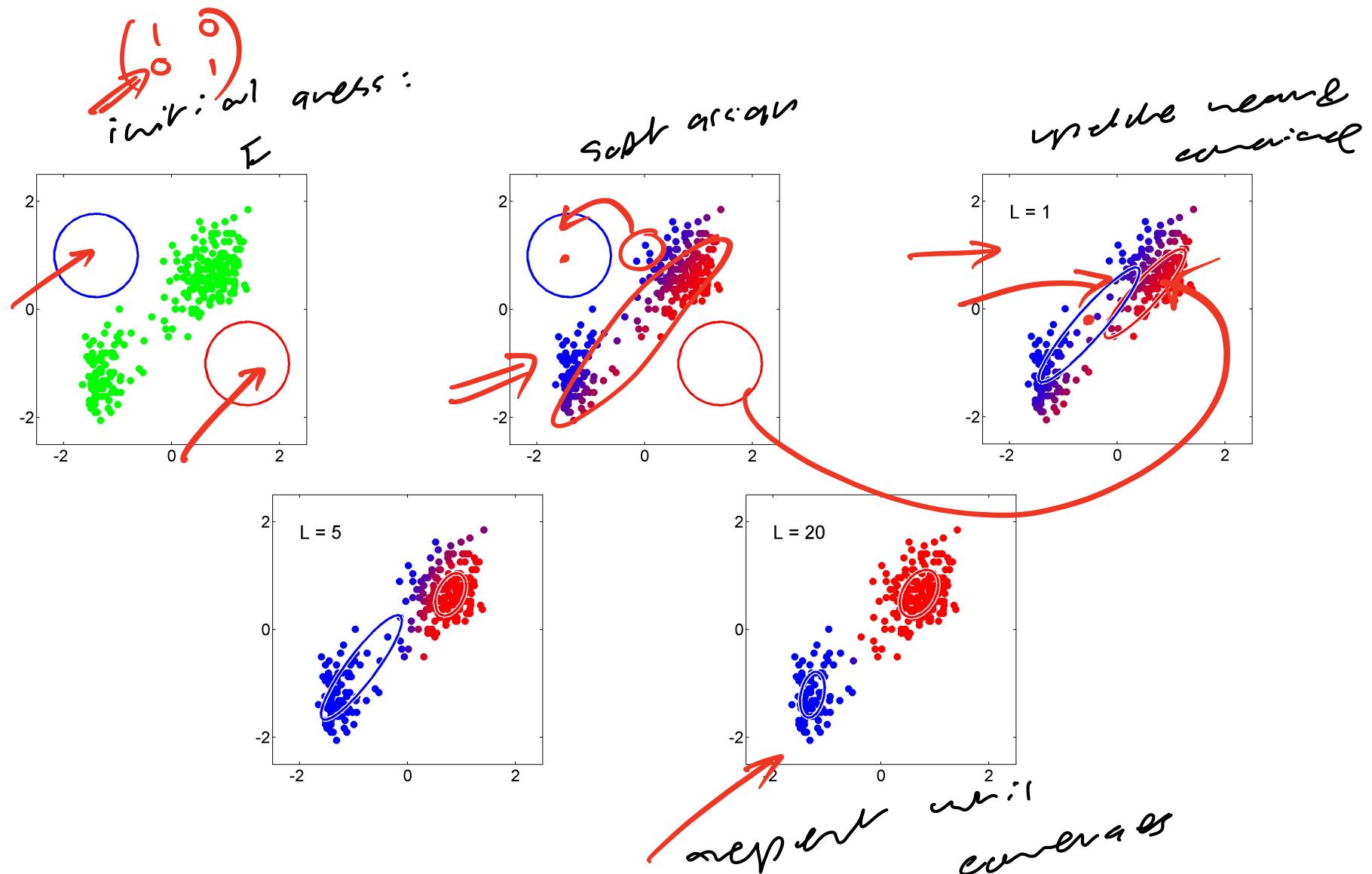
Intuition: Each point now contributes some fractional component to each of the parameters, with weights determined by γ_{nk}

EM procedure for GMM

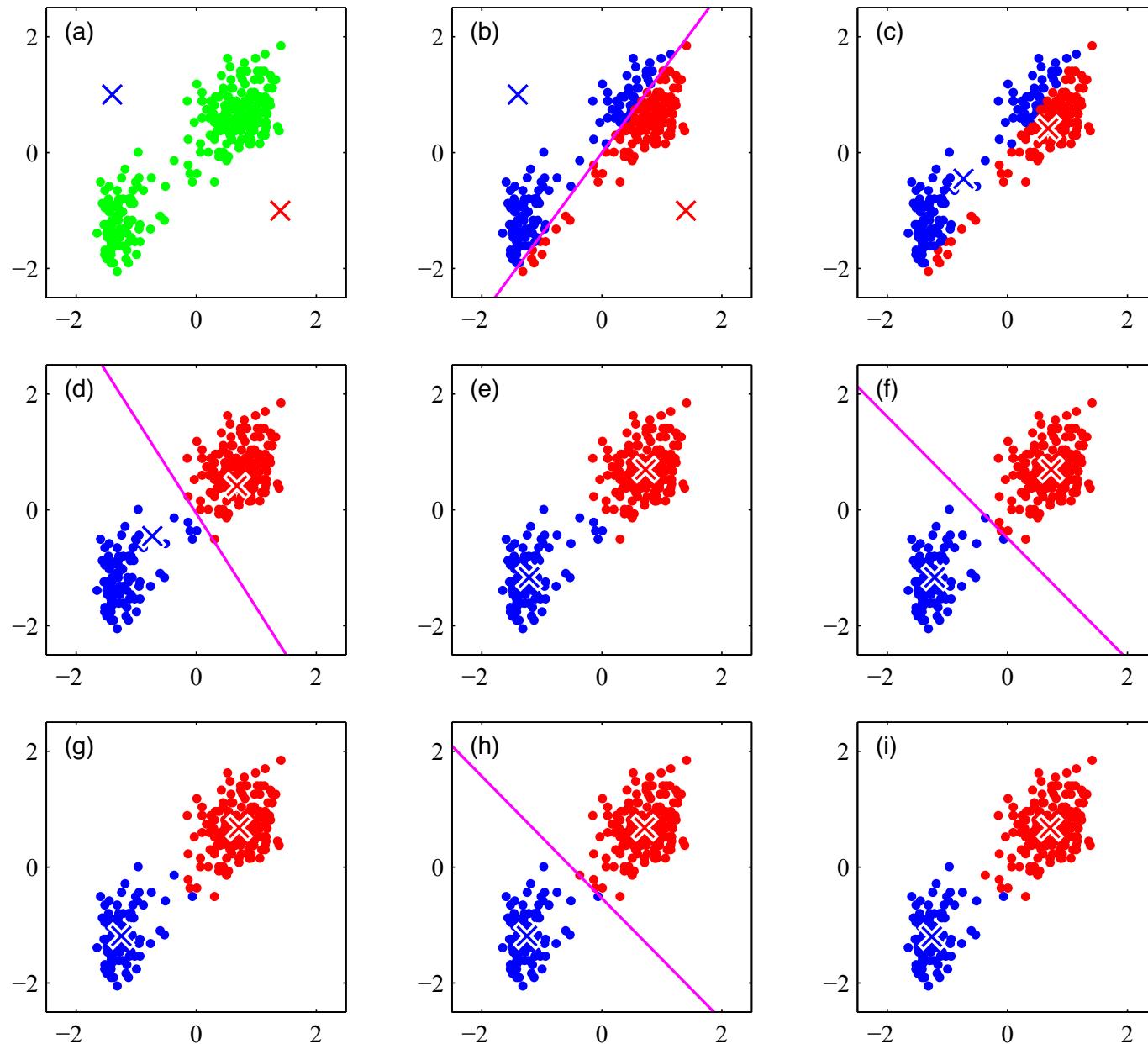
Alternate between estimating γ_{nk} and estimating θ

- Initialize θ with some values (random or otherwise)
- Repeat \curvearrowright
 - ▶ E-Step: Compute γ_{nk} using the current θ
 - ▶ M-Step: Update θ using the γ_{nk} we just computed
- Until Convergence

Example of GMM



Compare to K-means example



EM procedure for GMM

Questions to be answered next

- How does GMM relate to K -means?
- Is this procedure reasonable, i.e., are we optimizing a sensible criterion?
- Will this procedure converge?

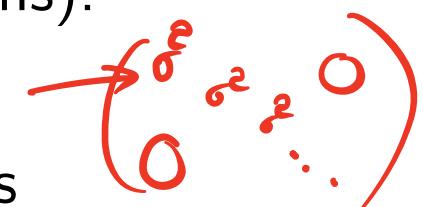
practical

GMMs and K-means

GMMs provide probabilistic interpretation for K-means

GMMs reduce to K-means under the following assumptions (in which case EM for GMM parameter estimation simplifies to K-means):

- Assume all mixture weights ω_k are equal
- Assume all Gaussians have $\sigma^2 \mathbf{I}$ covariance matrices
- Further assume $\sigma \rightarrow 0$, so we only need to estimate μ_k , i.e., means
- GMMs are more general model.



K-means is often called “hard” GMM or GMMs is called “soft” K-means

The posterior γ_{nk} provides a probabilistic assignment for x_n to cluster k

EM algorithm

- The estimates of the parameter θ in each iteration increase the likelihood.
- EM algorithm converges but only to a local optimum.

Summary

Clustering

- Group similar instances
- K-means
 - ▶ Minimize a cost function that measures the sum of squared distances from the cluster prototypes.
 - ▶ Iterative algorithm for minimizing the cost function.
- Variants: K-medoids
- Probabilistic interpretation of K-means: Gaussian Mixture Model
- Can define a number of mixture models for other kinds of data.
- Probabilistic interpretation: GMMs
 - ▶ Generalization of K-means
 - ▶ Estimation using an iterative EM algorithm.

witten notes:
measure similarity

↔ renewed